1 Извлечение данных

1.1 Извлечение информации из одной таблицы.

Итак, как мы видели, простейший SQL-запрос импеет вид

SELECT L
FROM R
WHERE C

где L есть список выражений, R есть имя таблицы, а C есть условие.

Смысл такого запроса совпадает со следующим выражением реляционной алгебры:

$$\pi_L(\sigma_C(R)).$$

Другими словами, мы начинаем с выбора в таблице R тех строк, которые удовлетворяют условию C, и затем проектируем эти строки на список атрибутов L.

1.2 Переменные для строчек таблиц

Как мы видели, в некоторых случаях надо сравнивать между собой поля различных строчек одной и той же таблицы. Это можно делать, но нужен способ для ссылок на соответствующие строчки.

Например, если нам нужно выбрать тех студентов, которые сдали Базы данных лучше, чем Информатику, то нужно использовать две копии одной и той же таблицы BALL.

```
select Ball1.stud_nomer
  from Ball Ball1, Ball Ball2
  where Ball1.stud_nomer = Ball2.stud_nomer
  AND Ball1.dis = "Информатика"
  AND Ball2.dis = "Базы данных" AND
  Ball1.res < Ball2.res
```

Здесь Ball1 и Ball2 являются псевдонимами одной и той же таблицы Ball.

1.3 Группировка данных и агрегатные функции.

Теперь рассмотрим возможности SQL по обработке табличных данных. Один из основных способов анализа данных связан с использованием инструкции `group by` в операторе `select`:

```
select ...
  from ...
  [where ... ]
  group by <список группировки>
  [order by ...]
```

Список группировки является списком полей, разделенных запятыми. Семантика оператора `group by` такая: все строки объединяются в несколько групп таким образом, что каждую группу образуют строки, у которых значения всех столбцов их списка группировки совпадают. Затем из каждой группы формируется одна строка.

Например, если в операторе `select` присутствует инструкция вида

```
select l_name ...
  group by l_name ...
```

то в результате выполнения этого оператора мы получим множество строк, в котором разные строки будут иметь различные значения поля `l_name`.

Из описанной семантики `group by` следует, в списке выбираемых полей после `select` нельзя использовать поля, не входящие в список группировки. В самом деле, если написать

```
select l_name, f_name
  from student
  group by l_name
```

то непонятно, чему должно быть равно значение второго столбца. В таблице есть два Ивановых, и значение `f_name` может быть равно и 'Иван', и 'Александр'.

Для того, чтобы из каждой группы выбрать одно значение для какого либо поля или сгенерировать новое значение, используются агрегатные функции: `max` — для выбора максимального значения, `min` — минимального, `avg` — среднего арифметического в группе, `sum` — для нахождения суммы в группе. Последние две функции могут использоваться только для столбцов числового типа. Еще одна агрегатная функция — `count(*)` — предназначена для подсчета количества строк в группе.

Рассмотрим несколько примеров:

```
select gr_nomer, count(*) as gr_count
  from student
  group by gr_nomer
```

11	2
12	3
13	1

В результате выполнения данного оператора будет получен набор строк, каждая из которых будет содержать два поля: первое — номер группы, второе — количество студентов в этой группе.

```
select gr_nomer, min(nomer) as min_nomer,
                 max(nomer) as max_nomer
  from student
  group by gr_nomer
```

11	010001	010002
12	011003	011005
13	011101	011101

Этот оператор для каждой группы вычисляет минимальный и максимальный номера студенческих билетов студентов из этой группы.

Заметим, что столбцы, формируемые с помощью агрегатных функций, не имеют имени, поэтому обычно им требуется

явно присваивать имя при помощи as.

```
select dis, avg(res) as res
  from ball
  group by dis
```

Будем считать, что таблица ball заполнена следующим образом:

stud_nomer	dis	dat	form	res
010001	Алгебра	04.01.2004	Зачет	100
010001	Анализ	08.01.2004	Экзамен	80
010001	Информатика	14.01.2004	Экзамен	75
010002	Алгебра	04.01.2004	Зачет	50
010002	Анализ	08.01.2004	Экзамен	70
010002	Информатика	14.01.2004	Экзамен	80
011003	Алгебра	09.01.2004	Экзамен	30
011003	Анализ	04.01.2004	Экзамен	70
011003	Информатика	14.01.2004	Зачет	90
011004	Алгебра	09.01.2004	Экзамен	60
011004	Анализ	04.01.2004	Экзамен	60
011004	Информатика	14.01.2004	Зачет	70
011005	Алгебра	09.01.2004	Экзамен	80
011005	Анализ	04.01.2004	Экзамен	70
011005	Информатика	14.01.2004	Зачет	80

Тогда приведенный оператор выдаст

dis	res
Алгебра	64
Анализ	70
Информатика	79

Этот оператор для каждой дисциплины вычисляет средний балл, полученный при ее сдаче. Того же можно добиться, записав:

```
select dis, sum(res) / count(*) as res
  from ball
  group by dis
```

Результат будет тем же самым, здесь только явно записан способ вычисления среднего арифметического: вычислить сумму и разделить на количество слагаемых.

Инструкции `group by` и `where` можно использовать вместе. В этом случае сначала осуществляется отбор строк при помощи `where`, а затем — группировка. Например, если мы хотим получить средний балл по каждой дисциплине, полученный при сдаче экзаменов, то следует написать:

```
select dis, avg(res) as res
  from ball
  where form='Экзамен'
  group by dis
```

При выполнении этого оператора сначала будут отобраны строки, в которых значение поля `form` равняется 'Экзамен', а потом среднее арифметическое будет вычисляться только среди значений поля `res` в этих строках.

dis	res
Алгебра	56,6666
Анализ	70
Информатика	77,5

Если бы по каким-то дисциплинам экзамены вообще не проводились, то они вообще не попали бы в результирующий набор. На этом примере видно, что в инструкции `where` можно использовать имена тех столбцов, по которым группировка не проводиться (в нашем случае — `form`). Это как раз и связано с тем, что фильтрация строк в `where` осуществляется до группировки.

Агрегатные функции можно использовать, даже если инструкции `group by` в операторе `select` нет. В этом случае считается, что все строки образуют одну группу. Например, оператор

```
select max(nomer) as max_nomer
  from student
```

вернет таблицу

max_nomer
011101

выбрав максимальный номер студенческого билета из всех, содержащихся в таблице `student`. Точно так же, оператор

```
select avg(res) as res
  from ball
  where form = 'Экзамен'
```

подсчитает среднее количество баллов набираемых студентами в ходе сдачи экзаменов:

res
67

Результатом и в том и другом случае является одна строка.

Как мы уже сказали, отбор строк с помощью инструкции `where` осуществляется до формирования групп. Для того, чтобы отбросить ненужные строки после группировки, используется инструкция `having`:

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

Поскольку `having` выполняется после формирования групп, то в условии после `having` можно использовать агрегатные функции, но нельзя использовать столбцы, не вошедшие в список группировки. Например, чтобы вывести средний балл по дисциплинам, для которых максимальный балл, полученный когда-либо, равен 100, следует выполнить

```
select dis, avg(res)
  from ball
  group by dis
  having max(res) = 100
```

Результат:

dis	
Алгебра	64

Сортировка результата выполняется тоже после группировки, поэтому в списке сортировки можно использовать агрегатные функции, но нельзя столбцы, не участвующие в группировке. Например, можно вывести номера групп в порядке убывания их численности:

```
select gr_nomer, count(*) as cnt
  from student
  group by gr_nomer
  order by count(*) DESC
```

gr_nomer	cnt
12	3
11	2
13	1

1.4 Соединение таблиц.

До сих пор мы использовали оператор `select` для извлечения данных из одной таблицы.

Для того, чтобы извлечь данные из нескольких таблиц сразу, используется соединение таблиц (не путать с рассматриваемым ниже объединением). Под соединением подразумевается операция, при которой к каждой строке одной таблицы приписываются все строки другой, удовлетворяющие некоторым условиям. Результатом соединения является новая таблица, содержащая все столбцы первой таблицы и все столбцы второй.

Соединения делят на внутренние и внешние. При внутреннем соединении все данные получаемой таблицы берутся из исходных таблиц. При внешних соединениях часть данных новой таблицы не соответствует данным исходных.

Самый простой способ внутреннего соединения — декартово произведение исходных таблиц. Декартово произведение записывается двумя способами: соединяя имена таблиц словами `cross join` или запятой:

```
from <имя таблицы> cross join <имя таблицы>
```

или

```
from <имя таблицы>, <имя таблицы>
```

Например,

```
select *
  from student cross join ball
```

или

```
select *
  from student, ball
```

Оба оператора выдадут один и тот же результат.

Простое декартово произведение используется редко. Гораздо чаще требуется только его часть. В нашем примере, нужно чтобы каждая строка таблицы `student` соединялась лишь с теми строками таблицы `ball`, у которых поля содержащие номер студенческого билета совпадают. Этого можно достичь двумя способами: либо в один из предыдущих операторов вставить инструкцию `where` с соответствующим условием:

```
select *
  from student, ball
  where nomer = stud_nomer
```

либо использовать соединение при помощи `inner join ... on`:

```
from <имя таблицы> inner join <имя таблицы>
   on <условие соединения>
```

Для нашего примера:

```
select *
  from student inner join ball                    (1)
    on nomer = stud_nomer
```

Оба этих оператора дают один и тот же результат, показанный на рис.1. Как видим, таблица получилась достаточно громоздкой. С одной стороны она содержит много столбцов, с другой — эти столбцы повторяются. В самом деле, столбцы `nomer` и `stud_nomer` по условию соединения содержат одно и то же. Поэтому если при соединении получается достаточно много столбцов, разумнее в операторе `select` явно перечислить лишь те, которые действительно нужны. Если, например, нас интересует для каждого студента лишь балл по каждой дисциплине, то для выборки можно использовать такой запрос:

```
select student.*, dis, res
  from student inner join ball
    on nomer = stud_nomer
```

Результат его выполнения:

l_name	f_name	m_name	nomer	gr_nomer	dis	res
Иванов	Иван	Иванович	010001	11	Алгебра	100
Иванов	Иван	Иванович	010001	11	Анализ	80
Иванов	Иван	Иванович	010001	11	Информатика	75
Иванов	Александр	Петрович	010002	11	Алгебра	50
Иванов	Александр	Петрович	010002	11	Анализ	70
Иванов	Александр	Петрович	010002	11	Информатика	80
Федоров	Николай	Андреевич	011005	12	Алгебра	80
Федоров	Николай	Андреевич	011005	12	Анализ	70
Федоров	Николай	Андреевич	011005	12	Информатика	80
Сидоров	Павел	Алексеевич	011003	12	Алгебра	30
Сидоров	Павел	Алексеевич	011003	12	Анализ	70
Сидоров	Павел	Алексеевич	011003	12	Информатика	90
Сорокин	Михаил	Анатольевич	011004	12	Алгебра	60
Сорокин	Михаил	Анатольевич	011004	12	Анализ	60
Сорокин	Михаил	Анатольевич	011004	12	Информатика	70

Запись

```
|<имя таблицы> . *
```

l_name	f_name	m_name	nomer	gr_nomer	stud_nomer	dis	dat	form	res
Иванов	Иван	Иванович	010001	11	010001	Алгебра	04.01.2004	Зачет	100
Иванов	Иван	Иванович	010001	11	010001	Анализ	08.01.2004	Экзамен	80
Иванов	Иван	Иванович	010001	11	010001	Информатика	14.01.2004	Экзамен	75
Иванов	Александр	Петрович	010002	11	010002	Алгебра	04.01.2004	Зачет	50
Иванов	Александр	Петрович	010002	11	010002	Анализ	08.01.2004	Экзамен	70
Иванов	Александр	Петрович	010002	11	010002	Информатика	14.01.2004	Экзамен	80
Федоров	Николай	Андреевич	011005	12	011005	Алгебра	09.01.2004	Экзамен	80
Федоров	Николай	Андреевич	011005	12	011005	Анализ	04.01.2004	Экзамен	70
Федоров	Николай	Андреевич	011005	12	011005	Информатика	14.01.2004	Зачет	80
Сидоров	Павел	Алексеевич	011003	12	011003	Алгебра	09.01.2004	Экзамен	30
Сидоров	Павел	Алексеевич	011003	12	011003	Анализ	04.01.2004	Экзамен	70
Сидоров	Павел	Алексеевич	011003	12	011003	Информатика	14.01.2004	Зачет	90
Сорокин	Михаил	Анатольевич	011004	12	011004	Алгебра	09.01.2004	Экзамен	60
Сорокин	Михаил	Анатольевич	011004	12	011004	Анализ	04.01.2004	Экзамен	60
Сорокин	Михаил	Анатольевич	011004	12	011004	Информатика	14.01.2004	Зачет	70

Рис. 1: Результат внутреннего объединения.

означает выбор всех столбцов из указанной таблицы, в нашем случае — из таблицы `student`.

Таблицу можно соединить саму с собой. Допустим, мы хотим получить все пары студентов, которые учатся в одной и той же группе. Это можно сделать следующим образом:

```
select s1.nomer as first, s2.nomer as second
  from student s1 inner join student s2
    on s1.gr_nomer = s2.gr_nomer
```

Здесь к каждой строке таблицы `student` последовательно будут присоединяться строки той же таблицы, у которых значение поля `gr_nomer` такое же. Результат:

first	second
010002	010001
010001	010001
010002	010002
010001	010002
011004	011005
011003	011005
011005	011005
011004	011003
011003	011003
011005	011003
011004	011004
011003	011004
011101	011101
011005	011004

На этом примере также продемонстрирован способ корреляции (временного переименования) исходных таблиц внутри оператора `select`: после имени соответствующей таблицы нужно написать новое (коррелированное) имя:

```
|<имя таблицы> <новое имя>
```

Корреляция действует только на время выполнения оператора `select`. В данном случае в качестве коррелированных имен используются `s1` и `s2`. Корреляция в данном операторе необходима, так как мы соединяем две таблицы, имеющие одно и то же имя, и если их не переименовывать, то будет непонятно, к какой из таблиц относятся поля `gr_nomer` или `nomer`. Для того чтобы указать, к какой таблице относится то или иное поле, используется точка:

```
<имя таблицы> . <имя поля>
```

В качестве имени таблицы используется коррелированное имя, или непосредственное имя таблицы, если корреляция отсутствует. Это бывает нужно не только при соединении таблицы с собой же, но и в случае, если несколько таблиц имеют столбцы с одинаковыми названиями. Скажем, если бы в таблице `ball` мы бы назвали первый столбец `nomer` вместо `stud_nomer`, то оператор (1) потребовалось бы переписать следующим образом:

```
select *
  from student inner join ball
    on student.nomer = ball.nomer
```

«Улучшим» наш предыдущий запрос. В самом деле, с одной стороны не нужно включать в таблицу пары студента с самим собой, с другой — каждая пара студентов присутствует дважды — в разных порядках. Сделаем так, что в паре студенты были различны, и каждая пара встречалась единожды:

```
select s1.nomer as first, s2.nomer as second
  from student s1 inner join student s2
    on s1.gr_nomer = s2.gr_nomer
  where s1.nomer < s2.nomer
```

Получим:

first	second
010001	010002
011003	011004
011003	011005
011004	011005

Чтобы соединить больше чем две таблицы, их нужно соединять последовательно:

```
<имя таблицы> inner join <имя таблицы>
    on <условие>
 inner join <имя таблицы>
    on <условие> ...
```

Предположим, что таблица professor заполнена следующим образом:

dis_name	gr	prof
Анализ	11	Климок
Алгебра	11	Колдунов
Информатика	11	Дудаков
Анализ	12	Хижняк
Алгебра	12	Колдунов
Информатика	12	Волушкова
Практикум на ЭВМ	11	Дудаков
Практикум на ЭВМ	12	Сорокин

В этом случае можно соединить все три таблицы для того, чтобы узнать, у каких преподавателей учился каждый из студентов:

```
select nomer, prof
  from student inner join ball
    on nomer = stud_nomer
  inner join professor
    on dis = dis_name and gr_nomer = gr
```
(2)

nomer	prof
010001	Колдунов
010001	Климок
010001	Дудаков
010002	Колдунов
010002	Климок
010002	Дудаков
011005	Колдунов
011005	Хижняк
011005	Волушкова
011003	Колдунов
011003	Хижняк
011003	Волушкова
011004	Колдунов
011004	Хижняк
011004	Волушкова

Того же можно добиться, записав:

```
select nomer, prof
  from student, ball, professor
  where nomer = stud_nomer and
    dis = dis_name and gr_nomer = gr
```
(3)

При соединении как первая так и вторая таблица может, в свою очередь, быть соединением. То есть предыдущий запрос можно переписать еще и так:

```
select nomer, prof
  from student inner join
      ball inner join professor on dis = dis_name
    on nomer = stud_nomer and gr_nomer = gr
```

В этом случае сначала соединяются таблицы `ball` и `professor`, а затем к ним присоединяется таблица `student`. В запросе (2) происходит наоборот: сначала соединяются таблицы `student` и `ball`, а затем к результату присоединяется `professor`. Для запроса (3) последовательность соединения выбирает исполнитель запроса (планировщик).

Ввиду того, что декартово произведение ассоциативно, ассоциативно и внутреннее соединение. То есть, неважно, в какой последовательности выполняются операторы внутреннего соединения: слева направо или наоборот.

При выполнении внутреннего соединения может оказаться так, что для каких-то строк одной таблицы не нашлось соответствующих строк в другой. Такие строки при внутреннем соединении отбрасываются.

Внешнее соединение предназначено для того, что сохранить все строки одной из таблиц или обеих сразу. Синтаксис конструкций: левое внешнее соединение —

```
<имя таблицы> left outer join <имя таблицы>
  on <условие>
```

правое внешнее соединение —

```
<имя таблицы> right outer join <имя таблицы>
  on <условие>
```

полное внешнее соединение —

```
<имя таблицы> full outer join <имя таблицы>
  on <условие>
```

В первом случае все строки первой таблицы попадут в результат соединения, во втором — все строки второй таблицы, в третьем — все строки и той и другой будут присутствовать.

Рассмотрим левое соединение, при котором первая таблица попадает в результат полностью. Если для какой-то строки первой таблицы нашлась хоть одна строка во второй, которая удовлетворяет условию соединения, то внешнее соединение ведет себя для этой строки так же, как внутреннее. Если же для строки первой таблицы пары из второй не нашлось, то такая строка тоже включается в результат, но все поля второй таблицы для этой строки принимают значение `NULL`, означающее отсутствие данных.

Например, можно найти для преподавателей средний балл, который получают у них студенты:

```
select prof, avg(res)
  from professor left outer join
      (ball inner join student
              on nomer = stud_nomer)
    on dis_name=dis and gr_nomer = gr
  group by prof
```

В этом примере с помощью левого внешнего соединения соединяются две таблицы: первая — `professor`, а вторая получена внутренним соединением таблиц `ball` и `student`. Результат:

prof	
Волушкова	80
Дудаков	77
Климок	75
Колдунов	64
Сорокин	NULL
Хижняк	66

Любые действия с `NULL` в результате дают `NULL`, в частности при нахождении среднего арифметического `avg` результатом будет `NULL`, если `NULL` встречается хотя бы в одной строке группы. Для проверки, имеет ли некоторое выражение значение `NULL`, используется предикат

```
|is null
```

Например, чтобы найти все предметы, оценки за которые еще не внесены в базу данных:

```
select dis_name, gr, prof
  from professor left outer join
      (ball inner join student
           on nomer = stud_nomer)
    on dis_name=dis and gr_nomer = gr
  where res is null
```

В результате выполнения этого оператора в столбце `res` окажутся значения `NULL` в тех строках, для которых отсутствуют строки в таблице `ball`.

dis_name	gr	prof
Практикум на ЭВМ	11	Дудаков
Практикум на ЭВМ	12	Сорокин

Левое внешнее соединение не ассоциативно ни с самим собой, ни с другими видами соединений. Рассмотрим, например, следующие три таблицы:

Таблица a	Таблица b		Таблица c	
a1	b1	b2	c1	c2
1	1	11	1	12

При выполнении соединения

```
a left outer join b on a1 = b1
  left outer join c on a1 = c1 and b2 = c2
```

получим таблицу вида

a1	b1	b2	c1	c2
1	1	11	NULL	NULL

Если же порядок соединения поменять:

```
a left outer join
    b left outer join c on b2 = c2
  on a1 = c1 and a1 = b1
```

то получается другой результат:

a1	b1	b2	c1	c2
1	NULL	NULL	NULL	NULL

Таким образом, если при соединении более чем двух таблиц используется внешнее соединение, то на порядок выполнения соединений нужно обращать тщательное внимание.

Правое внешнее соединение отличается от левого только тем, что в результат включаются все строки из правой таблицы, а значение `NULL` записывается в столбцы левой таблицы, если соответствующие строки в ней отсутствуют. Полное внешнее соединение сохраняет все строки и той и другой таблицы.

1.5 Подзапросы.

Самое мощное средство языка SQL — вложенные запросы (или подзапросы). Самой общей формой использования подзапросов является использование его в качестве условия в инструкциях `where` или `having`:

```
exists(<SQL-запрос>)
```

Это условие будет истинным, если таблица, сгенерированная в результате выполнения SQL-запроса в скобках, содержит хотя бы одну строку, в противном случае условие ложно. Частные случаи использования подзапросов — сравнения:

```
<выражение> <знак сравнения> any
    (<SQL-запрос>)
```

и

```
<выражение> <знак сравнения> all
    (<SQL-запрос>)
```

В обоих случаях SQL-запрос должен возвращать таблицу с одним столбцом, значения которого можно сравнивать с выражением. Первое условие будет истинным, если сравнение выполняется хотя бы для одной строки сгенерированной таблицы, второе — если условие выполняется для всех строк. В частности, если вложенный запрос вернет пустую таблицу, то первое условие ложно, а второе истинно. Если вложенный запрос возвращает ровно одну строку, то оба условия эквивалентны.

Рассмотрим несколько примеров. Найдем студента, который получил самый высокий балл:

```
select l_name, gr_nomer, nomer, dis, res
  from student inner join ball
    on nomer = stud_nomer
  where res = all(select max(res) from ball)
```

l_name	gr_nomer	nomer	dis	res
Иванов	11	010001	Алгебра	100

В данном случае для выполнения подзапроса не используется никакая информация

из таблиц основного запроса. Такие подзапросы называют несвязанными.

Изменим запрос, чтобы найти студентов, получивших самый высокий балл по каждому из предметов:

```
select l_name, gr_nomer, nomer, dis, res
  from student inner join ball
    on nomer = stud_nomer
  where res = all(
    select max(res)
      from ball b
      where ball.dis = b.dis)
```

l_name	gr_nomer	nomer	dis	res
Сидоров	12	011003	Информатика	90
Иванов	11	010001	Анализ	80
Иванов	11	010001	Алгебра	100

Здесь при выполнении подзапроса используется поле `dis` из таблицы `ball` основного запроса. Такие подзапросы называют связанными. В связи с использованием таблицы `ball` и во внешнем запросе, и в подзапросе возникает необходимость переименования таблицы вложенного запроса, чтобы отличать ее от таблицы основного запроса, и необходимость явного указания таблиц для поля `dis`.

Чтобы вывести все оценки по каждому предмету, кроме самых низких:

```
select l_name, gr_nomer, nomer, dis, res
  from student inner join ball
    on nomer = stud_nomer
  where res > any(
    select min(res)
      from ball b
      where ball.dis = b.dis)
```

l_name	gr_nomer	nomer	dis	res
Иванов	11	010002	Алгебра	50
Сорокин	12	011004	Алгебра	60
Федоров	12	011005	Алгебра	80
Иванов	11	010001	Алгебра	100
Иванов	11	010002	Анализ	70
Сидоров	12	011003	Анализ	70
Федоров	12	011005	Анализ	70
Иванов	11	010001	Анализ	80
Иванов	11	010001	Информатика	75
Иванов	11	010002	Информатика	80
Федоров	12	011005	Информатика	80
Сидоров	12	011003	Информатика	90

Каждый из предыдущих запросов можно записать при помощи `exists`:

```
select *
  from student inner join ball
    on nomer = stud_nomer
  where not exists(
    select * from ball b where b.res > ball.res)
```

```
select *
  from student inner join ball
    on nomer = stud_nomer
  where not exists(
    select *
      from ball b
      where ball.dis = b.dis and b.res > ball.res)

select *
  from student inner join ball
    on nomer = stud_nomer
  where exists(
    select *
      from ball b
      where ball.dis = b.dis and b.res < ball.res)
```

Еще одна форма использования подзапросов — условие:

```
|<выражение> in (<SQL-запрос>)
```

Здесь, как и в сравнениях, `any` и `all` SQL-запрос должен возвращать таблицу с одним столбцом, тип которого совместим с выражением. Условие считается истинным, если этот столбец содержит значение выражения. То есть, приведенное условие эквивалентно такому:

```
<выражение> = any(<SQL-запрос>)
```

Например, чтобы выбрать студентов, которые ни за один предмет не получили меньше 50 баллов, можно воспользоваться запросом:

```
select *
  from student
  where nomer not in (
    select stud_nomer from ball where res < 50)
```

1.6 Объединение таблиц

Предположим, мы хотим напечатать список пригласительных билетов на вечер, куда приглашаются и преподаватели, и студенты. Сложность создания такого списка в том, что фамилии преподавателей и студентов хранятся в разных таблицах, а в результате они должны составить один столбец таблицы приглашенных. Для выполнения этой задачи лучше всего использовать инструкцию объединения `union`

```
select ...
  union
select ...
  union
...
```

Все операторы `select`, участвующие в инструкции `union`, должны генерировать таблицы с одинаковым количеством столбцов и одинаковыми типами столбцов. При этом следует иметь ввиду, что инструкция `union` неявно применяет `distinct` к получаемой таблице, то есть она отбрасывает повторяющиеся строки. Например, если мы напишем

```
select l_name from student
  union
select prof from professor
```

То получим:

l_name
Волушкова
Дудаков
Иванов
Климок
Колдунов
Сидоров
Сорокин
Федоров
Хижняк

Получилось не совсем то, что нужно, так как вместо трех Сорокиных образовалась одна строка. Чтобы `union` не отбрасывал повторения нужно к нему добавлять слово `all`:

```
select l_name from student
  union all
select prof from professor
```

Тогда получается

l_name
Иванов
Иванов
Сидоров
Сорокин
Федоров
Сорокин
Колдунов
Колдунов
Климок
Хижняк
Дудаков
Волушкова
Дудаков
Сорокин

Теперь повторяются фамилии преподавателей. Чтобы этого избежать, как мы уже говорили следует использовать слово distinct в запросе, выбирающем фамилии преподавателей:

```
select l_name from student
  union all
select distinct prof from professor
```

l_name
Иванов
Иванов
Сидоров
Сорокин
Федоров
Сорокин
Волушкова
Дудаков
Климок
Колдунов
Сорокин
Хижняк

Теперь все правильно. Для удобства желательно отсортировать результат. Если с помощью `union` или `union all` несколько запросов соединяются в один, то инструкция сортировки `order by` ставиться в самом конце, чтобы отсортировать все результаты объединения:

```
select ...
  union [all]
select ...
...
  union [all]
select ...
order by ...
```

Для нашего примера

```
select l_name from student
  union all
select distinct prof from professor
order by l_name
```

Получим

l_name
Волушкова
Дудаков
Иванов
Иванов
Климок
Колдунов
Сидоров
Сорокин
Сорокин
Сорокин
Федоров
Хижняк

Инструкция `union` выполняет объединение таблиц как множеств строк. Точно так же используются инструкции `intersect` и `except`, означающие пересечение и вычитание соответсвенно. Например, чтобы узнать, какие фамилии встречаются как среди студентов, так и среди преподавателей, можно выполнить запрос

```
select l_name from student
  intersect
select prof from professor
order by l_name
```

l_name
Сорокин

1.7 Представления

Представления — это запросы, которые хранятся в базе данных и используются как таблицы. Когда представление используется в качестве таблицы, то вместо него подставляется

результат выполнения запроса. Для того чтобы отличать таблицы, реально хранящиеся в базе данных, от представлений реально хранящиеся таблицы называют основными.

Представление создается с помощью оператора `create view`:

```
create view <имя представления> as
  <SQL-запрос>
```

Например, представление, содержащее информацию о сдаче студентами экзамена по анализу:

```
create view analysis as
  select stud_nomer, dat, res from ball
  where dis = 'Анализ'
```

Напомним, что таблицы является неупорядоченной совокупностью строк, то же должно относиться и к представлениям: порядок следования строк в представлении не определен. Поэтому в операторе SQL, образующем представление, инструкции `order by` быть не должно. Другое важное условие на SQL-запрос: все его столбцы должны иметь имена. Если в представлении в качестве столбцов используются какие-то выражения, то они обязательно должны быть поименованы с помощью `as`.

Предположим, мы хотим создать представление для среднего балла каждого студента:

```
create view avg_ball as
  select stud_nomer, avg(res) as avg_res
  from ball
  group by stud_nomer
```

Теперь представление `avg_ball` можно использовать как таблицу, содержащую два столбца: `stud_nomer` и `avg_res`. Например, чтобы вывести ее содержимое:

```
select *
  from avg_ball
```

stud_nomer	avg_res
010001	85
010002	66
011003	63
011004	63
011005	76

При изменении основной таблицы `ball` изменения автоматически отразятся и в представлении `avg_ball`. Представления можно использовать как часть соединения как с основными таблицами, так и друг с другом.

Другой пример использования представлений — двойная группировка данных. С помощью одного оператора `select` можно выполнить только одну группировку, например, найти средний балл для каждого студента. Если же требуется еще и среди этих средних баллов найти самый высокий, то для этого придется использовать представление, так как вложенные агрегатные функции язык SQL не допускает:

```
select max(avg_res)
  from avg_ball
```

Нельзя написать непосредственно:

```
select max(avg(res))
  from ball
  group by stud_nomer
```

Однако, SQL допускает в качестве исходных таблиц запроса использовать другие запросы. Тогда наш запрос можно переписать в виде:

```
select max(avg_res)
from (select avg(res) as avg_res
       from ball
       group by stud_nomer)
```

То есть, вместо имени представления в инструкции `from` просто пишется SQL запрос, реализующий это представление.

2 Изменение данных.

2.1 Изменение данных основных таблиц.

Самая простая операция изменения данных — вставка новой строки:

```
insert into <имя таблицы>(<список полей>)
  values (<список значений>)
```

Например, чтобы добавить в таблицу `ball` новую строку, нужно записать следующее:

```
insert into ball(stud_nomer, dis, dat, form, res)
  values ('010001', 'Практикум на ЭВМ',
            '29.12.2003', 'Зачет', 85)
```

Если какой-либо столбец таблицы в списке полей не указан, то будет попытка вставить на его место значение по умолчанию, если оно есть, или `NULL`, если значение по умолчанию отсутствует.

Например, оператор

```
insert into ball(stud_nomer, dis, dat, res)
  values ('010001', 'Практикум на ЭВМ',
                '29.12.2003', 85)
```

приведет к вставке строки

010001	Практикум на ЭВМ	04.01.2004	Экзамен	100

Оператор

```
insert into ball(stud_nomer, dis)
  values ('010001', 'Практикум на ЭВМ')
```

вставит

010001	Практикум на ЭВМ	NULL	Экзамен	NULL

А оператор

```
insert into ball(stud_nomer)
  values ('010001')
```

приведет к ошибке и не выполнится, так как для поля `dis` значение по умолчанию не задано, а значение `NULL` это поле не допускает.

Более сложный оператор — удаление данных из таблицы:

```
delete from <имя таблицы>
  [where <условие>]
```

Если условие отсутствует, то удаляются все строки из таблицы. Если оно записано, то удаляются только строки, ему удовлетворяющие.

Например, чтобы удалить всю информацию из таблицы `ball` о студенте с номером '010001', следует написать:

```
delete from ball
  where stud_nomer = '010001'
```

Правила написания условия такие же, как в инструкции `where` оператора `select`.

Оператор изменения существующих строк:

```
update <имя таблицы>
  set <имя поля> = <значение>
    [,<имя поля> = <значение>
    [, ... ]]
  [where <условие>]
```

При выполнении оператора в указанные после слова `set` поля записываются соответствующие значения. Если присутствует инструкция `where`, то изменения касаются лишь строк, удовлетворяющих условию, иначе — всех строк таблицы.

Если, например, нужно изменить номер студента с '010001' на '010010', то следует изменить таблицы `student` и `ball`:

```
update student
  set nomer = '010010'
  where nomer='010001'

update ball
  set stud_nomer = '010010'
  where stud_nomer='010001'
```

В качестве новых значений для полей могут использоваться выражения с участием полей. Если нужно увеличить все баллы за экзамен по информатике на 5, то это можно сделать так:

```
update ball
  set res = res + 5
  where dis = 'Информатика' and form = 'Экзамен'
```

Как и в операторе `select`, в инструкциях `where` операторов `delete` и `update` можно использовать подзапросы. Например, чтобы студентам, набравшим наибольшее число баллов на экзамене по информатике, установить результат в 100 баллов, можно выполнить такой оператор:

```
update ball
  set res = 100
  where dis = 'Информатика' and
    form = 'Экзамен' and res = any(
      select max(res)
        from ball
        where dis = 'Информатика' and
          form = 'Экзамен')
```

2.2 Изменение данных представлений.

Гарантируется, что представления, построенные с помощью SQL-запросов к одной основной таблице и без использования группировки, можно изменять, подобно основной таблице. Это легко объяснимо — такие представления просто являются частью основной таблицы и изменения, вносимые в эти представления, легко преобразуются в изменения основной таблицы.

Например, представление `analisys` обязательно является обновляемым. Можно, например, выполнить оператор удаления данных:

```
delete from analysis
  where stud_nomer = '010001'
```

или оператор изменения:

```
update analysis
  set dat = '07.01.2004'
```

Первый из этих операторов просто удалит из основной таблицы `ball` строки, удовлетворяющие указанному условию, а второй — изменит поле `dat` на '07.01.2004' во всех строках таблицы `ball`, которые попали в представление `analysis`.

Более сложная ситуация с оператором `insert`. Для его использования могут возникнуть два препятствия. Первое из них: представление содержит не все столбцы исходной таблицы. В частности, наше представление `analysis` содержит 3 из 5 столбцов таблицы `ball`. Поэтому, если попытаться выполнить оператор вставки

```
insert into analysis(stud_nomer, dat, res)
  values ('011101', '04.01.2004', 60)
```

то возникнут проблемы с заполнением полей `dis` и `form` таблицы `ball`. Ситуация разрешается с помощью записи в такие поля значения по умолчанию или `NULL`, если значения по умолчанию не определены. В нашем случае для `form` будет использоваться значение по умолчанию ('Экзамен'), а для `dis` — `NULL`. Но так как поле `dis` не допускает значений `NULL`, то этот оператор вставки приведет к ошибке и не будет выполнен.

Другой подводный камень добавления строк в представлении — появление строк, не удовлетворяющих условиям представления. Предположим, что представление `analysis` определено так:

```
create view analysis as
  select stud_nomer, dis, dat, res
    from ball
    where dis = 'Анализ'
```

Если теперь в это представления вставить строку, для которой поле `dis` будет иметь значение 'Алгебра':

```
insert into analysis(stud_nomer, dis, dat, res)
  values('011101', 'Алгебра', '15.01.2004', 60)
```

то формально никакой ошибки не будет, эта строка вставиться в таблицу `ball`. Но при повторном выборе данных из представления `analysis` эта строка не появится, так как не удовлетворяет условию представления. Получается ситуация, когда мы вроде бы вставляем данные, но не видим их. То же самое может случиться при использовании оператора `update`: можно изменить данные таким образом, что они перестанут удовлетворять условию представления и окажутся как бы удаленными из него.

Если нужно избежать таких проблем, представление создается с параметром `with check option`:

```
create view as ...
  with check option
```

Например,

```
create view analysis as
  select stud_nomer, dis, dat, res from ball
  where dis = 'Анализ'
  with check option
```

Представления, созданные таким образом, не допускают вставку данных, не удовлетворяющих условию представления. То есть, приведенный выше оператор `insert` для последнего представления вызовет ошибку и не вставит новой строки. Точно так же через представление `analysis` нельзя произвести изменение существующих строк, в результате которого строки перестанут удовлетворять условиям представления. Например, оператор

```
update analysis
  set dis = 'Алгебра'
```

вызовет ошибку.