§ 7. Обзор основных операторов структурированного языка запросов к БД SQL

1. Общие сведения о языках запросов к реляционным БД
2. Структура языка SQL и его типы данных. Понятие представления
3. Основные операторы языка SQL

1. Общие сведения о языках запросов к реляционным БД

Запрос представляет собой специальным образом описанное требование, определяющее состав производимых на БД операций по выборке, удалению или модификации хранимых данных.

Для подготовки запросов чаще всего используются два основных языка описания запросов: **QBE** и **SQL**.

Главное отличие между этими языками заключается в способе формирования запросов: язык **QBE** предполагает *ручное* формирование запроса, в то время как использование языка **SQL** означает *программирование* запроса.

Теоретической основой языка **QBE** является ***реляционное исчисление с переменными-доменами***. Этот язык позволяет формировать сложные запросы к БД путем заполнения предлагаемой СУБД запросной формы. Такой способ формирования запросов обеспечивает высокую наглядность и не требует указания алгоритма выполнения операции. Достаточно описать образец ожидаемого результата (откуда и название языка – запрос по образцу).

Для того чтобы узнать имена доступных отношений БД, в языке **QBE** предусмотрен запрос на выборку имен отношений. Имена атрибутов исходного отношения могут вводиться в шаблон вручную или автоматически. Во втором случае используется запрос на выборку имен атрибутов.

В современных СУБД, например, в **ACCESS** (диалоговые окна) и **VISUAL FOX PRO**, многие действия по подготовке запросов с помощью языка **QBE** выполняются визуально с помощью мыши («протаскиванием» мышью имени атрибута одного отношения к атрибуту другого).

Первый вариант языка **QBE** был предложен в 1975-1977 г. **М. М. Злуффом**.

Краткая история стандартизации языка SQL

Деятельность по стандартизации языка **SQL** началась практически одновременно с появлением его первых коммерческих реализаций. В 1982 г. комитету по **БД** *Американского национального института стандартов* (**ANSI**) было поручено разработать спецификацию стандартного языка реляционных **БД**.

Первый документ из числа проектов стандарта датирован октябрем 1985 г. и является уже не первым проектом стандарта **ANSI**. Стандарт был принят **ANSI** в 1986 г., а в 1987 г. одобрен *Международной организацией по стандартизации* (**ISO**). Этот стандарт языка **SQL** принято называть **SQL/86**.

К 1989 г. стандарт **SQL/86** был несколько расширен, был подготовлен и принят следующий стандарт, получивший название **ANSI/ISO SQL/89**.

Осознавая неполноту стандарта **SQL/89**, на фоне завершения разработки этого стандарта специалисты различных компаний начали работу над стандартом **SQL2**. Эта работа также длилась несколько лет, было выпущено много проектов стандарта, пока, наконец, в марте 1992 г. не был принят окончательный проект стандарта **SQL/92**. Этот стандарт стал существенно полнее стандарта **SQL/89** и охватывал практически все аспекты, необходимые для реализации приложений: *манипулирование схемой БД*, *динамический* **SQL**, *управление транзакциями* и *сессиями* (сессия - это последовательность транзакций, в пределах которой сохраняются временные отношения), и др.

В 1995 г. стандарт был дополнен спецификацией интерфейса уровня вызова (Call-Level Interface – **SQL/CLI**). **SQL/CLI** представляет собой набор спецификаций интерфейсов процедур, вызовы которых позволяют выполнять динамически задаваемые операторы **SQL**. Интерфейсы процедур определены для всех основных языков программирования: C, Ada, Pascal и др. Стандарт **SQL/CLI** послужил основой для создания повсеместно распространенных сегодня интерфейсов **ODBC** (Open Database Connectivity) и **JDBC** (Java Database Connec-tivity). По сути дела **SQL/CLI** представляет собой альтернативу

В **1996** г. к стандарту **SQL/92** был добавлен еще один компонент **SQL/PSM** (Persistent Stored Modules). Основная цель этой спецификации состоит в том, чтобы стандартизировать способы определения и использования *хранимых процедур,* то есть программ, включающих операторы **SQL**, которые сохраняются в **БД**, могут вызываться приложениями и выполняются внутри **СУБД**.

Незадолго до завершения работ по определению стандарта **SQL2** была начата разработка стандарта **SQL3**, которую частично удалось завершить только в 1999 г., и по этой причине стандарт получил название **SQL:1999** (самые первые проекты **SQL3** занимали около 1500 страниц). В 1999 г. были приняты пять первых частей стандарта **SQL:1999**.

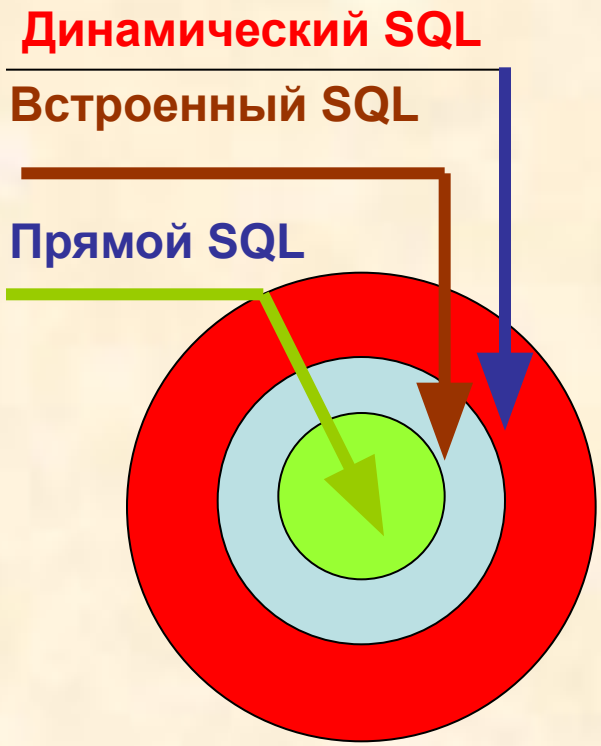
Первая часть стандарта посвящена описанию концептуальной структуры стандарта; *вторая* – образует базис стандарта; *третья* – уточняет по сравнению с **SQL/92** спецификацию **SQL/CLI**; в *четвертой* части специфицируется **SQL/PSM** – синтаксис и семантика языка определения хранимых процедур; в *пятой* части определяются правила связывания **SQL** для стандартных версий языков программирования **FORTRAN**, **PASCAL** и др.

В конце 2003 г. был принят и опубликован новый вариант Международного стандарта **SQL:2003**, который существенно отличается от стандарта **SQL:1999**, и состоит из **9** частей. Особо следует отметить новые части – с номерами **13** и **14** (некоторые части по сравнению со стандартом **SQL:1999** перестали существовать). Часть **13** посвящена использованию подпрограмм и типов **SQL** в языке программирования **Java**. Часть **14** посвящена спецификации языковых средств, позволяющих работать с **XML**-документами в среде **SQL**.

2. Структура языка SQL и его типы данных. Понятие представления

Опишем базовые механизмы языка **SQL**. Язык **SQL**, соответствующий последним стандартам, представляет собой богатый и сложный язык. Поэтому приходится разбивать язык на уровни такие, что каждый уровень языка включает все конструкции, входящие в более низкие уровни. Возможны различные способы разбиения языка на уровни. Один из способов разбиения языка **SQL** на уровни имеет следующий вид.

Разделение языка SQL на уровни



- **Прямой SQL** (*di-rect*) содержит кон-струкции, которые можно использо-вать при прямом взаимодействии ко-нечного пользова-теля с СУБД (напри-мер, в интерактив-ном режиме).

- **Встроенный** (*embedded*) уровень расширяется конструкциями, позволяющими использовать возможности прямого **SQL** в программах, написанных на традиционных языках программирования.
- На уровне **динамического** (*dynamic*) языка **SQL** во встраиваемый **SQL** добавляются конструкции, позволяющие приложениям обращаться к СУБД с конструкциями прямого **SQL**, которые динамически образуются во время выполнения программы.

Типы данных в языке SQL

Данные, хранящиеся в столбцах таблиц **SQL**-ориентированной БД, являются типизированными, то есть представляют собой значения одного из типов данных, предопределенных в языке **SQL** или определяемых пользователями путем применения соответствующих средств языка. Для этого при определении отношения каждому его атрибуту назначается некоторый тип данных (или домен), и в дальнейшем СУБД должна следить за тем, чтобы в каждом столбце каждой строки каждого отношения присутствовали только допустимые значения.

Категории типов данных

- *Точные целочисленные типы* (**INTEGER**, **SMALLINT**).
- *Точные типы, допускающие наличие дробной части* (**NUMERIC** (p, s), где p задает точность значений, а s – число десятичных цифр в дробной части.
- **DECIMAL** (p, s), **DECIMAL** (p), **DECIMAL** .
- ***Литералы типов*** *точных чисел*, представляемые в виде *строк* символов, изображающих десятичные числа со знаком или без знака, допускается внутри числа разделительная точка).

Приближенные числовые типы

- **REAL** – числа с плавающей точкой одинарной точности (точность определяется конкретной реализацией).
- **DOUBLE PRECISION** – числа с плавающей точкой двойной точности.
- **FLOAT(p)** – параметрический тип (точность задается параметром p).
- **Литеральное выражение вида xEy**.

Типы символьных строк

- Тип **CHARACTER** (x) (или **CHAR**) - значениями являются символьные строки (набор символов ASCII).
- Тип **CHARACTER VARYING** (x), где x – количество символов в строке.
- Тип **CHARACTER LARGE OBJECT** – предназначен для определения столбцов, хранящих большие и разные по размеру группы символов.
- ***Литералы символов строк***, заключенные в одинарные или двойные кавычки.

Операции над символьными типами

- Операция ***конкатенации*** (обозначается «||») - соединение строк.
- Функция **выделения подстроки** (обозначается **SUBSTRING**) принимает три аргумента – строку, номер начальной позиции и длину.
- Функция **APPER** возвращает строку, в которой все строчные буквы строки-аргумента заменяются прописными.

- Функция **LOVER** заменяет в заданной строке все прописные буквы строчными.
- Функция определения длины (**CHARA-CTER_LENGTH, OCTET_LENGTH, BIT_-LENGTH**) возвращает длину заданной символьной строки в *символах*, *октетах* или *битах* в виде целого числа.
- Функция определения позиции (**POSITION**) определяет первую позицию в строке S, с которой в нее входит заданная строка S1 (если не входит, возвращается нуль).

Типы даты и времени

- Тип **DATE**. Значения этого типа состоят из компонентов-значений года (4 десятичных цифры), месяца (2 десятичные цифры от 01 до 12) и дня некоторой даты (2 десятичные цифры от 01 до 31).
- **Литералы типа DATE** представляются в виде строки «'yyyy-mm-dd'», например, литерал '1949-04-08' означает дату 8 апреля 1949 г.

- Тип **TIME**. Значения этого параметризованного типа состоят из компонентов-значений часа (2 десятичные цифры от 00 до 23), минуты (2 десятичные цифры от 00 до 59) и секунды (2 десятичные цифры от 00 до 59, но может включать и дополнительные цифры, представляющие доли секунды).
- Существуют и другие типы даты и времени: *типы временной метки, типы времени и временной метки с временной зоной, типы временных интервалов*.

Булевский тип

При определении атрибута булевского типа указывается просто спецификация **BOOLEAN**. Булевский тип состоит из трех значений: ***true***, ***false*** и ***unknown*** (соответствующие литералы обозначаются **TRUE**, **FALSE** и **UNKNOWN**). Поддерживается возможность построения булевских выражений, которые вычисляются в трехзначной логике.

Другие типы данных в SQL

- *Типы коллекций*: типы **массивов ARRAY** (в стандарте SQL:1999) и типы **мультимножеств MULTISET** (стандарт SQL:2003).
- *Анонимные строчные типы* **ROW**.
- *Типы, определяемые пользователем – структурные типы* и *индивидуальные* типы.
- ***Ссылочные типы***.

Замечание 1

В **SQL** для СУБД **FOX PRO** тип данных атрибутов обозначается одной из букв: **C** – *символьный*, **M** – *примечания*, **D** – *дата*, **L** – *логический*, **F**–*действительное число с плавающей точкой*, **N** – *числовой*.

Понятие представления

В результате выборки данных из одного или нескольких отношений может быть получено множество кортежей, называемое представлением.

Представление по существу является таблицей, формируемой в результате выполнения запроса.

Преимущество использования представлений по сравнению запросами к основной таблице состоит в том, что представление будет модифицировано автоматически каждый раз, когда таблица, лежащая в его основе, изменяется. Содержание представления не фиксировано, и обновляется каждый раз, когда на него ссылаются в команде.

Для удобства работы с представлениями в язык **SQL** введено понятие ***курсора***.

Описание и использование курсора

Курсор представляет собой своеобразный *указатель*, используемый для перемещения по наборам кортежей при их обработке.

В *описательной* части программы выполняют связывание переменной типа курсор (**CURSOR**) с оператором языка **SQL** (обычно с оператором **SELECT**).

В выполняемой части программы производится *открытие* курсора (**OPEN**<имя курсора>), *перемещение* курсора по кортежам (**FETCH** <имя курсора>), сопровождаемое соответствующей обработкой и, наконец, *закрытие* курсора (**CLOSE** <имя курсора>).

3. Основные операторы языка SQL

Опишем *минимальное подмножество* языка **SQL**, опираясь на его реализацию в *стандартном интерфейсе* **ODBC** (совместимость открытых БД) фирмы **MICROSOFT** (стандарт **ISO SQL/92** и **ANSI SQL/92**). Операторы языка **SQL** можно условно разделить на два подъязыка: *язык определения данных* (**DDL**) и *язык манипулирования данными* (**DML**).

Операторы подъязыка DDL

НАЗВАНИЕ	НАЗНАЧЕНИЕ
CREATE TABLE	Создание таблицы
DROP TABLE	Удаление таблицы
ALTER TABLE	Изменение структуры таблицы
CREATE INDEX	Создание индекса
DROP INDEX	Удаление индекса
CREATE VIEW	Создание представления
DROP VIEW	Удаление представления
GRANT	Назначение привилегий
REVOKE	Удаление привилегий

Формат операторов подъязыка DDL

1. CREATE TABLE <имя_таблицы>
(< имя_столбца > < тип данных >
[**NOT NULL**] [, < имя_столбца >
< тип данных > [**NOT NULL**]]…)

В СУБД **FOX PRO** вместо **CREATE TABLE** пишут **CREATE DBF** .

Конструкция **NOT NULL** служит дополнительным правилом контроля вводимых значений и для столбца означает, что в нем должно быть определено *не пустое* значение. В некоторых СУБД в качестве **NOT NULL** указывается размер поля в символах.

Пример 1

```
CREATE TABLE
  STUDENTS
(SNUM INTEGER,
SFAM  CHAR (20),
SIMA CHAR (10),
SOTCH CHAR (15),
STIP DECIMAL);
```

Создано отношение с именем «Студенты» и атрибутами соответственно «Номер студенческого билета», «Фамилия», «Имя», «Отчество» студента и «Размер получаемой стипендии».

Замечания 2 и 3

- Порядок расположения атрибутов в отношении определяется тем, в какой последовательности они указаны в команде создания отношения.
- Кроме создания отношения в **SQL/92** аналогичной командой **CREATE SCHEMA** можно создавать схемы с необязательным указанием автора, а также можно создать домен – **CREATE DOMAIN.**

2. Оператор удаления отношения

Оператор удаления отношения имеет формат

DROP TABLE <имя отношения>

Перед удалением отношения необходимо убедиться в том, что оно не ***ссылается*** на другое отношение и ***не используется*** в каком-либо представлении. Кроме того, чтобы удалить отношение, пользователь должен быть его ***собственником***. Перед удалением отношение должно быть ***очищено***.

3. Формат оператора изменения структуры отношения

ALTER TABLE <имя отношения> ({**ADD**, **MODIFY**, **DROP**} <имя_атрибута> [<тип данных>] [**NOT NULL**] [, ({**ADD**, **MODIFY**, **DROP**} <имя_атрибута>[<тип данных>] [**NOT NULL**]…]);

ADD, **MODIFY**, **DROP** – соответственно добавление, изменение и удаление одного или нескольких атрибутов отношения.

Пример 2

Добавим к отношению STUDENTS два атрибута для хранения информации о курсе и специальности студента:

```
ALTER TABLE STUDENTS
(ADD COURS INTEGER,
SPEC CHAR (10));
```

4. Формат оператора создания индекса

CREATE [UNIQUE] INDEX <имя_индекса> **ON** <имя_отношения> (<имя_атрибута> [**ASC**|**DESC**] [, имя_атрибута> [**ASC**|**DESC**]…]);

Индексы создаются для ускорения выполнения запросных и поисковых операций с отношением. Опция **UNIQUE** (уникальность значений во всех указанных атрибутах) является *необязательной.*

ASC|**DESC** – сортировка значений.

Пример 3

Очевидно, что в отношении STUDENTS одним из наиболее употребляемых может стать индекс по атрибуту, содержащему фамилию студента. Тогда команда для создания такого индекса будет следующей:

CREATE INDEX SFAMIND ON STUDENTS (SFAM);

5. Формат оператора удаления индекса

DROP INDEX <имя_индекса>

Этот оператор позволяет удалять созданный ранее индекс с соответствующим именем. При этом удаление индекса не воздействует на данные, которые содержатся в индексированных атрибутах.

6. Формат оператора создания представления

CREATE VIEW <имя_представления>
[(<имя_атрибута> [,<имя_атрибута>]…)]
AS <оператор **SELECT**>;

Представление формируется в результате выполнения запроса. Если имена атрибутов в представлении не указываются, то будут использоваться имена атрибутов из запроса, описываемого соответствующим оператором **SELECT**.

Пример 4

Создадим представление с именем OTLSTUD, которое содержит информацию о студентах, получающих стипендию в размере 1200 руб. (его можно использовать в командах наравне с другими таблицами):

```
CREATE VIEW OTLSTUD AS SELECT *
FROM STUDENTS WHERE STIP= 1200;
```

7. Формат оператора удаления представления

DROP VIEW <имя_представления>

Следует различать *модифицируемые* представления и представления, предназначенные только для *чтения*. Модифицируемые представления в **SQL** должны удовлетворять некоторым критериям, на которых мы не останавливаемся и рекомендуем изучить их самостоятельно.

8. Операторы назначения и удаления привилегий

В языке **SQL** одним из принципов защиты данных служит система привилегий, то есть прав пользователя на проведение тех или иных действий над определенными объектами БД. Создает пользователей и назначает им привилегии **администратор** БД. Существует несколько типов привилегий, соответствующих типам операций над БД.

Каждый пользователь БД в среде **SQL** имеет специальное идентификационное имя (**ID**) доступа. Команда, посланная в БД, ассоциируется с определенным пользователем, то есть со специальным идентификатором доступа. В системах с большим количеством пользователей существует специальная процедура входа в систему, которую пользователь должен обязательно выполнить для получения доступа.

Каждый пользователь имеет набор привилегий. Эти привилегии могут изменяться со временем: новые добавляться командой **GRANT**, старые удаляться командой **REVOKE**. Различные форматы команды **GRANT** позволяют сразу передавать несколько привилегий одному пользователю (списки привилегий разделяются запятыми), нескольким пользователям, для отдельных пользователей можно вводить ограничения и т.д.

Пример 5

Пусть пользователь **Гамидов** владеет таблицей студентов **STUDENTS** и желает разрешить пользователю **Куриловой** выполнить запрос к этой таблице. Тогда **Гамидов** должен в этом случае выполнить команду

GRANT SELECT ON STUDENTS TO Курилова

Теперь **Курилова** может выполнять запросы к таблице **STUDENTS** без других привилегий, то есть она может только выбирать значения, но не может выполнить любое другое действие, которое бы воздействовало на значения в таблице, включая использование ее в качестве родительской таблицы внешнего ключа.

Формат операторов подъязыка DML

НАЗВАНИЕ ОПЕРАТОРА	НАЗНАЧЕНИЕ ОПЕРАТОРА
SELECT	Выборка кортежей
UPDATE	Изменение кортежей
INSERT	Вставка новых кортежей
DELETE	Удаление кортежей

1. Формат оператора выборки кортежей

```
SELECT[ALL|DISTINCT]
<список данных>
FROM <список отношений>
[WHERE <условие выборки>]
[GROUP BY <имя_атрибута>
[, <имя_атрибута>]...]
[HAVING <условие поиска>]
[ORDER BY <спецификация>
[, <спецификация> ]...];
```

Оператор **SELECT** позволяет производить выборку и вычисления над данными из одного или нескольких отношений. Результатом выполнения этого оператора является ответная таблица, которая может иметь (**ALL**), или не иметь (**DISTINCT**) повторяющиеся кортежи. По умолчанию в ответную таблицу включаются все кортежи, в том числе и повторяющиеся.

В отборе участвуют кортежи одной или нескольких таблиц, перечисленных в списке таблиц после слова **FROM**.

При использовании в списке данных нескольких таблиц для указания принадлежности атрибута некоторой таблице применяют конструкцию вида

<Имя_отношения> . <Имя_атрибута> .

Список данных может содержать и выражения над атрибутами, в том числе, знаки арифметических операций, константы и круглые скобки.

Операнд **WHERE** задает условия, которым должны удовлетворять кортежи в результирующей таблице. Выражение <условие выборки> является логическим. Его элементами могут быть имена атрибутов, операции сравнения, логические связки ***and***, ***or***, ***not***, круглые скобки, а также специальные функции **LIKE**, **NULL**, **IN** и др.

Операнд **GROUP BY** позволяет выделять в результирующем множестве кортежей ***группы***. Группой являются кортежи с совпадающими значениями в столбцах, перечисленных за ключевыми словами **GROUP BY**. Выделение групп требуется для использования в логических выражениях операндов **WHERE** и **HAVING**, где **HAVING<условия поиска>** является специальной формой операнда **WHERE**, и указывает дополнительный критерий отбора данных в каждую группу.

В логических и арифметических выражениях можно использовать следующие функции как групповые:

AVG, MAX, MIN, SUM, COUNT.

Наконец, опция **ORDER BY** задает порядок сортировки результирующего множества кортежей. Она аналогична конструкции оператора **CREATE INDEX**.

Каждая <спецификация> представляет собой пару

<имя_атрибута> [**ASC**|**DESC**].

2. Формат оператора изменения кортежей

```
UPDATE
   <имя_отношения>
SET<имя_атрибута> =
   = {<выражение>,
   NULL}
[, SET
   <имя_атрибута>=
   = {<выражение>,
   NULL}…]
[ WHERE <условие>];
```

Выполнение оператора **UPDATE** состоит в изменении значений в определенных операндом **SET** атрибутах для тех кортежей, которые удовлетворяют условию, заданному операндом **WHERE**.

3. Форматы оператора вставки новых кортежей

а) **INSERT INTO** <имя_отноше-ния> [(<список атрибутов>)]
VALUES (<список значений>);

б) **INSERT INTO** <имя_отноше-ния> [(<список атрибутов>)]
<предложение **SELECT**>;

4. Формат оператора удаления кортежей и замечание

DELETE FROM
<имя_отношения>
[**WHERE** <условие>];

Замечание. *Язык SQL обладает реляционной полнотой.*