1 Создание базы данных

Язык SQL является языком, ориентированным на работы с реляционными базами данных. Традиционно, весь язык SQL разделяется на три части: язык определения данных DDL, язык манипулирования данными DML и язык управления данными DCL. К DDL относятся операторы работы со схемой базы данных, к DML — операторы выборки и изменения данных таблиц, не изменяющие схему, язык DCL состоит, в основном, из операторов управления транзакциями и назначения прав доступа.

1.1 Определение таблиц, домены.

Совокупность имен таблиц, их столбцов и типов столбцов называется сигнатурой или схемой базы данных. Кроме них схема базы данных содержит вторичные элементы, такие как представления, процедуры или права доступа. Схема создается с помощью оператора `create schema`:

```
create schema <имя схемы>
<перечень элементов схемы>
```

Элементы схемы — определения таблиц, представлений, ограничений целостности, доменов и других объектов, образующих единое целое.

Рассмотрим основной оператор языка DDL — определение новой таблицы в схеме базы данных. В простейшем случае оператор имеет такой вид:

```
create table <имя таблицы>
  (<описание столбцов>)
```

Здесь <описание столбцов> — это набор записей следующего вида, разделенных запятыми:

```
<имя столбца> <тип столбца>
```

Стандартные имена типов: `integer` — целые числа, `float` — действительные числа, `char(длина)`, `varchar(длина)` — символьные строки постоянной и переменной, соответственно, длины, `date` — дата. Каждая реализация языка SQL накладывает какие-то свои ограничения на значения этих типов и вводит свои собственные типы данных. Рассмотрим пример определения таблицы с информацией о студентах:

```
create table student(
  l_name char(20),
  f_name char(20),
  m_name char(20),
  nomer char(6),
  gr_nomer integer)
```

В данном примере определяется таблица student, которая имеет 5 столбцов с именами `l_name`, `f_name`, `m_name`, `nomer` и `gr_nomer` соответственно, каждый из первых четырех столбцов хранит строки. Максимальная длина строк в первых трех столбцах — 20 символов, в четвертом — 6 символов. Последний столбец хранит целые числа.

В силу того, что строки отношений не имеют фиксированного порядка следования, отличить какую-то строку от других можно, только основываясь на значениях ее полей. Например, если в вышеприведенной таблице `student` будут две строки вида

```
('Иванов', 'Иван', 'Иванович', '041000', 11),
```

то их невозможно различить. В силу этого, любые изменения, которые предполагается внести в одну из этих строк, неизбежно коснуться и другой. В частности, если мы захотим удалить эту строку, то удалятся сразу обе строки. Такое поведение, обычно, нежелательно. Тем более, что существует способ, с помощью которого от повторяющихся строк в таблице можно легко избавиться.

Рассмотрим пример про книги в библиотеке. Так как количество экземпляров каждой книги в библиотеке обычно больше единицы, то если создавать для каждого экземпляра книги отдельную строку, в таблице появятся повторяющиеся строки. Но вместо того, чтобы иметь в этой таблице отдельную строку для каждого экземпляра книги, можно просто добавить к таблице новый столбец, который и будет хранить количество экземпляров:

code	author	title	city	publishing	year	number
...						
6479234	Оре О.	Теория графов	М.	Наука	1968	20
...						

Удаление строки из исходной таблицы будет соответствовать уменьшению поля number на 1, а добавление повторяющейся строки — увеличению на 1. Этот способ, очевидно, применим к любой таблице, в которой могут быть повторения.

Таким образом, обычной практикой является то, что разные строки таблицы различаются значениями каких-либо полей. Например, в таблице студентов не может быть двух одинаковых строк хотя бы потому, что каждый студент имеет индивидуальный номер студенческого билета. Поэтому, даже если два студента будут иметь абсолютно одинаковые имена, их можно различить по этому номеру. Основываясь на этом принципе, в каждой таблице базы данных обычно выделяют группу полей, называемую первичным ключом, такую, что любые две строки таблицы различаются хотя бы одни полем из этой группы.

Например, в нашей таблице `student` эта группа состоит из одного поля nomer. Для указания первичного ключа в операторе `create table` используется дополнительная декларация:

```
|primary key(<список полей первичного ключа>)
```

В нашем случае это выглядит так:

```
create table student(
  l_name char(20),
  f_name char(20),
  m_name char(20),
  nomer char(6),
  gr_nomer integer,
  primary key(nomer))
```

Теперь система сама должна следить, чтобы значения столбца nomer в разных строках были различны. Определим еще одну таблицу для хранений сведений об успеваемости студентов:

```
create table ball(
  stud_nomer char(6),
  dis char(50),
  dat date,
  form char(10),
  res integer,
  primary key(stud_nomer, dis, dat))
```

В данной таблице `stud_nomer` — номер студенческого билета. Очевидно, что столбцы nomer из таблицы `student` и `stud_nomer` из таблицы `ball` содержат одну и ту же информацию и имеют один и тот же тип. Если бы таблица `student` содержала еще поле

```
phone_nomer char(6)
```

для хранения номера домашнего телефона студента, то это уже был бы другой тип информации, несмотря на совпадение типа данных. Точно так же, столбы `gr_nomer` и `res` в таблицах `student` и `ball` соответственно хранят различного рода информацию. Для того, чтобы более четко разделять столбцы с разной информацией, удобно определить домены. Домен — тип данных, предназначенный для хранения однородной информации. В нашем примере можно создать следующие домены:

- `create domain name_type char(20)` — для хранения имен,
- `create domain nomer_type char(6)` — номера зачетных книжек,
- `create domain dis_type char(50)` — названия дисциплин,
- `create domain form_type char(10)` — типы отчетности,
- `create domain result_type integer` — итоговые оценки,
- `create domain group_type integer` — номера групп.

С учетов этих доменов описания двух предыдущих таблиц будут выглядеть так:

```
create table student(
  l_name name_type,
  f_name name_type,
  m_name name_type,
  nomer nomer_type,
  gr_nomer group_type,
  primary key(nomer))

create table ball(
  stud_nomer nomer_type,
  dis dis_type,
  dat date,
  form form_type,
  res result_type,
  primary key(stud_nomer, dis, dat))
```

Среди всех значений, которые может принимать какое-либо поле, есть одно специальное значение — `NULL`, которое означает отсутствие данных. В ряде случаев, данные в столбце не должны принимать такое значение. В частности, его, как

правило, не должны принимать поля, образующие первичный ключ. Для указания этого свойства поля после его определения пишется `NOT NULL`. То же самое можно написать в определении домена. В нашем примере значение `NULL` по смыслу может содержаться только в полях `dat` и `res` таблицы `ball`, так как только значения этих столбцов заранее неизвестны и вносятся позднее.

Еще один атрибут полей — значение по умолчанию. Иногда бывает так, что одно из значений поля используется гораздо чаще всех остальных или же имеет некоторый специальный смысл. Например, в нашем примере для поля form чаще всего будет использоваться значение 'Экзамен'. Если определить значение по умолчанию для какого-либо поля, то при работе с данными в ряде случаев это значение можно не указывать, что мы потом увидим на примерах. Для указания значения по умолчанию после определения поля пишется слово `DEFAULT` и это значение. То же можно писать и в определении домена. `DEFAULT` можно комбинировать с `NOT NULL`.

Итак, с учетом вышесказанного, для нашей базы данных

следует определить такую схему:

```
create schema students
create domain name_type char(20) NOT NULL
create domain nomer_type char(6) NOT NULL
create domain dis_type char(50) NOT NULL
create domain form_type char(10) NOT NULL
             DEFAULT 'Экзамен'
create domain result_type integer
create domain group_type integer NOT NULL
create table student(
  l_name name_type,
  f_name name_type,
  m_name name_type,
  nomer nomer_type,
  gr_nomer group_type,
  primary key(nomer))

create table ball(
  stud_nomer nomer_type,
  dis dis_type,
  dat date,
  form form_type,
  res result_type,
  primary key(stud_nomer, dis, dat))

create table professor(
  dis_name dis_type,
  gr group_type,
  prof name,
  primary key(dis_name, gr))
```

Последняя таблица содержит имена преподавателей, ведущих предметы в группах.

2 Извлечение данных

2.1 Извлечение информации из одной таблицы.

Основным оператором языка DML является оператор `select` — извлечение данных из таблиц. Результатом выполнения оператора `select` всегда является таблица. В простейшем виде оператор имеет такой формат:

```
select *
  from <имя таблицы>
```

Например,

```
select *
  from student
```
(1)

Данный оператор извлекает все строки из указанной таблице, в примере — из таблицы `student`. Столбцы будут следовать в том же порядке, что и в соответствующей таблице. В указанном примере сначала будет следовать фамилия, затем имя, отчество и, наконец, номер студента и номер группы.

При рассмотрении примеров мы будем считать что в таблице student содержатся следующие сведения:

l_name	f_name	m_name	nomer	gr_nomer
Иванов	Иван	Иванович	010001	11
Иванов	Александр	Петрович	010002	11
Федоров	Николай	Андреевич	011005	12
Сидоров	Павел	Алексеевич	011003	12
Сорокин	Михаил	Анатольевич	011004	12
Сорокин	Николай	Сергеевич	011101	13

В результате выполнения оператора (1) эта таблица и получится.

Чаще всего нужна лишь информация из нескольких столбцов таблицы. Чтобы ограничить извлечение информации

только необходимыми столбцами используется такой синтаксис:

```
select <список столбцов>
  from <имя таблицы>
```

Например, если мы хотим извлечь только номера и фамилии студентов, то следует написать:

```
select nomer, l_name
  from student
```

В данном случае, столбцы в извлекаемых строках будут следовать в том же порядке, в каком они перечислены в операторе `select`. То есть получим такой результат:

nomer	l_name
011004	Сорокин
010001	Иванов
011005	Федоров
011001	Сорокин
010002	Иванов
011003	Сидоров

Столбцы в извлекаемых строках можно переименовать. Для этого после имени столбца следует написать `as` и новое имя. Например, оператор

```
select nomer, l_name as last_name
  from student
```

извлекает из таблицы `student` те же данные, но столбец фамилий будет иметь имя `last_name`:

nomer	last_name
011004	Сорокин
010001	Иванов
011005	Федоров
011001	Сорокин
010002	Иванов
011003	Сидоров

Кроме имен столбцов в списке после `select` можно писать произвольные выражения. Например,

```
select 'студент' as pos, f_name, l_name
  from student
```

выдаст таблицу

pos	f_name	l_name
студент	Иван	Иванов
студент	Александр	Иванов
студент	Павел	Сидоров
студент	Михаил	Сорокин
студент	Николай	Федоров
студент	Николай	Сорокин

Столбцы выражений не имеют имен, поэтому обычно имена им требуется присваивать при помощи as.

Как мы уже говорили, порядок строк в таблицах не определён. Следовательно, приведенные выше операторы `select` выдадут нам информацию в случайном порядке, что и продемонстрировано в предыдущем примере (порядок в каждом случае будет определяться внутренними особенностями реализации). Чтобы в точности указать порядок следования извлеченных строк используется инструкция `order by`:

```
select ...
  from ...
  order by <способ сортировки>
```

Способ сортировки представляет собой список полей (или выражений), разделенных запятыми после каждого из которых может стоять одно из слов: `ASC` — сортировка по данному полю осуществляется по возрастанию, `DESC` — по убыванию. Если после имени поля не стоит ни одно из этих слов, то подразумевается сортировка по возрастанию. Сортировка строк осуществляется сначала по первому полю из списка, если две строки имеют равные значения первого поля, то — по второму и т.д.

Например, чтобы вывести список номеров и фамилий отсортированный в порядке возрастания номеров, следует записать:

```
select nomer, l_name
  from student
  order by nomer
```

тогда получим

nomer	l_name
010001	Иванов
010002	Иванов
011003	Сидоров
011004	Сорокин
011005	Федоров
011101	Сорокин

Если хотим вывести фамилии, имена и отчества студентов, в убывающем порядке:

```
select l_name, f_name, m_name
  from student
  order by l_name desc, f_name desc, m_name desc
```

Следующий шаг — выборка не всех строк, а только нескольких. Простейший случай — когда нужно исключить повторяющиеся строки. В таблице с первичным ключом таких строк, конечно, нет, но если из таблицы выбираются не все столбцы, а только их часть, то они могут появиться. Например, в результате выполнения оператора

```
select l_name
  from student
```

появятся несколько строк с фамилиями 'Иванов' и 'Сорокин'

l_name
Иванов
Иванов
Сидоров
Сорокин
Сорокин
Федоров

Строки в исходной таблице `student` различались, например, из-за наличия поля `nomer`, но так как это поле не выбирается, то строки становятся одинаковыми. Для исключения повторов в этом и любом другом случае используется слово `distinct`, которое ставится после `select`:

```
select distinct ...
  from ...
  [order by ...]
```

Здесь и далее в квадратных скобках мы будем писать необязательные конструкции. В нашем случае, чтобы просто вывести список различных фамилий следует написать:

```
select distinct l_name
  from student
```

и получим

l_name
Иванов
Сидоров
Сорокин
Федоров

Если мы хотим наложить более сложное условие на извлекаемые строки, понадобиться инструкция `where`:

```
select ...
  from ...
  where <условие>
  [order by ... ]
```

Условие может принимать одно из трех значений: TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). В результате выполнения такого оператора выбираются в точности те строки, для которых условие истинно, строки для которых условие ложно или неизвестно отбрасываются.

Условие в простейшем случае представляет собой сравнения полей с константами и другими полями. Простейшие условия можно соединять логическими операциями `and`, `or` и `not` по правилам трехзначной логики:

`and`	T	U	F
T	T	U	F
U	U	U	F
F	F	F	F

`or`	T	U	F
T	T	T	T
U	T	U	U
F	T	U	F

`not`	
T	F
U	U
F	T

Для некоторых условий (`like`, `in`, `between`) можно ставить `not` не перед всем условием, а перед одним из указанным слов, что более соответствует естественному языку. Например, вместо

```
not (a like b)
```

можно писать

```
a not like b
```

Точно так же вместо

```
not (a in (... ))
```

можно написать

```
a not in (... )
```

В качестве сравнений используются равенство (`=`), неравенство (`<>`), отношения порядка (`<`, `>`, `<=`, `>=`). Если одно из сравниваемых значений есть `NULL`, то результат сравнения — UNKNOWN (это же относится и к любым другим условиям).

В противном случае результат всегда является истинным или ложным. Например, если мы хотим извлечь информация о всех студентах, фамилии которых начинаются на букву 'C' нужно написать:

```
select *
  from student
  where l_name >= 'C' and l_name < 'T'
  order by nomer
```

Получится

Сидоров	Павел	Алексеевич	011003	12
Сорокин	Михаил	Анатольевич	011004	12
Сорокин	Николай	Сергеевич	011101	13

Чтобы получить список студентов 12 группы, следует выполнить:

```
select *
  from student
  where gr_nomer = 12
```

Федоров	Николай	Андреевич	011005	12
Сидоров	Павел	Алексеевич	011003	12
Сорокин	Михаил	Анатольевич	011004	12

Если требует список студентов с номерами от '011000' до '011099', то соответствующий оператор:

```
select *
  from student
  where nomer >= '011000' and nomer <= '011099'
  order by nomer
```

Последнее условие принадлежности значения поля некоторому отрезку значений (в нашем случае это отрезок от '011000' до '011099') можно написать с помощью сравнения

between ... and:

```
select *
  from student
  where nomer between '011000' and '011099'
  order by nomer
```

Данный оператор делает то же самое, что и предыдущий.

Для проверки на равенство одному из нескольких значений используется in:

```
|<выражение> in (<список выражений>)
```

Такое условие истинно, если значение выражения равняется значению одного из элементов списка. Например, если нас интересуют студенты только 11 и 13 групп, то их можно выбрать так:

```
select *
  from student
  where gr_nomer in (11, 13)
  order by nomer
```

Для строк предусмотрена проверка соответствия шаблону — like:

```
|<строка> like <шаблон>
```

Шаблон тоже является строкой, в которой символы подчеркивания ('_'), процента ('%') и квадратные скобки ('[', ']') интерпретируются особым образом. Строка считается соответствующей шаблону, если ее можно получить из шаблона с помощью следующих действий, выполненных одновременно:

- каждый символ '_' заменить каким-либо (в точности одним символом),
- каждый символ '%' заменить какой-либо (возможно, пустой) строкой,

- на место каждой строки, заключенной в квадратные скобки поставить один из символов этой строки.

Например, чтобы записать условие, что фамилия начинается с буквы 'С', можно использовать такой синтаксис:

```
l_name like 'С%'
```

Если требуется извлечь информацию о студентах, фамилии которых начинаются на буквы 'С', 'Т' или 'И', можно выполнить следующий оператор:

```
select *
  from student
  where l_name like '[СТИ]%'
```

Чтобы получить информацию о студентах, у которых третья цифра номера — единица:

```
select *
  from student
  where nomer like '__1%'
```