6 ЯЗЫК ЗАПРОСОВ SQL

Прежде всего, язык SQL сочетает средства DDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД – именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые *представления* БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или их части другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

6.1 Типы данных

В языке SQL поддерживаются следующие типы данных: CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION. Эти типы данных классифицируются на типы строк символов, точных чисел и приблизительных чисел.

К первому классу относится CHARACTER. Спецификатор типа имеет вид CHARACTER (lenght), где lenght задает длину строк данного типа. Заметим, что в SQL/89 нет типа строк переменного размера, хотя во многих реализациях они допускаются. Литеральные строки символов изображаются в виде 'последовательность символов' (например, 'example').

Представителями второго класса типов являются NUMERIC, DECIMAL (или DEC), INTEGER (или INT) и SMALLINT. Спецификатор типа NUMERIC

имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представляемые с точностью precision и масштабом scale. Здесь и далее, если опущен масштаб, то он полагается равным 0, а если опущена точность, то ее значение по умолчанию определяется в реализации.

Спецификатор типа DECIMAL (или DEC) имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представленные с масштабом scale и точностью, равной или большей значения precision.

INTEGER специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью. SMALLINT специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью, не большей, чем точность чисел типа INTEGER.

Заметим еще, что в большинстве реализаций SQL поддерживаются некоторые дополнительные типы данных, например, DATE, TIME, INTERVAL, MONEY. Некоторые из этих типов специфицированы в стандарте SQL, но в текущих реализациях синтаксические и семантические свойства таких типов могут различаться.

6.2 Язык определения данных DDL

В соответствии с правилами SQL каждая таблица БД имеет *простое* и *квалифицированное* имена. В качестве квалификатора имени выступает «идентификатор полномочий» таблицы, который обычно в реализациях совпадает с именем пользователя, и квалифицированное имя таблицы имеет вид:

```
<идентификатор полномочий>.<простое имя>
```

Подход к определению схемы в SQL состоит в том, что все таблицы с одним идентификатором полномочий создаются (определяются) путем выполнения одного оператора определения схемы.

6.2.1 Оператор определения схемы

В операторе определения схемы содержится идентификатор полномочий и список элементов схемы, каждый из которых может быть определением таблицы, определением представления (view) или определением привилегий. Каждое из этих определений представляется отдельным оператором SQL, но все они должны быть встроены в оператор определения схемы.

Для этих операторов мы приведем синтаксис, поскольку это позволит более четко описать их особенности.

6.2.2 Определение таблицы

Оператор определения таблицы имеет следующий синтаксис:

```
<table definition> ::=
   CREATE TABLE  <table name>   (<table element>
   [{,<table element>}...])
<table element> ::=
  <column definition>
| <table constraint definition>
```

Кроме имени таблицы, в операторе специфицируется список элементов таблицы, каждый из которых служит либо для определения столбца, либо для определения ограничения целостности определяемой таблицы. Требуется наличие хотя бы одного определения столбца. Оператор CREATE TABLE определяет так называемую базовую таблицу, т.е. реальное хранилище данных.

Для определения столбцов таблицы и ограничений целостности используются специальные операторы, которые должны быть вложены в оператор определения таблицы.

6.2.3 Определение столбца

Оператор определения столбца описывается следующими синтаксическими правилами:

```
<column definition> ::=
    <column name> <data type>
    [<default clause>] [<column constraint>...]
<default clause> ::=
    DEFAULT { <literal> | USER | NULL }
<column constraint> ::=
    NOT NULL [<unique specification>]
  | <references specification>
  | CHECK (<search condition>)
```

Кроме обязательной части, в которой определяется имя столбца и его тип данных, определение столбца может содержать два необязательных раздела: раздел значения столбца по умолчанию и раздел ограничений целостности столбца.

В разделе значения по умолчанию указывается значение, которое должно быть помещено с строку, заносимую в данную таблицу, если значение данного столбца явно не указано. Значение по умолчанию может быть указано в виде литеральной константы с типом, соответствующим типу столбца; путем задания ключевого слова USER, которому при выполнении оператора занесения строки соответствует символьная строка, содержащая имя текущего пользователя (в этом случае столбец должен иметь тип символьных строк); или путем задания ключевого слова NULL, означающего, что значением по умолчанию является неопределенное значение. Если значение столбца по умолчанию не специфицировано и в разделе ограничений целостности

столбца указано NOT NULL, то попытка занести в таблицу строку с неспецифицированным значением данного столбца приведет к ошибке.

Указание в разделе ограничений целостности NOT NULL приводит к неявному порождению проверочного ограничения целостности для всей таблицы «CHECK (C IS NOT NULL)» (где C – имя данного столбца). Если ограничение NOT NULL не указано и раздел умолчаний отсутствует, то неявно порождается раздел умолчаний DEFAULT NULL. Если указана спецификация уникальности, то порождается соответствующая спецификация уникальности для таблицы.

Если в разделе ограничений целостности указано ограничение по ссылкам данного столбца (<reference specification>), то порождается соответствующее определение ограничения по ссылкам для таблицы:

```
FOREIGN KEY(C) <reference specification>.
```

Наконец, если указано проверочное ограничение столбца, то условие поиска этого ограничения должно ссылаться только на данный столбец, и неявно порождается соответствующее проверочное ограничение для всей таблицы.

6.3 Язык запросов DQL

Ниже приведены синтаксические правила формирования запросов в стандарте SQL.

```
<cursor specification> ::=
   <query expression> [<order by clause>]
<query expression> ::=
   <query term>
|  <query expression> UNION [ALL] <query term>
<query term> ::=
   <query specification>
|  (<query expression>)
<query specification> ::=
   (SELECT [ALL | DISTINCT] <select list> <table
expression>)
<select statement> ::=
   SELECT [ALL | DISTINCT] <select list>
   INTO <select target list> <table expression>
<subquery> ::=
   (SELECT [ALL | DISTINCT] <result specification>
   <table expression>
<table expression> ::=
   <from clause>
  [<where clause>]
  [<group by clause>]
  [<having clause>]
```

Язык допускает три типа синтаксических конструкций, начинающихся с ключевого слова SELECT: *спецификация курсора* (cursor specification), *оператор выборки* (select statement) и *подзапрос* (subquery). Основой для них является синтаксическая конструкция *табличное выражение* (table expression). Семантика табличного выражения состоит в том, что на основе последовательного применения разделов from, where, group by и having из заданных в разделе from таблиц строится некоторая новая результирующая таблица, порядок следования строк которой не определен и среди строк которой могут находиться дубликаты (т.е. в общем случае таблица-результат табличного выражения не является множеством). На самом деле именно структура табличного выражения наибольшим образом характеризует структуру запросов языка SQL.

6.3.1 Спецификация курсора

Наиболее общей является конструкция спецификация курсора. *Курсор* – это понятие языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД. К табличным выражениям, участвующим в спецификации курсора, не предъявляются какие-либо ограничения. Как видно из сводки синтаксических правил, при определении спецификации курсора используются три дополнительных конструкции: спецификация запроса, выражение запросов и раздел ORDER BY.

В *спецификации запроса* задается список выборки. В общем случае это список арифметических выражений над значениями столбцов результата табличного выражения и констант, однако чаще всего это просто список столбцов. В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк, но другое число столбцов, содержащих результаты вычисления соответствующих арифметических выражений из списка выборки. В типичной ситуации – это результат выполнения операции проекции результата табличного выражения на список столбцов. Кроме того, в спецификации запроса могут содержаться ключевые слова ALL или DISTINCT. При наличии ключевого слова DISTINCT из таблицы, полученной применением списка выборки к результату табличного выражения, удаляются строки-дубликаты; при указании ALL (или просто при отсутствии DISTINCT) удаление строк-дубликатов не производится.

Выражение запросов – это выражение, строящееся по указанным синтаксическим правилам на основе спецификаций запросов. Единственной операцией, которую разрешается использовать в выражениях запросов, является операция UNION (объединение таблиц) с возможной разновидностью UNION ALL. К таблицам-операндам выражения запросов предъявляется требование, что все они должны содержать одно и то же число столбцов и соответствующие столбцы всех операндов должны быть одного и

того же типа. Выражение запросов вычисляется слева направо с учетом скобок. При выполнении операции UNION производится обычное теоретико-множественное объединение операндов, т.е. из результирующей таблицы удаляются дубликаты. При выполнении операции UNION ALL образуется результирующая таблица, в которой могут содержаться строки-дубликаты.

Наконец, раздел ORDER BY позволяет установить желаемый порядок просмотра результата выражения запросов. Синтаксис ORDER BY следующий:

```
<order by clause> ::=
   ORDER BY <sort specification>
  [{,<sort specification>}...]
<sort specification> ::=
   {<unsigned integer> | <column specification>}
  [ASC | DESC]
```

Как видно из этих синтаксических правил, фактически задается список столбцов результата выражения запросов, и для каждого столбца указывается порядок просмотра строк результата в зависимости от значений этого столбца (ASC – по возрастанию (умолчание), DESC – по убыванию). Столбцы можно задавать их именами в том и только в том случае, когда (1) выражение запросов не содержит операций UNION или UNION ALL и (2) в списке выборки спецификации запроса этому столбцу соответствует арифметическое выражение, состоящее только из имени столбца. Во всех остальных случаях в разделе ORDER BY должен указываться порядковый номер столбца в таблице-результате выражения запросов.

6.3.2 Подзапрос

Подзапрос – это запрос, который может входить в предикат условия выборки оператора SQL. Поскольку подзапрос всегда вложен в некоторый другой оператор SQL, то в качестве констант в арифметическом выражении выборки и логических выражениях разделов WHERE и HAVING можно использовать значения столбцов текущих строк таблиц, участвующих в (под)запросах более внешнего уровня.

6.4 Табличное выражение

Стандарт SQL рекомендует рассматривать вычисление табличного выражения как последовательное применение разделов FROM, WHERE, GROUP BY и HAVING к таблицам, заданным в списке FROM.

6.4.1 Раздел FROM

Раздел FROM имеет следующий синтаксис:

```
<from clause> ::=
   FROM <table reference>
 ({,<table reference>}...]
<table reference> ::=
   <table name> [<correlation name>]
```

Результатом выполнения раздела FROM является расширенное декартово произведение таблиц, заданных списком таблиц раздела FROM.

Как видно из синтаксиса, рядом с именем таблицы можно указывать еще одно имя «correlation name». Фактически это некоторый синоним (alias) имени таблицы, который можно использовать в других разделах табличного выражения для ссылки на строки именно этого вхождения таблицы.

Если табличное выражение содержит только раздел FROM (это единственный обязательный раздел табличного выражения), то результат табличного выражения совпадает с результатом раздела FROM.

Примеры.

Декартово произведение отношений R1 и R2:

```
SELECT * FROM R1, R2
```

Проекция декартова произведения R1⊕R2 на столбцы A и B:

```
SELECT R1.A, R2.B FROM R1, R2
```

6.4.2 Предложение WHERE

Если в табличном выражении присутствует предложение WHERE, то оно вычисляется следующим за FROM. Синтаксис предложения WHERE следующий:

```
<where clause> ::= WHERE <search condition>
<search condition> ::=
   <boolean term>
 ( <search condition> OR <boolean term>
<Boolean term> ::=
   <boolean factor>
 ( <boolean term> AND <boolean factor>
<boolean factor> ::= [NOT] <boolean primary>
<boolean primary> ::= <predicate> | (<search
condition>)
```

Вычисление предложения WHERE производится по следующим правилам: Пусть R – результат вычисления раздела FROM. Тогда условие (search condition) применяется ко всем строкам R, и результатом предложения WHERE является таблица, состоящая из тех строк R, для которого результатом

вычисления условия является true. Если условие выборки включает подзапросы, то каждый подзапрос вычисляется для каждого кортежа таблицы R.

Заметим, что поскольку SQL допускает наличие в базе данных неопределенных значений, то вычисление условия поиска производится не в булевой, а в трехзначной логике со значениями true, false и null (неизвестно). Для любого предиката известно, в каких ситуациях он может порождать значение unknown. Булевские операции AND, OR и NOT работают в трехзначной логике следующим образом:

```
true AND null = null
false AND null = false
null AND null = null
true OR null = true
false OR null = null
null OR null = null
NOT null = null
```

Среди предикатов условия поиска в соответствии с SQL могут находиться следующие предикаты: предикат сравнения, предикат between, предикат in, предикат like, предикат null и предикаты exists, any, all. Сразу заметим, что во всех реализациях SQL на эффективность выполнения запроса существенно влияет наличие в условии поиска простых предикатов сравнения (предикатов, задающих сравнение столбца таблицы с константой). Наличие таких предикатов позволяет СУБД использовать индексы при выполнении запроса, т.е. избегать полного просмотра таблицы.

Синтаксис *предиката сравнения* определяется следующими правилами:

```
<comparison predicate> ::=
   <value expression> <comp op>
  {<value expression> | <subquery>}
<comp op> ::=
   = | <> | < | > | <= | >=
```

Арифметические выражения левой и правой частей предиката сравнения строятся по общим правилам построения арифметических выражений и могут включать в общем случае имена столбцов таблиц из раздела FROM и константы. Типы данных арифметических выражений должны быть сравнимыми (например, если тип столбца a таблицы A является типом символьных строк, то предикат «a = 5» недопустим).

Если правый операнд операции сравнения задается подзапросом, то дополнительным ограничением является то, что мощность результата подзапроса должна быть не более единицы. Если хотя бы один из операндов операции сравнения имеет неопределенное значение, или если правый

операнд является подзапросом с пустым результатом, то значение предиката сравнения равно null.

Заметим, что значение арифметического выражения не определено, если в его вычислении участвует хотя бы одно неопределенное значение. Еще одно важное замечание: в контексте GROUP BY, DISTINCT и ORDER BY неопределенное значение выступает как специальный вид определенного значения, т.е. возможно, например, образование группы строк, значение указанного столбца которых является неопределенным.

Предикат between имеет следующий синтаксис:

```
<between predicate> ::=
   <value expression>
   [NOT] BETWEEN <value expression> AND <value
expression>
```

Результат «x BETWEEN y AND z» тот же самый, что результат «x >= y AND x <= z». Результат «x NOT BETWEEN y AND z» тот же самый, что результат «NOT (x BETWEEN y AND z)».

Предикат in определяется следующими синтаксическими правилами:

```
<in predicate> ::=
   <value expression> [NOT] IN
  {<subquery> | (<in value list>)}
<in value list> ::=
   <value specification>
 {,<value specification>}...
```

Типы левого операнда и значений из списка правого операнда (напомним, что результирующая таблица подзапроса должна содержать ровно один столбец) должны быть сравнимыми.

Значение предиката равно true в том и только в том случае, когда значение левого операнда совпадает хотя бы с одним значением списка правого операнда. Если список правого операнда пуст (так может быть, если правый операнд задается подзапросом) или значение «подразумеваемого» предиката сравнения x = y (где x – значение арифметического выражения левого операнда) равно false для каждого элемента y списка правого операнда, то значение предиката in равно false. В противном случае значение предиката in равно null. По определению значение предиката «x NOT IN S» равно значению предиката «NOT (x IN S)».

Предикат like имеет следующий синтаксис:

```
<like predicate> ::=
   <column specification> [NOT] LIKE <pattern>
   [ESCAPE <escape character>]
```

```
<pattern> ::= <value specification>
   <escape character> ::= <value specification>
```

Типы данных столбца левого операнда и образца должны быть типами символьных строк. В разделе ESCAPE должен специфицироваться одиночный символ.

Значение предиката равно true, если pattern является подстрокой заданного столбца. При этом если раздел ESCAPE отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона: символ подчеркивания («_») обозначает любой одиночный символ; символ процента («%») обозначает последовательность произвольных символов произвольной длины (может быть, нулевой).

Если же раздел ESCAPE присутствует и специфицирует некоторый одиночный символ x, то пары символов «x_» и «x%» представляют одиночные символы «_» и «%» соответственно.

Значение предиката like есть unknown, если значение столбца, либо шаблона не определено.

Значение предиката «x NOT LIKE y ESCAPE z» совпадает со значением «NOT x LIKE y ESCAPE z».

Предикат null описывается синтаксическим правилом:

```
<null predicate> ::=
   <column specification> IS [NOT] NULL
```

Этот предикат всегда принимает значения true или false. При этом значение «x IS NULL» равно true тогда и только тогда, когда значение x не определено. Значение предиката «x NOT IS NULL» равно значению «NOT x IS NULL».

Предикат exists имеет следующий синтаксис:

```
<exists predicate> ::=
   EXISTS <subquery>
```

Значением этого предиката всегда является true или false, и это значение равно true тогда и только тогда, когда результат вычисления подзапроса не пуст.

Предикат any имеет следующий синтаксис:

```
<any predicate> ::=
   <value expression> <comp op> ANY <subquery>
```

Значением этого предиката всегда является true или false, и это значение равно true тогда и только тогда, когда предикат сравнения справедлив хотя бы для одного кортежа результата подзапроса.

Предикат all имеет следующий синтаксис:

```
<all predicate> ::=
   <value expression> <comp op> ALL <subquery>
```

Значением этого предиката всегда является true или false, и это значение равно true тогда и только тогда, когда предикат сравнения справедлив для всех кортежей результата подзапроса.

Примеры. Пусть база данных, предназначенная для учета результатов сессии, состоит из следующих отношений:

- Сводная ведомость – `Ведом(ФИО, Дисц, Оценка);`
- Список студенческих групп – `Группы(ФИО, Группа);`
- Список дисциплин – `Дисц(Группа, Дисц).`

1) Выдать список дисциплин, по которым проводится контроль знаний в текущую сессию:

```
SELECT DISTINCT Дисц FROM Дисц
```

Префикс (имя отношения) можно опустить, если это не приводит к неоднозначной трактовке списка выборки.

2) Выдать список студентов, сдавших физику на «отлично»:

```
SELECT ФИО FROM Ведом
      WHERE Дисц = “Физика” AND Оценка = 5
```

3) Выдать список студентов, получивших несколько неудовлетворительных оценок на экзаменах:

```
SELECT DISTINCT ФИО FROM Ведом A, Ведом B
      WHERE A.ФИО = B.ФИО AND A.Дисц <> B.Дисц
            AND A.Оценка = 2 AND B.Оценка = 2
```

В данном примере запрос обрабатывает два экземпляра отношения Ведом, для которых введены псевдонимы A и B.

4) Выдать список студентов, не явившихся на экзамены, и дисциплины:

```
SELECT ФИО, Дисц FROM Ведом WHERE Оценка IS NULL
```

6.4.3 Предложение GROUP BY

Если в табличном выражении присутствует предложение GROUP BY, то оно выполняется следом за WHERE. Синтаксис раздела GROUP BY следующий:

```
<group by clause> ::=
```

```
GROUP BY <column specification>
[{,<column specification>}...]
```

Если обозначить через R таблицу, являющуюся результатом предыдущего раздела (FROM или WHERE), то результатом предложения GROUP BY является разбиение R на множество групп строк, состоящего из минимального числа групп, таких, что для каждого столбца из списка столбцов раздела GROUP BY во всех строках каждой группы, включающей более одной строки, значения этого столбца равны.

6.4.4 Предложение HAVING

Наконец, последним при вычислении табличного выражения используется предложение HAVING (если оно присутствует). Синтаксис этого предложения следующий:

```
<having clause> ::=
   HAVING <search condition>
```

Предложение HAVING может осмысленно появиться в табличном выражении только в том случае, когда в нем присутствует предложение GROUP BY. Условие этого раздела задается для групп строк сгруппированной таблицы. Формально предложение HAVING может присутствовать и в табличном выражении, не содержащем GROUP BY. В этом случае полагается, что результат вычисления предыдущих разделов представляет собой сгруппированную таблицу, состоящую из одной группы без выделенных столбцов группирования.

Условие отбора групп предложения HAVING строится по тем же синтаксическим правилам, что и условие отбора кортежей предложения WHERE, и может включать те же самые предикаты. Однако имеются специальные синтаксические ограничения по части использования в условии поиска спецификаций столбцов таблиц из раздела FROM данного табличного выражения. Эти ограничения следуют из того, что условие раздела HAVING задает условие на целую группу, а не на индивидуальные кортежи.

Поэтому в арифметических выражениях предикатов, входящих в условие выборки предложения HAVING, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе GROUP BY. Остальные столбцы можно специфицировать только внутри спецификаций агрегатных функций COUNT, SUM, AVR, MIN и MAX, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки предложения HAVING: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

Результатом выполнения предложения HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть true. В частности, если предложение HAVING присутствует в табличном выражении, не содержащем GROUP BY, то результатом его выполнения будет либо пустая таблица, либо результат выполнения предыдущих разделов табличного выражения, рассматриваемый как одна группа без столбцов группирования.

6.5 Агрегатные функции и результаты запросов

Агрегатные функции (в стандарте SQL они называются функциями над множествами) определяются в SQL следующими синтаксическими правилами:

```
<set function specification> ::=
   COUNT(*) | <distinct set function>
            | <all set function>
<distinct set function> ::=
   { AVG | MAX | MIN | SUM | COUNT }
    (DISTNICT <column specification>)
<all set function> ::=
   { AVR | MAX | MIN | SUM } ([ALL] <value expression>)
```

Как видно из этих правил, в стандарте SQL определены пять агрегатных функций:

- COUNT – число строк или значений;
- MAX – максимальное значение;
- MIN – минимальное значение;
- SUM – суммарное значение;
- AVR – среднее значение.

6.5.1 Семантика агрегатных функций

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(*), фактический (т.е. требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций – null.

Пусть T обозначает тип значений из этого списка. Тогда результат вычисления функции COUNT – точное число с масштабом и точностью, определяемыми в реализации. Тип результата значений функций MAX и MIN

совпадает с T. При вычислении функций SUM и AVR тип T не должен быть типом символьных строк, а тип результата функции – это тип точных чисел с определяемыми в реализации масштабом и точностью, если T – тип точных чисел, и тип приблизительных чисел с определяемой в реализации точностью, если T – тип приблизительных чисел.

Вычисление функции COUNT(*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением null во всех строках.

Если агрегатная функция специфицирована с ключевым словом DISTINCT, то список значений строится из значений указанного столбца. (Подчеркнем, что в этом случае не допускается вычисление арифметических выражений!) Далее из этого списка удаляются неопределенные значения, и в нем устраняются значения-дубликаты. Затем вычисляется указанная функция.

Если агрегатная функция специфицирована без ключевого слова DISTINCT (или с ключевым словом ALL), то список значений формируется из значений арифметического выражения, вычисляемого для каждой строки заданного множества. Далее из списка удаляются неопределенные значения, и производится вычисление агрегатной функции. Обратите внимание, что в этом случае не допускается применение функции COUNT!

6.5.2 Результаты запросов

Агрегатные функции можно разумно использовать в спецификации курсора, операторе выборки и подзапросе после ключевого слова SELECT (будем называть в этом подразделе все такие конструкции списком выборки), и в условии выборки раздела HAVING. Стандарт допускает более экзотические использования агрегатных функций в подзапросах (агрегатная функция на группе кортежей внешнего запроса), но на практике они встречаются очень редко.

Рассмотрим различные случаи применения агрегатных функций в списке выборки в зависимости от вида табличного выражения.

Если результат табличного выражения R не является сгруппированной таблицей, то появление хотя бы одной агрегатной функции от множества строк R в списке выборки приводит к тому, что R неявно рассматривается как сгруппированная таблица, состоящая из одной (или нуля) групп с отсутствующими столбцами группирования. Поэтому в этом случае в списке выборки не допускается прямое использование спецификаций строк R: все они должны находиться внутри спецификаций агрегатных функций. Результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к R.

Аналогично обстоит дело в том случае, когда R представляет собой сгруппированную таблицу, но табличное выражение не содержит раздела GROUP BY (и, следовательно, содержит раздел HAVING). Если в предыдущем случае было два варианта формирования списка выборки: только с прямым указанием столбцов R или только с указанием их внутри спецификаций агрегатных функций, то в данном случае возможен только

второй вариант. Результат табличного выражения явно объявлен сгруппированной таблицей, состоящей из одной группы, и результат запроса можно формировать только путем применения агрегатных функций к этой группе строк. Опять результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к R.

Наконец, рассмотрим случай, когда R представляет собой «настоящую» сгруппированную таблицу, т.е. табличное выражение содержит раздел GROUP BY и, следовательно, определен, по крайней мере, один столбец группирования. В этом случае правила формирования списка выборки полностью соответствуют правилам формирования условия выборки раздела HAVING: допускает прямое использование спецификации столбцов группирования, а спецификации остальных столбцов R могут появляться только внутри спецификаций агрегатных функций. Результатом запроса является таблица, число строк в которой равно числу групп в R, и каждая строка формируется на основе значений столбцов группирования и агрегатных функций для данной группы.

Примеры.

1) По каждой из дисциплин выдать количество различных оценок в сводной ведомости:

```
SELECT Дисц, COUNT(DISTINCT Оценка) FROM Ведом
      WHERE Оценка IS NOT NULL
      GROUP BY Дисц
```

2) Для каждой из групп и дисциплин выдать количество студентов, успешно сдавших экзамены, и средний балл:

```
SELECT Группы.Группа, Ведом.Дисц, COUNT(*), AVR(Оценка)
      FROM Группы, Ведом
      WHERE Ведом.ФИО = Группы.ФИО
            AND Ведом.Оценка IS NOT NULL
            AND Ведом.Оценка > 2
      GROUP BY Ведом.Дисц, Группы.Группа
```

3) Выдать список групп, в которых по одной и той же дисциплине получено более одного «неуда»:

```
SELECT Группы.Группа, FROM Ведом, Группы
      WHERE Ведом.ФИО = Группы.ФИО AND Ведом.Оценка = 2
      GROUP BY Ведом.Дисц, Группы.Группа
      HAVING COUNT(*) > 1
```

В этом примере агрегатная функция используется в условии отбора групп.

4) Выдать список студентов, успешно сдавших все экзамены:

```
SELECT ФИО FROM Ведом
      WHERE Оценка IS NOT NULL AND Оценка > 2
      GROUP BY ФИО
      HAVING COUNT(*) = (SELECT COUNT(*)FROM Группы, Дисц
            WHERE Группы.Группа = Дисц.Группа
                  AND Ведом.ФИО = Группы.ФИО)
```

Вложенный запрос возвращает количество дисциплин, по которым должен сдать экзамен текущий студент из внешнего запроса. COUNT во внешнем запросе подсчитывает количество успешно сданных этим студентом экзаменов.

Рассмотрим базу данных для учета поставок комплектующих на предприятие. Она состоит из двух отношений:
Ассортимент комплектующих – `Ассорт(Ном_Дет,Наименование);`
Список поставок – `Поставки(Поставщик, Ном_Дет, Дата).`

5) Выдать список поставщиков, которые поставляют весь ассортимент комплектующих:

```
SELECT DISTINCT Поставщик FROM Поставки sp
      WHERE NOT EXIST (SELECT Ном_Дет FROM Ассорт
            WHERE NOT EXIST (select * FROM Поставки sp1
                  WHERE sp1.Поставщик = sp.Поставщик
                  and sp1.Ном_Дет = Ассорт.Ном_Дет))
```

Фактически список поставщиков формируется по условию двойного отрицания: в него попадают только те поставщики, для которых в ассортименте нет такой детали, которую они не поставляли! Для иллюстрации многовариантности реализации запросов в SQL приведем второй вариант этого запроса, реализованный с помощью селекции групп:

```
SELECT DISTINCT Поставщик FROM Поставки
      GROUP BY Поставщик
      HAVING COUNT(DISTINCT Ном_Дет) =
            (SELECT COUNT(Ном_Дет) FROM Ассорт)
```

Отметим, что данный запрос реализуется с помощью операции деления отношений Поставки [Ном_Дет:Ном_Дет] Ассотр. Таким образом, здесь приведены два варианта описания операции деления отношений на SQL.

Выдать список студентов, сдавших сессию на 4 и 5:

```
SELECT ФИО FROM Ведом
      WHERE 4 >= ALL
```

```
            (SELECT Ведом.Оценка FROM Ведом w
WHERE ФИО = w.ФИО)
```

6) Предположим, что учет успеваемости на лабораторных занятиях ведется с помощью отношения: Лаб(ФИО, Дисц, Лаб_раб, Оценка)

Выдать список студентов, получивших на экзамене оценку не меньшую, чем хотя бы одна из оценок по лабораторным работам по данной дисциплине

```
SELECT ФИО FROM Ведом
      WHERE Ведом.Оценка >= ANY
             (SELECT Лаб.Оценка FROM Лаб
                  WHERE Ведом.Дисц = Лаб.Дисц AND
                        Ведом.ФИО = Лаб.ФИО)
```

6.6 Объединения

Объединения были введены в стандарте SQL2. Результат объединения используется разделом FROM аналогично списку таблиц.

Пусть: R = {r}; Q = {q}, схемы отношений S_R = {A}; S_Q = {B} имеют эквивалентные наборы атрибутов C. Тогда в условии объединения должны фигурировать атрибуты из набора C.

Шаблон выражения объединения выглядит следующим образом:

```
Левое_отн {INNER | {LEFT | RIGHT | FULL} [OUTER]} JOIN
Правое_отн {ON условие | USING (список столбцов)}
```

Различают внутренние объединения (inner), внешние (outer) – правые (right), левые (left) и полные (full).

Внутреннее объединение. Результат состоит из сцепления кортежей (r, q), для которых справедливо условие. Разновидностью внутреннего является естественное объединение natural inner, при котором из (r, q) удаляются дубликаты значений набора атрибутов C.

Левое внешнее объединение. Результат состоит из сцепления кортежей (r, q). В него входят все кортежи левого отношения R и те кортежи Q, для которых справедливо условие. Если условие не выполняется, то значения атрибутов {B} правого отношения Q устанавливаются в NULL.

Правое внешнее объединение. В результат объединения входят все кортежи правого отношения Q и те кортежи R, для которых справедливо условие. Если условие не выполняется, то значения атрибутов {A} левого отношения R устанавливаются в NULL.

Полное внешнее объединение. В результат объединения входят все кортежи правого и левого отношений. Если условие не выполняется, то значения атрибутов {A} и {B} устанавливаются в NULL поочередно.

Пример. Сформировать сводную ведомость по результатам сессии.

```
SELECT ФИО, Дисц, Оценка FROM
      (Группы NATURAL INNER JOIN Дисц) LEFT JOIN Ведом
            USING (ФИО, Дисц)
```

USING задает атрибуты, равенство значений которых является условием объединения.

6.7 Язык манипулирования данными DML

Манипулирование данными в SQL сводится к трем операциям: добавление (вставки) кортежей (INSERT), удаления (DELETE) и модификации (UPDATE).

6.7.1 Добавление кортежей

Синтаксис оператора вставки кортежа следующий:

```
<insert statement> ::=
   INSERT INTO <table name> [<column names>]
   VALUES <value expressions>
```

Вставляется один кортеж с заданными значениями атрибутов. Если список столбцов отсутствует, то значения в списке должны соответствовать доменам атрибутов в порядке их следования в схеме отношения.

Вставка нескольких кортежей:

```
<insert statement> ::=
   INSERT INTO <table name> [(<column names>)]
   <query expression>
```

Схема результата запроса должна соответствовать списку <column names>, либо, при его отсутствии, схеме отношения.

6.7.2 Удаление кортежей

Оператор удаления кортежа из отношения:

```
<delete statement > ::=
   DELETE FROM <table name>
   [WHERE <search condition>]
```

Таблица T, указанная в разделе FROM оператора DELETE, должна быть изменяемой. На условие поиска накладывается то условие, что на таблицу T

не должны содержаться ссылки ни в каком вложенном подзапросе предикатов раздела WHERE.

Фактически оператор выполняется следующим образом: последовательно просматриваются все строки таблицы T, и те строки, для которых результатом вычисления условия выборки является true, удаляются из таблицы T. При отсутствии раздела WHERE удаляются все строки таблицы T.

6.7.3 Модификация кортежей

Оператор модификации обладает следующим синтаксисом:

```
<update statement> ::=
   UPDATE <table name>
   SET <set clause>
 [{,<set clause>}...]
   [WHERE <search conditions>]
<set clause> ::=
  <column name> =
{ <value expression> | NULL }
```

Таблица T, указанная в операторе UPDATE, должна быть изменяемой. На условие поиска накладывается то условие, что на таблицу T не должны содержаться ссылки ни в каком вложенном подзапросе предикатов раздела WHERE.

Оператор фактически выполняется следующим образом: таблица T последовательно просматривается, и каждая строка, для которой результатом вычисления условия поиска является true, изменяется в соответствии с разделом SET. Если арифметическое выражение в разделе SET содержит ссылки на столбцы таблицы T, то при вычислении арифметического выражения используются значения столбцов текущей строки до их модификации. При отсутствии раздела WHERE модифицируются все строки таблицы T.

Примеры.

1) Зачислить нового студента:

```
INSERT INTO Группы (ФИО, Группа)
     VALUES (“Петров И.И.”, “ИС-Р41”)
```

2) Добавить пустую ведомость по дисциплине «Базы данных» для группы ИС-Р41:

```
INSERT INTO Ведом (ФИО, Дисц)
```

```
SELECT ФИО, Дисц FROM Группы, Дисц
      WHERE Группы.Группа = Дисц.Группа AND
      Дисц.Дисц = "БД" AND
      Группы.Группа = "ИС-Р41"
```

3) Отчислить задолжников, имеющих больше двух неудов:

```
DELETE FROM Группы WHERE ФИО IN
   (SELECT Ведом.ФИО FROM Ведом
      WHERE Оценка = 2
      GROUP BY Ведом.ФИО
      HAVING COUNT(*) >= 2)
```

Отметим, что в данном случае аргументом предиката IN является результат запроса.

4) Зафиксировать пересдачу экзамена по Базам данных студентом И.И. Ивановым:

```
UPDATE Ведом SET Оценка = 4
   WHERE ФИО = "Иванов И.И." AND Дисц = "БД"
```

5) Для учета студентов, получающих стипендию, необходимо добавить атрибут Стипендия в схему отношения Группы:

```
Группы(ФИО, Группа, Стипендия)
```

Начисление стипендии по результатам сессии:

```
UPDATE Группы SET Стипендия = True
   WHERE Группы.ФИО NOT IN
      (SELECT Ведом.ФИО FROM Ведом
         WHERE Оценка IS NOT NULL
            AND Оценка <= 3 )
```

Всем студентам, сдавшим сессию без троек, начисляется стипендия.