ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.20 SQL-программирование

Направление подготовки *09.03.01 Информатика и вычислительная техника*

Профиль образовательной программы *Автоматизированные системы обработки информации и управления*

Форма обучения *заочная*

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция №1 (2 часа).

Тема: «Эффективное выполнение запросов для извлечения данных.»

1.1.1 Вопросы лекции:

1.Эффективное выполнение запросов для извлечения данных.
2. Синтаксис оператора SELECT.
3.Построение условий выбора данных с применением операторов сравнения, логических операторов и логических связок.

1.1.2 Краткое содержание вопросов:

1.Эффективное выполнение запросов для извлечения данных.
2. Синтаксис оператора SELECT.
3.Построение условий выбора данных с применением операторов сравнения, логических операторов и логических связок.

Предложение SELECT

Оператор SELECT – один из наиболее важных и самых распространенных операторов SQL. Он позволяет производить выборки данных из таблиц и преобразовывать к нужному виду полученные результаты. Будучи очень мощным, он способен выполнять действия, эквивалентные операторам реляционной алгебры, причем в пределах единственной выполняемой команды. При его помощи можно реализовать сложные и громоздкие условия отбора данных из различных таблиц.

Оператор SELECT – средство, которое полностью абстрагировано от вопросов представления данных, что помогает сконцентрировать внимание на проблемах доступа к данным.

Примеры его использования наглядно демонстрируют один из основополагающих принципов больших (промышленных) СУБД: средства хранения данных и доступа к ним отделены от средств представления данных. Операции над данными производятся в масштабе наборов данных, а не отдельных записей.

Оператор SELECT имеет следующий формат:

SELECT [ALL | DISTINCT] {*|[имя_столбца
[AS новое_имя]]} [,...n]
FROM имя_таблицы [[AS] псевдоним] [,...n]
[WHERE <условие_поиска>]
[GROUPBYимя_столбца [,...n]]
[HAVING <критерии выбора групп>]
[ORDER BY имя_столбца [,...n]]

Оператор SELECT определяет поля (столбцы), которые будут входить в результат выполнения запроса. В списке они разделяются запятыми и приводятся в такой очередности, в какой должны быть представлены в результате запроса. Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки. Символом * можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен. Если обрабатывается ряд таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. Имя_таблицы.Имя_поля.

Предложение FROM

Предложение FROM задает имена таблиц и представлений, которые содержат поля, перечисленные в операторе SELECT. Необязательный параметр псевдонима – это сокращение, устанавливаемое для имени таблицы.
Обработка элементов оператора SELECT выполняется в следующей последовательности:
 FROM – определяются имена используемых таблиц;
 WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;
 GROUP BY – образуются группы строк , имеющих одно и то же значение в указанном столбце;
 HAVING – фильтруются группы строк объекта в соответствии с указанным условием;
 SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;
 ORDER BY – определяется упорядоченность результатов выполнения операторов.

Порядок предложений и фраз в операторе SELECT не может быть изменен. Только два предложения SELECT и FROM являются обязательными, все остальные могут быть опущены. SELECT – закрытая операция: результат запроса к таблице представляет собой другую таблицу. Существует множество вариантов записи данного оператора, что иллюстрируется приведенными ниже примерами.

Пример 4.1.*Составить список сведений о всех клиентах.*
SELECT * FROM Клиент
Пример 4.1. Список сведений о всех клиентах.

Параметр WHERE определяет критерий отбора записей из входного набора. Но в таблице могут присутствовать повторяющиеся записи (дубликаты). Предикат ALL задает включение в выходной набор всех дубликатов, отобранных по критерию WHERE. Нет необходимости указывать ALL явно, поскольку это значение действует по умолчанию.

Пример 4.2. Составить список всех фирм.
SELECT ALL Клиент.Фирма FROM Клиент
Или (что эквивалентно)
SELECT Клиент.Фирма FROM Клиент

Пример 4.2. Список всех фирм.
Результат выполнения запроса может содержать дублирующиеся значения, поскольку в отличие от операций реляционной алгебры оператор SELECT не исключает повторяющихся значений при выполнении выборки данных.

Предикат DISTINCT следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие дублирующие записи в выбранных полях. Значения для каждого из приведенных в инструкции SELECT полей должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор.

Причиной ограничения в применении DISTINCT является то обстоятельство, что его использование может резко замедлить выполнение запросов.

Откорректированный пример 4.2 выглядит следующим образом:
SELECT DISTINCT Клиент.Фирма
FROM Клиент

Предложение WHERE

С помощью WHERE-параметра пользователь определяет, какие блоки данных из приведенных в списке FROM таблиц появятся в результате запроса. За ключевым словом WHERE следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса.
Существует пять основных типов условий поиска (или предикатов):
 Сравнение: сравниваются результаты вычисления одного выражения с результатами вычисления другого.
 Диапазон: проверяется, попадает ли результат вычисления выражения в заданный диапазон значений.
 Принадлежность множеству: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.
 Соответствие шаблону: проверяется, отвечает ли некоторое строковое значение заданному шаблону.
 Значение NULL: проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).

Сравнение
В языке SQL можно использовать следующие операторы сравнения: = – равенство; < – меньше; > – больше; <= – меньше или равно; >= – больше или равно; <> – не равно.

Пример 4.3. Показать все операции отпуска товаров объемом больше 20.
```
SELECT * FROM Сделка
WHERE Количество>20
```
Пример 4.3. Операции отпуска товаров объемом больше 20.

Более сложные предикаты могут быть построены с помощью логических операторов AND, OR или NOT, а также скобок, используемых для определения порядка вычисления выражения.
Вычисление выражения в условиях выполняется по следующим правилам:
 Выражение вычисляется слева направо.
 Первыми вычисляются подвыражения в скобках.
 Операторы NOT выполняются до выполнения операторов AND и OR.
 Операторы AND выполняются до выполнения операторов OR.
Для устранения любой возможной неоднозначности рекомендуется использовать скобки.

Пример 4.4. Вывести список товаров, цена которых больше или равна 100 и меньше или равна 150.
```
SELECT Название, Цена
        FROM Товар
        WHERE Цена>=100 And Цена<=150
```
Пример 4.4. Список товаров, цена которых больше или равна 100 и меньше или равна 150.

Пример 4.5. Вывести список клиентов из Москвы или из Самары.
```
SELECT Фамилия, ГородКлиента
        FROM Клиент
        WHERE ГородКлиента="Москва" Or
                ГородКлиента="Самара"
```
Пример 4.5. Список клиентов из Москвы или из Самары.

Диапазон
Оператор BETWEEN используется для поиска значения внутри некоторого интервала,

определяемого своими минимальным и максимальным значениями. При этом указанные значения включаются в условие поиска.

Пример 4.6. Вывести список товаров, цена которых лежит в диапазоне от 100 до 150 (запрос эквивалентен примеру 4.4).

```
SELECT Название, Цена
        FROM Товар
        WHERE Цена Between 100 And 150
```

Пример 4.6. Список товаров, цена которых лежит в диапазоне от 100 до 150.

При использовании отрицания NOT BETWEEN требуется, чтобы проверяемое значение лежало вне границ заданного диапазона.

Пример 4.7. Вывести список товаров, цена которых не лежит в диапазоне от 100 до 150.

```
SELECT Товар.Название, Товар.Цена
        FROM Товар
        WHERE Товар.Цена Not Between 100 And 150
```

Или (что эквивалентно)

```
SELECT Товар.Название, Товар.Цена
        FROM Товар
        WHERE (Товар.Цена<100) OR (Товар.Цена>150)
```

Пример 4.7. Список товаров, цена которых не лежит в диапазоне от 100 до 150.

Принадлежность множеству

Оператор IN используется для сравнения некоторого значения со списком заданных значений, при этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке. При помощи оператора IN может быть достигнут тот же результат, что и в случае применения оператора OR, однако оператор IN выполняется быстрее.

Пример 4.8. Вывести список клиентов из Москвы или из Самары (запрос эквивалентен примеру 4.5).

```
SELECT Фамилия, ГородКлиента
        FROM Клиент
        WHERE ГородКлиентain ("Москва", "Самара")
```

Пример 4.8. Список клиентов из Москвы или из Самары

NOT IN используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Пример 4.9. Вывести список клиентов, проживающих не в Москве и не в Самаре.

```
SELECT Фамилия, ГородКлиента
        FROM Клиент
        WHERE ГородКлиента
                Notin ("Москва","Самара")
```

Пример 4.9. Список клиентов, проживающих не в Москве и не в Самаре.

Соответствие шаблону

С помощью оператора LIKE можно выполнять сравнение выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

 Символ % – вместо этого символа может быть подставлено любое количество произвольных символов.

 Символ _ заменяет один символ строки.
 [ ] – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях.
 [^] – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример 4.10. Найти клиентов, у которых в номере телефона вторая цифра – 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
        FROM Клиент
        WHERE Клиент.ТелефонLike "_4%"
```

Пример 4.10. Выборка клиентов, у которых в номере телефона вторая цифра – 4.

Пример 4.11. Найти клиентов, у которых в номере телефона вторая цифра – 2 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
        FROM Клиент
        WHERE Клиент.ТелефонLike "_[2,4]%"
```

Пример 4.11. Выборка клиентов, у которых в номере телефона вторая цифра – 2 или 4.

Пример 4.12. Найти клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
        FROM Клиент
        WHERE Клиент.ТелефонLike "_[2-4]%"
```

Пример 4.12. Выборка клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.

Пример 4.13. Найти клиентов, у которых в фамилии встречается слог "ро".

```
SELECT Клиент.Фамилия
        FROM Клиент
        WHERE Клиент.ФамилияLike "%ро%"
```

Пример 4.13. Выборка клиентов, у которых в фамилии встречается слог "ро".

Значение NULL

Оператор IS NULL используется для сравнения текущего значения со значением NULL – специальным значением, указывающим на отсутствие любого значения. NULL – это не то же самое, что знак пробела (пробел – допустимый символ) или ноль (0 – допустимое число). NULL отличается и от строки нулевой длины (пустой строки).

Пример 4.14. Найти сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

```
SELECT Фамилия, Телефон
        FROM Клиент
        WHERE Телефон IsNull
```

Пример 4.14. Выборка сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

IS NOT NULL используется для проверки присутствия значения в поле.

Пример 4.15. Выборка клиентов, у которых есть телефон (поле Телефон содержит какое-либо значение).

```
SELECT Клиент.Фамилия, Клиент.Телефон
        FROM Клиент
        WHERE Клиент.Телефон Is Not Null
```

Пример 4.15. Найти клиентов, у которых есть телефон (поле Телефон содержит какое-либо значение).

Предложение ORDER BY

В общем случае строки в результирующей таблице SQL-запроса никак не упорядочены. Однако их можно требуемым образом отсортировать, для чего в оператор SELECT помещается фраза ORDER BY, которая сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом ORDER BY через запятую. Способ сортировки задается ключевым словом, указываемым в рамках параметра ORDER BY следом за названием поля, по которому выполняется сортировка. По умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом ASC. Для выполнения сортировки в обратной последовательности необходимо после имени поля, по которому она выполняется, указать *ключевое слово* DESC. Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример 4.16.Вывести список клиентов в алфавитном порядке.

```
SELECT Клиент.Фамилия, Клиент.Фирма
        FROM Клиент
        ORDER BY Клиент.Фамилия
```

Пример 4.16. Список клиентов в алфавитном порядке.

Во фразе ORDER BY может быть указано и больше одного элемента. Главный (первый) ключ сортировки определяет общую упорядоченность строк результирующей таблицы. Если во всех строках результирующей таблицы значения главного ключа сортировки являются уникальными, нет необходимости использовать дополнительные ключи сортировки. Однако, если значения главного ключа не уникальны, в результирующей таблице будет присутствовать несколько строк с одним и тем же значением старшего ключа сортировки. В этом случае, возможно, придется упорядочить строки с одним и тем же значением главного ключа по какому-либо дополнительному ключу сортировки.

Пример 4.17. Вывести список фирм и клиентов. Названия фирм упорядочить в алфавитном порядке, имена клиентов в каждой фирме отсортировать в обратном порядке.

```
SELECT Клиент.Фирма, Клиент.Фамилия
        FROM Клиент
        ORDER BY Клиент.Фирма, Клиент.Фамилия DESC
```

Пример 4.17. Список фирм и клиентов. Названия фирм в алфавитном порядке, имена клиентов в каждой фирме в обратном порядке.

1. 2 Лекция №2 (2 часа).

Тема: «Понятие представлений»

1.2.1 Вопросы лекции:

1.Понятие представлений.
2.Роль представлений в вопросах безопасности данных.

3.Процесс управления представлениями: создание, изменение, применение, удаление представлений.

1.2.2 Краткое содержание вопросов:

1.Понятие представлений.
2.Роль представлений в вопросах безопасности данных.
3.Процесс управления представлениями: создание, изменение, применение, удаление представлений.

Определение представления

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. Обычные таблицы относятся к **базовым,** т.е. содержащим данные и постоянно находящимся на устройстве хранения информации. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц. Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности. Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются. **Представление** - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. Результат выполнения этого запроса в каждый момент времени становится **содержанием** представления. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

У СУБД есть две возможности реализации представлений. Если его определение простое, то система формирует каждую запись представления по мере необходимости, постепенно считывая исходные данные из базовых таблиц. В случае сложного определения СУБД приходится сначала выполнить такую операцию, как материализация представления, т.е. сохранить информацию, из которой состоит представление, во временной таблице. Затем система приступает к выполнению пользовательской команды и формированию ее результатов, после чего временная таблица удаляется.

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. Для хранения представления используется только оперативная память. В отличие от других объектов базы данных представление не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого представления.
Создания и изменения представлений в стандарте языка и реализации в MS SQL Server совпадают и представлены следующей командой:

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

Рассмотрим назначение основных параметров.

По умолчанию имена столбцов в представлении соответствуют именам столбцов в исходных таблицах. Явное указание имени столбца требуется для вычисляемых столбцов или при объединении нескольких таблиц, имеющих столбцы с одинаковыми именами. Имена столбцов перечисляются через запятую, в соответствии с порядком их следования в представлении.

Параметр WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования. Если при определении представления необходимо скрыть имена исходных таблиц и столбцов, а также алгоритм объединения данных, необходимо применить этот аргумент.

Параметр WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через представление, на соответствие критериям, определенным в операторе SELECT. Это означает, что не допускается выполнение изменений, которые приведут к исчезновению строки из представления. Такое случается, если для представления установлен горизонтальный фильтр и изменение данных приводит к несоответствию строки установленным фильтрам. Использование аргумента WITH CHECK OPTION гарантирует, что сделанные изменения будут отображены в представлении. Если пользователь пытается выполнить изменения, приводящие к исключению строки из представления, при заданном аргументе WITH CHECK OPTION сервер выдаст сообщение об ошибке и все изменения будут отклонены.

Пример 10.1. Показать в представлении клиентов из Москвы.

Создание представления:

```
CREATE VIEW view1 AS
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва'
```

Пример 10.1. Представление клиентов из Москвы.

Выборка данных из представления:

```
SELECT * FROM view1
```

Обращение к представлению осуществляется с помощью оператора SELECT как к обычной таблице. Представление можно использовать в команде так же, как и любую другую таблицу. К представлению можно строить запрос, модифицировать его (если оно отвечает определенным требованиям), соединять с другими таблицами. Содержание представления не фиксировано и обновляется каждый раз, когда на него ссылаются в команде. Представления значительно расширяют возможности управления данными. В частности, это прекрасный способ разрешить доступ к информации в таблице, скрыв часть данных. Так, в примере 10.1 представление просто ограничивает доступ пользователя к данным таблицы Клиент, позволяя видеть только часть значений.

Выполним команду:

```
INSERT INTO view1 VALUES (12,'Петров', 'Самара')
```

Это допустимая команда в представлении, и строка будет добавлена с помощью представления *view1* в таблицу Клиент. Однако, когда информация будет добавлена, строка исчезнет из представления, поскольку название города отлично от Москвы. Иногда такой подход может стать проблемой, т.к. данные уже находятся в таблице, но пользователь их не видит и не в состоянии выполнить их удаление или модификацию. Для исключения подобных моментов служит WITH CHECK OPTION в определении представления. Фраза размещается в определении представления, и все команды модификации будут подвергаться проверке.

```
ALTER VIEW view1
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва'
WITH CHECK OPTION
```

Пример 10.2. Создание представления с проверкой команд модификации.

Для такого представления вышеупомянутая вставка значений будет отклонена

системой.

Таким образом, представление может изменяться командами модификации DML, но фактически модификация воздействует не на само представление, а на базовую таблицу.

Представление удаляется командой:

DROP VIEW имя_представления [,...n]

Обновление данных в представлениях

Не все представления в SQL могут быть модифицированы. 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;
- по возможности не применяет в своем определении подзапросы;
- не использует константы или выражения значений среди выбранных полей вывода;
- в просмотр должен быть включен каждый столбец таблицы, имеющий атрибут NOT NULL;
- 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. Различия между модифицируемыми представлениями и представлениями, предназначенными только для чтения, не случайны. Цели, для которых их используют, различны. С модифицируемыми представлениями в основном обходятся точно так же, как и с базовыми таблицами. Фактически, пользователи не могут даже осознать, является ли объект, который они запрашивают, базовой таблицей или представлением, т.е. прежде всего это средство защиты для сокрытия конфиденциальных или не относящихся к потребностям данного пользователя частей таблицы. Представления в режиме <только для чтения> позволяют получать и форматировать данные более рационально. Они создают целый арсенал сложных запросов, которые можно выполнить и повторить снова, сохраняя полученную информацию. Результаты этих запросов могут затем использоваться в других запросах, что позволит избежать сложных предикатов и снизить вероятность ошибочных действий.

```
CREATE VIEW view2 AS
SELECT Клиент.Фамилия, Клиент.Фирма,
        Сделка.Количество
FROM Клиент INNER JOIN Сделка
ON Клиент.КодКлиента=Сделка.КодКлиента
```

Пример 10.3. Немодифицируемое представление с данными из разных таблиц.

```
CREATE VIEW view3(Тип, Общ_остаток) AS
SELECT Тип, Sum(Остаток)
FROM Товар
GROUP BY Тип
```

Пример 10.4. Немодифицируемое представление с группировкой и итоговыми функциями.

Обычно в представлениях используются имена, полученные непосредственно из имен полей основной таблицы. Однако иногда необходимо дать столбцам новые имена, например, в случае итоговых функций или вычисляемых столбцов.

```
CREATE VIEW view4(Код, Название,
        Тип, Цена, Налог) AS
SELECT КодТовара, Название,
        Тип, Цена, Цена*0.05 FROM Товар
```

Пример 10.5. Модифицируемое представление с вычислениями.

Преимущества и недостатки представлений

Механизм представления - мощное средство СУБД, позволяющее скрыть реальную структуру БД от некоторых пользователей за счет определения представлений. Любая реализация представления должна гарантировать, что состояние представляемого отношения точно соответствует состоянию данных, на которых определено это представление. Обычно вычисление представления производится каждый раз при его использовании. Когда представление создается, информация о нем записывается в каталог БД под собственным именем. Любые изменения в данных адекватно отобразятся в представлении - в этом его отличие от очень похожего на него запроса к БД. В то же время запрос представляет собой как бы <мгновенную фотографию> данных и при изменении последних запрос к БД необходимо повторить. Наличие представлений в БД необходимо для обеспечения логической независимости данных.

Если система обеспечивает физическую независимость данных, то изменения в физической структуре БД не влияют на работу пользовательских программ. Логическая независимость подразумевает тот факт, что при изменении логической структуры данных влияние на пользовательские программы также не оказывается, а значит, система должна уметь решать проблемы, связанные с ростом и реструктуризацией БД. Очевидно, что с увеличением количества данных, хранимых в БД, возникает необходимость ее расширения за счет добавления новых атрибутов или отношений - это называется **ростом БД**. **Реструктуризация** данных подразумевает сохранение той же самой информации, но изменяется ее расположение, например, за счет перегруппировки атрибутов в отношениях. Предположим, некоторое отношение в силу каких-либо причин необходимо разделить на два. Соединение полученных отношений в представлении воссоздает исходное отношение, а у пользователя складывается впечатление, что никакой реструктуризации не производилось. Помимо решения проблемы реструктуризации представление можно применять для просмотра одних и тех же данных разными пользователями и в различных вариантах. С помощью представлений пользователь имеет возможность ограничить объем данных для удобства работы. Наконец, механизм представлений позволяет скрыть служебные данные, не интересные пользователям.

В случае выполнения СУБД на отдельно стоящем персональном компьютере использование представлений обычно имеет целью лишь упрощение структуры запросов к базе данных. Однако в случае многопользовательской сетевой СУБД представления играют ключевую роль в определении структуры базы данных и организации защиты информации. Рассмотрим основные преимущества применения представлений в подобной среде.

Независимость от данных

С помощью представлений можно создать согласованную, неизменную картину структуры базы данных, которая будет оставаться стабильной даже в случае изменения формата исходных таблиц (например, добавления или удаления столбцов, изменения связей, разделения таблиц, их реструктуризации или переименования).

Если в таблицу добавляются или из нее удаляются не используемые в представлении столбцы, то изменять определение этого представления не потребуется.

Если структура исходной таблицы переупорядочивается или таблица разделяется, можно создать представление, позволяющее работать с виртуальной таблицей прежнего формата. В случае разделения исходной таблицы, прежний формат может быть виртуально воссоздан с помощью представления, построенного на основе соединения вновь созданных таблиц - конечно, если это окажется возможным. Последнее условие можно обеспечить с помощью помещения во все вновь созданные таблицы первичного ключа прежней таблицы.

Актуальность

Изменения данных в любой из таблиц базы данных, указанных в определяющем запросе, немедленно отображается на содержимом представления.

Повышение защищенности данных

Права доступа к данным могут быть предоставлены исключительно через ограниченный набор представлений, содержащих только то подмножество данных, которое необходимо пользователю. Подобный подход позволяет существенно ужесточить контроль за доступом отдельных категорий пользователей к информации в базе данных.

Снижение стоимости

Представления позволяют упростить структуру запросов за счет объединения данных из нескольких таблиц в единственную виртуальную таблицу. В результате многотабличные запросы сводятся к простым запросам к одному представлению.

Дополнительные удобства

Создание представлений может обеспечивать пользователей дополнительными удобствами - например, возможностью работы только с действительно нужной частью данных. В результате можно добиться максимального упрощения той модели данных, которая понадобится каждому конечному пользователю.

Возможность настройки

Представления являются удобным средством настройки индивидуального образа базы данных. В результате одни и те же таблицы могут быть предъявлены пользователям в совершенно разном виде.

Обеспечение целостности данных

Если в операторе CREATE VIEW будет указана фраза WITH CHECK OPTION, то СУБД станет осуществлять контроль за тем, чтобы в исходные таблицы базы данных не была введена ни одна из строк, не удовлетворяющих предложению WHERE в определяющем запросе. Этот механизм гарантирует целостность данных в представлении. Практика ограничения доступа некоторых пользователей к данным посредством создания специализированных представлений, безусловно, имеет значительные преимущества перед предоставлением им прямого доступа к таблицам базы данных. Однако использование представлений в среде SQL не лишено *недостатков.*

Ограниченные возможности обновления

В некоторых случаях представления не позволяют вносить изменения в содержащиеся в них данные.

Структурные ограничения

Структура представления устанавливается в момент его создания. Если определяющий запрос представлен в форме SELECT * FROM_, то символ * ссылается на все столбцы, существующие в исходной таблице на момент создания представления. Если впоследствии в исходную таблицу базы данных добавятся новые столбцы, то они не появятся в данном представлении до тех пор, пока это представление не будет удалено и вновь создано.

Снижение производительности

Использование представлений связано с определенным снижением производительности. В одних случаях влияние этого фактора совершенно незначительно, тогда как в других оно может послужить источником существенных проблем. Например, представление, определенное с помощью сложного многотабличного запроса, может потребовать значительных затрат времени на обработку, поскольку при его разрешении потребуется выполнять соединение таблиц всякий раз, когда понадобится доступ к данному представлению. Выполнение разрешения представлений связано с использованием дополнительных вычислительных ресурсов.

2. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ
ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1 Лабораторная работа №1 ЛР-1 (2 часа).

Тема: «Определение структурированного языка запросов SQL»

2.1.1 Задание для работы:

1.Определение структурированного языка запросов SQL.
2. Реляционная база данных, СУБД.
3.Классификация команд SQL.

2.1.2 Краткое описание проводимого занятия:

Типы команд SQL
Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим набором предложений. Язык SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ. Основные категории команд языка SQL предназначены для выполнения различных функций, включая построение объектов базы данных и манипулирование ими, начальную загрузку данных в таблицы, обновление и удаление существующей информации, выполнение запросов к базе данных, управление доступом к ней и ее общее администрирование.
Основные категории команд языка SQL:
 DDL – язык определения данных;
 DML – язык манипулирования данными;
 DQL – язык запросов;
 DCL – язык управления данными;
 команды администрирования данных;
 команды управления транзакциями
Определение структур базы данных (DDL)
Язык определения данных (DataDefinitionLanguage, DDL) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными*командамиязыка DDL*являютсяследующие: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.
Манипулирование данными (DML)
Язык манипулирования данными (DataManipulationLanguage, DML) используется для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд: INSERT, UPDATE, DELETE.
Выборка данных (DQL) Язык запросов DQL наиболее известен пользователям реляционной базы данных, несмотря на то, что он включает всего одну команду SELECT. Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования запросов к реляционной базе данных.
Язык управления данными (DCL - DataControlLanguage)
Команды управления данными позволяют управлять доступом к информации, находящейся внутри базы данных. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными следующие: GRANT, REVOKE.
Команды администрирования данных
С помощью команд администрирования данных пользователь осуществляет контроль за

выполняемыми действиями и анализирует операции базы данных; они также могут оказаться полезными при анализе производительности системы.

Команды управления транзакциями

Существуют следующие команды, позволяющие управлять транзакциями базы данных: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Преимущества языка SQL

Язык SQL является основой многих СУБД, т.к. отвечает за физическое структурирование и запись данных на диск, а также за чтение данных с диска, позволяет принимать SQL-запросы от других компонентов СУБД и пользовательских приложений. Таким образом, SQL – мощный инструмент, который обеспечивает пользователям, программам и вычислительным системам доступ к информации, содержащейся в реляционных базах данных.

Основные **достоинства языка SQL** заключаются в следующем:

- **стандартность** – как уже было сказано, использование языка SQL в программах стандартизировано международными организациями;
- **независимость** от конкретных СУБД – все распространенные СУБД используют SQL, т.к. реляционную базу данных можно перенести с одной СУБД на другую с минимальными доработками;
- **возможность переноса** с одной вычислительной системы на другую – СУБД может быть ориентирована на различные вычислительные системы, однако приложения, созданные с помощью SQL, допускают использование как для локальных БД, так и для крупных многопользовательских систем;
- **реляционная основа языка** – SQL является языком реляционных БД, поэтому он стал популярным тогда, когда получила широкое распространение реляционная модель представления данных. Табличная структура реляционной БД хорошо понятна, а потому язык SQL прост для изучения;
- **возможность создания** интерактивных запросов – SQL обеспечивает пользователям немедленный доступ к данным, при этом в интерактивном режиме можно получить результат запроса за очень короткое время без написания сложной программы;
- **возможность программного** доступа к БД – язык SQL легко использовать в приложениях, которым необходимо обращаться к базам данных. Одни и те же операторы SQL употребляются как для интерактивного, так и программного доступа, поэтому части программ, содержащие обращение к БД, можно вначале проверить в интерактивном режиме, а затем встраивать в программу;
- **обеспечение различного** представления данных – с помощью SQL можно представить такую структуру данных, что тот или иной пользователь будет видеть различные их представления. Кроме того, данные из разных частей БД могут быть скомбинированы и представлены в виде одной простой таблицы, а значит, представления пригодны для усиления защиты БД и ее настройки под конкретные требования отдельных пользователей;
- **возможность** динамического изменения и расширения структуры БД – язык SQL позволяет манипулировать структурой БД, тем самым обеспечивая гибкость с точки зрения приспособленности БД к изменяющимся требованиям предметной области;
- **поддержка архитектуры** клиент-сервер – SQL – одно из лучших средств для реализации приложений на платформе клиент-сервер. SQL служит связующим звеном между взаимодействующей с пользователем клиентской системой и серверной системой, управляющей БД, позволяя каждой из них сосредоточиться на выполнении своих функций.

Любой язык работы с базами данных должен предоставлять пользователю следующие **возможности:**

- создавать базы данных и таблицы с полным описанием их структуры;

 выполнять основные операции манипулирования данными, в частности, вставку, модификацию и удаление данных из таблиц;
 выполнять простые и сложные запросы, осуществляющие преобразование данных.

Кроме того, язык работы с базами данных должен решать все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и синтаксис его команд – достаточно просты и доступны для изучения. И наконец, он должен быть универсальным, т.е. отвечать некоторому признанному стандарту, что позволит использовать один и тот же синтаксис и структуру команд при переходе от одной СУБД к другой.

Язык SQL удовлетворяет практически всем этим требованиям.
Язык SQL является примером языка с трансформирующейся ориентацией, или же языка, предназначенного для работы с таблицами с целью преобразования входных данных к требуемому выходному виду. Он включает только команды определения и манипулирования данными и не содержит каких-либо команд управления ходом вычислений. Подобные задачи должны решаться либо с помощью языков программирования или управления заданиями, либо интерактивно, в результате действий, выполняемых самим пользователем.

По причине подобной незавершенности в плане организации вычислительного процесса **язык SQL может использоваться двумяспособами**.

Первый предусматривает интерактивную работу, заключающуюся во вводе пользователем с терминала отдельных SQL-операторов. *Второй* состоит во внедрении SQL-операторов в программы на процедурных языках.

Язык SQL относительно прост в изучении. Поскольку это не процедурный язык, в нем необходимо указывать, какая информация должна быть получена, а не как ее можно получить.

Язык SQL может использоваться широким кругом специалистов, включая администраторов баз данных, прикладных программистов и множество других конечных пользователей.

Язык SQL – первый и пока единственный стандартный язык для работы с базами данных, который получил достаточно широкое распространение. Практически все крупнейшие разработчики СУБД в настоящее время создают свои продукты с использованием языка SQL либо с SQL-интерфейсом. В него сделаны огромные инвестиции как со стороны разработчиков, так и со стороны пользователей. Он стал частью архитектуры приложений, является стратегическим выбором многих крупных и влиятельных организаций.

Язык SQL используется в других стандартах и даже оказывает влияние на разработку иных стандартов как инструмент определения (например, стандарт RemoteDataAccess, RDA). Создание языка способствовало не только выработке необходимых
теоретических основ, но и подготовке успешно реализованных технических решений. Это особенно справедливо в отношении оптимизации запросов, методов распределения данных и реализации средств защиты. Начали появляться специализированные реализации языка, предназначенные для новых рынков: системы управления обработкой транзакций (OnLineTransactionProcessing, OLTP) и системы оперативной аналитической обработки или системы поддержки принятия решений (OnLineAnalytical Processing, OLAP). Уже известны планы дальнейших расширений стандарта, включающих поддержку распределенной обработки, объектно-ориентированного программирования, расширений пользователей и мультимедиа.

Запись SQL-операторов
Для успешного изучения языка SQL необходимо привести краткое описание структуры

SQL-операторов и нотации, которые используются для определения формата различных конструкций языка. Оператор SQL состоит из зарезервированных слов, а также из слов, определяемых пользователем. Зарезервированные слова являются постоянной частью языка SQL и имеют фиксированное значение. Их следует записывать в точности так, как это установлено, нельзя разбивать на части для переноса с одной строки на другую. Слова, определяемые пользователем, задаются им самим (в соответствии с синтаксическими правилами) и представляют собой идентификаторы или имена различных объектов базы данных. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных и являются именами таблиц, представлений, столбцов и других объектов базы данных. Символы, которые могут использоваться в создаваемых пользователем идентификаторах языка SQL, должны быть определены как набор символов. Стандарт SQL задает набор символов, который используется по умолчанию, – он включает строчные и прописные буквы латинского алфавита (A-Z, a-z), цифры (0-9) и символ подчеркивания (_). На формат идентификатора накладываются следующие ограничения:

- идентификатор может иметь длину до 128 символов;
- идентификатор должен начинаться с буквы;
- идентификатор не может содержать пробелы.

<идентификатор>::=<буква>
{<буква>|<цифра>}[,...n]

Большинство компонентов языка не чувствительны к регистру. Поскольку у языка SQL свободный формат, отдельные SQL-операторы и их последовательности будут иметь более читаемый вид при использовании отступов и выравнивания.

Язык, в терминах которого дается описание языка SQL, называется **метаязыком.** Синтаксические определения обычно задают с помощью специальной металингвистической символики, называемой Бэкуса-Науэра формулами (БНФ). Прописные буквы используются для записи зарезервированных слов и должны указываться в операторах точно так, как это будет показано. Строчные буквы употребляются для записи слов, определяемых пользователем. Применяемые в нотации БНФ символы и их обозначения показаны в таблице.

Таблица 1.1.

Символ	Обозначение
::=	Равно по определению
|	Необходимость выбора одного из нескольких приведенных значений
<…>	Описанная с помощью метаязыка структура языка
{…}	Обязательный выбор некоторой конструкции из списка
[…]	Необязательный выбор некоторой конструкции из списка
[,…n]	Необязательная возможность повторения конструкции от нуля до нескольких раз

2.2 Лабораторная работа №2 ЛР-2 (2 часа).

Тема: «Построение нетривиальных запросов»

2.2.1 Задание для работы:

1.Построение нетривиальных запросов.
2. Способ построения подзапросов, возвращающих множественные и единичные значения с использованием операторов EXISTS, ALL, ANY.

2.2.2 Краткое описание проводимого занятия:

Понятие подзапроса

Часто невозможно решить поставленную задачу путем одного запроса. Это особенно актуально, когда при использовании условия поиска в предложении **WHERE** значение, с которым надо сравнивать, заранее не определено и должно быть вычислено в момент выполнения оператора SELECT. В таком случае приходят на помощь законченные операторы SELECT, внедренные в тело другого оператора SELECT. Внутренний подзапрос представляет собой также оператор SELECT, а кодирование его предложений подчиняется тем же правилам, что и основного оператора SELECT.

Внешний оператор SELECT использует результат выполнения внутреннего оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут быть помещены непосредственно после оператора сравнения (=, <, >, <=, >=, <>) в предложения WHERE и HAVING внешнего оператора SELECT – они получают название подзапросов или вложенных запросов. Кроме того, внутренние операторы SELECT могут применяться в операторах INSERT, UPDATE и DELETE.

Подзапрос – это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки.

К подзапросам применяются следующие правила и ограничения:

 фраза ORDER BY не используется, хотя и может присутствовать во внешнем подзапросе;

 список в предложении SELECT состоит из имен отдельных столбцов или составленных из них выражений – за исключением случая, когда в подзапросе присутствует ключевое слово EXISTS;

 по умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении FROM. Однако допускается ссылка и на столбцы таблицы, указанной во фразе FROM внешнего запроса, для чего применяются квалифицированные имена столбцов (т.е. с указанием таблицы);

 если подзапрос является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операции.

Существует два типа подзапросов:

 Скалярный подзапрос возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение.

 Табличный подзапрос возвращает множество значений, т.е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.

Использование подзапросов, возвращающих единичное значение

Пример 7.1. Определить дату продажи максимальной партии товара.

```
SELECT Дата, Количество
        FROM Сделка
        WHERE Количество=(SELECT Max(Количество) FROM Сделка)
```

Пример 7.1. Определение даты продажи максимальной партии товара.

Во вложенном подзапросе определяется максимальное количество товара. Во внешнем подзапросе – дата, для которой количество товара оказалось равным максимальному. Необходимо отметить, что нельзя прямо использовать предложение *WHERE Количество=Max(Количество)*, поскольку применять обобщающие функции в предложениях WHERE запрещено. Для достижения желаемого результата следует создать подзапрос, вычисляющий максимальное значение количества, а затем использовать его во внешнем операторе SELECT, предназначенном для выборки дат сделок, где количество товара совпало с максимальным значением.

Пример 7.2. Определить даты сделок, превысивших по количеству товара среднее значение и указать для этих сделок превышение над средним уровнем.

```
SELECT Дата, Количество,
Количество-(SELECT Avg(Количество)
            FROM Сделка) AS Превышение
FROM Сделка
WHERE Количество>
            (SELECT Avg(Количество)
      FROM Сделка)
```

Пример 7.2. Определение даты сделок, превысивших по количеству товара среднее значение и указать для этих сделок превышение над средним уровнем.

В приведенном примере результат подзапроса, представляющий собой среднее значение количества товара по всем сделкам вообще, используется во внешнем операторе SELECT как для вычисления отклонения количества от среднего уровня, так и для отбора сведений о датах.

Пример 7.3. Определить клиентов, совершивших сделки с максимальным количеством товара.

```
SELECT Клиент.Фамилия
FROM Клиент INNERJOIN Сделка
ON Клиент.КодКлиента=Сделка.КодКлиента
WHERE Сделка.Количество=
      (SELECT Max(Сделка.Количество)
      FROM Сделка)
```

Пример 7.3. Определение клиентов, совершивших сделки с максимальным количеством товара.

Здесь показан пример использования подзапроса при выборке данных из разных таблиц.

Пример 7.4. Определить клиентов, в сделках которых количество товара отличается от максимального не более чем на 10%.

```
SELECT Клиент.Фамилия,
      Сделка.Количество
FROM Клиент INNER JOIN Сделка
ON Клиент.КодКлиента=
      Сделка.КодКлиента
WHERE Сделка.Количество>=0.9*
      (SELECT Max(Сделка.Количество)
      FROM Сделка)
```

2.3 Лабораторная работа №3 ЛР-3 (2 часа).

Тема: «Запросы модификации данных»

2.3.1 Задание для работы:

1. Запросы модификации данных в реляционной таблице.
2. Целостность данных.
3. Целостность сущностей и ссылочная целостность

2.3.2 Краткое описание проводимого занятия:

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью запросов действия пользователь может добавить, удалить или обновить блоки данных.

Существует три вида запросов действия:

 INSERT INTO – запрос добавления;
 DELETE – запрос удаления;
 UPDATE – запрос обновления.

Запрос добавления

Оператор INSERT применяется для добавления записей в таблицу. Формат оператора:

<оператор_вставки>::=INSERT INTO <имя_таблицы>
[(имя_столбца [,...n])]
{VALUES (значение[,...n])|
<SELECT_оператор>}

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Первая форма оператора INSERT с параметром VALUES предназначена для вставки единственной строки в указанную таблицу. Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в определенном порядке, установленном при создании таблицы. Если в операторе INSERT указывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение NULL, за исключением тех случаев, когда при описании столбца использовался параметр DEFAULT. *Список значений должен следующим образом соответствовать списку столбцов:*

 количество элементов в обоих списках должно быть одинаковым;
 должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.
 типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример 8.1. Добавить в таблицу ТОВАР новую запись.

```
INSERT INTO Товар (Название, Тип, Цена)
        VALUES(" Славянский ", " шоколад ", 12)
```

Пример 8.1. Добавление в таблицу ТОВАР новой записи.

Если столбцы таблицы ТОВАР указаны в полном составе и в том порядке, в котором они

перечислены при создании таблицы ТОВАР, оператор можно упростить.
INSERT INTO Товар VALUES (" Славянский ",
" шоколад ", 12)

Вторая форма оператора INSERT с параметром SELECT позволяет скопировать множество строк из одной таблицы в другую. Предложение SELECT может представлять собой любой допустимый оператор SELECT. Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора SELECT, применимы и в этом случае.
Поскольку оператор SELECT в общем случае возвращает множество записей, то оператор INSERT в такой форме приводит к добавлению в таблицу аналогичного числа новых записей.

Пример 8.2. Добавить в итоговую таблицу сведения об общей сумме ежемесячных продаж каждого наименования товара.

```
INSERT INTO Итог
        (Название, Месяц, Стоимость )
SELECT Товар.Название, Month(Сделка.Дата)
        AS Месяц, Sum(Товар.Цена*Сделка.Количество)
        AS Стоимость
FROM Товар INNER JOIN Сделка
        ON Товар.КодТовара= Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Пример 8.3. Удалить все прошлогодние сделки.

```
DELETE
FROM Сделка
WHERE Year(Сделка.Дата)=Year(GETDATE())-1
```

Пример 8.3. Удаление всех прошлогодних сделок.

В приведенном примере условие отбора формируется с учетом года (функция Year) от текущей даты (функция GETDATE( )).

Запрос обновления

Оператор UPDATE применяется для изменения значений в группе записей или в одной записи указанной таблицы.
Формат оператора:

```
<оператор_изменения> ::=
        UPDATE имя_таблицы SET имя_столбца=
                <выражение>[,...n]
[WHERE <условие_отбора>]
```

Параметр имя_таблицы – это либо имя таблицы базы данных, либо имя обновляемого представления. В предложении SET указываются имена одного и более столбцов, данные в которых необходимо изменить. Предложение WHERE является необязательным.
Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы.
Если предложение WHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию отбора. Выражение представляет собой новое значение соответствующего столбца и должно быть совместимо с ним по типу данных.

2.4 Лабораторная работа №4, 5 ЛР-4, 5 (4 часа).

Тема: «Определение функций пользователя, примеры их создания и использования»

2.4.1 Задание для работы:

1. Определение функций пользователя, примеры их создания использования.
2. Типы функций.
3. Встроенные функции языка SQL.

2.4.2 Краткое описание проводимого занятия:

Понятие функции пользователя

При реализации на языке SQL сложных алгоритмов, которые могут потребоваться более одного раза, сразу встает вопрос о сохранении разработанного кода для дальнейшего применения. Эту задачу можно было бы реализовать с помощью хранимых процедур, однако их архитектура не позволяет использовать процедуры непосредственно в выражениях, т.к. они требуют промежуточного присвоения возвращенного значения переменной, которая затем и указывается в выражении. Естественно, подобный метод применения программного кода не слишком удобен. Многие разработчики уже давно хотели иметь возможность вызова разработанных алгоритмов непосредственно в выражениях.

Возможность создания пользовательских функций была предоставлена в среде MS SQL Server 2000. В других реализациях SQL в распоряжении пользователя имеются только встроенные функции, которые обеспечивают выполнение наиболее распространенных алгоритмов: поиск максимального или минимального значения и др.

Функции пользователя представляют собой самостоятельные объекты базы данных, такие, например, как хранимые процедуры или триггеры. Функция пользователя располагается в определенной базе данных и доступна только в ее контексте.

В SQL Server имеются следующие классы функций пользователя:

 Scalar – функции возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;

 Inline– функции содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения типа данных TABLE;

 Multi-statement – функции также возвращают пользователю значение типа данных TABLE, содержащее набор данных, однако в теле функции находится множество команд SQL (INSERT, UPDATE и т.д.).

Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения функции.

Пользовательские функции сходны с хранимыми процедурами, но, в отличие от них, могут применяться в запросах так же, как и системные встроенные функции.

Пользовательские функции, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а пользовательские функции способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

Функции Scalar

Создание и изменение функции данного типа выполняется с помощью команды:

<определение_скаляр_функции>::=

{CREATE | ALTER } FUNCTION [владелец.]

имя_функции

([{ @имя_параметраскаляр_тип_данных

[=default]}[,...n]])

```
RETURNS скаляр_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING}
        [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN скаляр_выражение
END
```

Рассмотрим назначение параметров команды.
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. Дополнительные параметры, с которыми должна быть создана функция, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания функции, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы функции. Кроме того, в теле функции может выполняться обращение к различным объектам базы данных, а потому изменение или удаление соответствующих объектов может привести к нарушению работы функции.

Чтобы избежать этого, требуется запретить внесение изменений, указав при создании этой функции ключевое слово SCHEMABINDING. 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.

Пример 11.1. Создать и применить функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. Владелец функции – пользователь с именем user1.

```
CREATE FUNCTION
        user1.sales(@data DATETIME)
RETURNS INT
AS
BEGIN
DECLARE @c INT
SET @c=(SELECT SUM(количество)
        FROM Сделка
        WHERE дата=@data)
RETURN (@c)
END
```

2.5 Лабораторная работа №6, 7 ЛР-6, 7 (4 часа).

Тема: «Триггеры»

2.5.1 Задание для работы:

1.Триггеры: создание и применение.

2.Определение триггера, область его использования, место и роль триггера в обеспечении целостности данных.

3.Типы триггеров.

4.Программирование триггера.

2.5.2 Краткое описание проводимого занятия:

Определение триггера в стандарте языка SQL

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные.

Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером. Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать. В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. *С помощью триггеров достигаются следующие цели:*

 проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;

 выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
 накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
 поддержка репликации.

Основной формат команды CREATE TRIGGER показан ниже:

<Определение_триггера>::=
CREATE TRIGGER имя_триггера
BEFORE | AFTER <триггерное_событие>
ON <имя_таблицы>
[REFERENCING
<список_старых_или_новых_псевдонимов>]
[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. Время 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).

Обозначение <список_старых_или_новых_псевдонимов> относится к таким компонентам, как старая или новая строка (OLD / NEW) либо старая или новая таблица (OLD TABLE / NEW TABLE). Ясно, что старые значения не применимы для событий вставки, а новые – для событий удаления.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное их преимущество заключается в том, что стандартные функции сохраняются внутри базы данных и согласованно активизируются при каждом ее обновлении. Это может существенно упростить приложения. Тем не менее следует упомянуть и о присущих триггеру недостатках:
 сложность: при перемещении некоторых функций в базу данных усложняются задачи ее проектирования, реализации и администрирования;
 скрытая функциональность: перенос части функций в базу данных и сохранение их в виде одного или нескольких триггеров иногда приводит к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает его работу, но, к сожалению, может стать причиной незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не в состоянии контролировать все процессы, происходящие в базе данных;
 влияние на производительность: перед выполнением каждой команды по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этой команды.

Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки ее снижение может стать особенно заметным. Очевидно, что при возрастании количества триггеров увеличиваются и накладные расходы, связанные с такими операциями.

Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление "мертвых" блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

Реализация триггеров в среде MS SQL Server

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения триггера:

```
<Определение_триггера>::=
{CREATE | ALTER} TRIGGER имя_триггера
ON {имя_таблицы | имя_просмотра }
[WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
[UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(имя_столбца)
[ {AND | OR} UPDATE(имя_столбца)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_бит_обработки}
бит_маска_изменения)
{оператор_бит_сравнения }бит_маска [...n]}
sql_оператор [...n]
}
}
```

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри триггера к другим базам данных, в том числе и расположенным на удаленном сервере.

Рассмотрим назначение аргументов из команды CREATE | ALTER TRIGGER. Имя триггера должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.

При указании аргумента WITH ENCRYPTION сервер выполняет шифрование кода триггера, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

Типы триггеров

В SQL Server существует два параметра, определяющих поведение триггеров:

 AFTER. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER-триггерами.

 INSTEAD OF. Триггер вызывается вместо выполнения команд. В отличие от AFTER-триггера INSTEAD OF-триггер может быть определен как для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер. Триггеры различают по типу команд, на которые они реагируют.

Существует три типа триггеров:

 INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды INSERT.

 UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE.

 DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.