**Лабораторная работа № 3  
1. Создание простых запросов на выборку**

**Теоретические сведения**

Рассмотрим следующие вопросы:

* выборка данных из одной таблицы с помощью оператора *SELECT*;
* использование в запросах операторов и встроенных функций MySQL.

Для выполнения запросов (извлечения строк из одной или нескольких таблиц БД) используется оператор *SELECT*. Результатом запроса всегда является таблица. Результаты запроса могут быть использованы для создания новой таблицы. Таблица, полученная в результате запроса, может стать предметом дальнейших за­просов.

Общая форма оператора *SELECT*:

*SELECT столбцы FROM таблицы*

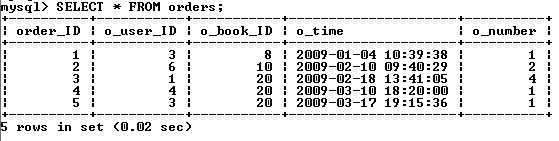
*[WHERE условия]*

*[GROUP BY группа [HAVING групповые\_условия] ]*

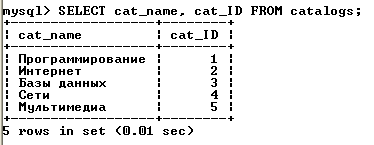
*[ORDER BY имя\_поля]*

*[LIMIT пределы];*

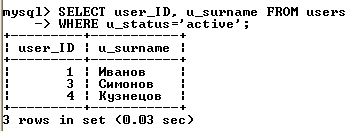
Оператор *SELECT* имеет много опций. Их можно использовать или не исполь­зовать, но они должны указываться в том порядке, в каком они приведены. Если требуется вывести все столбцы таблицы, необязательно перечислять их после ключевого слова *select*, достаточно заменить этот список символом \*.



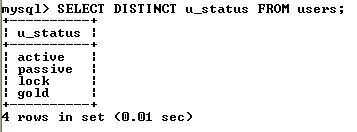
Список столбцов в операторе *select* используют, если нужно изменить порядок следования столбцов в результирующей таблице или выбрать часть столбцов.



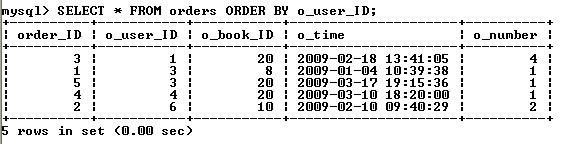
**Условия выборки**. Гораздо чаще встречается ситуация, когда необходимо изменить количество выводимых строк. Для выбора записей, удовлетворяющих определен­ным критериям поиска, можно использовать конструкцию *WHERE*.



В запросе можно использовать ключевое слово *DISTINCT*, чтобы результат не содержал повторений уже имеющихся значений, например:

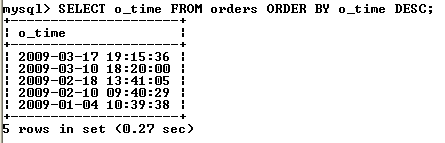


**Сортировка**. Результат выборки – записи, расположенные в том порядке, в котором они хранятся в БД. Чтобы отсортировать значения по одному из столбцов, необходимо после конструкции *order by* указать этот столбец, например:



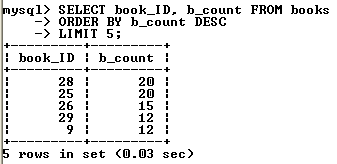
Сортировку записей можно производить по нескольким столбцам (их следует указать после слов *order by* через запятую). Число столбцов, указываемых в конструкции *order by*, не ограничено.

По умолчанию сортировка производится в прямом порядке (записи располагаются от наименьшего значения поля сортировки до наибольшего). Обратный порядок сортировки реализуется с помощью ключевого слова *desc*:

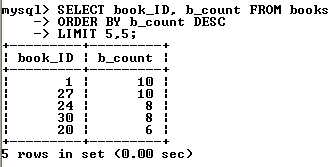


Для прямой сортировки существует ключевое слово *asc*, но так как записи сортируются в прямом порядке по умолчанию, данное ключевое слово опускают.

**Ограничение выборки**. Результат выборки может содержать тысячи записей, вывод и обработка которых занимают значительное время. Поэтому информацию часто разбивают на страницы и предоставляют ее пользователю частями. Постраничная навигация используется при помощи ключевого слова *limit*, за которым следует число выводимых записей. Следующий запрос извлекает первые 5 записей, при этом осуществляется обратная сортировка по полю *b\_count*:



Для извлечения следующих пяти записей используется ключевое слово *limit* с двумя цифрами. Первая указывает позицию, начиная с которой необходимо вернуть результат, вторая цифра – число извлекаемых записей, например:



При определении смещения нумерация строк начинается с нуля (по­этому в последнем примере для шестой строки указано смещение 5).

**Группировка записей**. Конструкция *GROUP ВУ* позволяет группировать извлекаемые строки. Она полезна в комбинации с функциями, применяемыми к группам строк. Эти функции (табл. 6) называются агрегатами (суммирующими функциями) и вычисляют одно значение для каждой группы, создаваемой конструкцией *group by*. Функции позволяют узнать число строк в группе, подсчитать среднее значение, получить сумму значений столбцов. Результирующее значение рассчитывается для значений, не равных *null* (исключение – функция *count(\*)*). Допустимо использование этих функций в запросах без группировки (вся выборка – одна группа).

Пример использования функции *count( )*, которая возвращает число строк в таблице, значения указанного столбца для которых отличны от *NULL*:

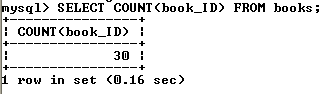
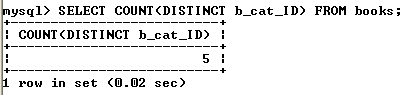


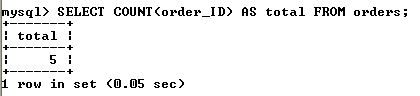
Таблица 6

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| *AVG ( [DISTINCT]  expr)* | Возвращает среднее значение аргумента *expr*. В качестве аргумента обычно выступает имя столбца. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |
| *COUNT ( )* | Подсчитывает число записей и имеет несколько форм. Форма *COUNT (выражение)* возвращает число записей в таблице, поле *выражение* для которых не равно *null*. Форма *count(\*)* возвращает общее число строк в таблице независимо от того, принимает какое-либо поле значение *null* или нет. Форма *COUNT (DISTINCT выражение1, выражение2, ... )* позволяет использовать ключевое слово *distinct*, которое позволяет подсчитать только уникальные значения столбца |
| *MIN ( [DISTINCT]  expr)* | Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце *expr*. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |
| *MAX ( [DISTINCT]  expr)* | Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце *expr*. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |
| *STD (expr)* | Возвращает стандартное среднеквадратичное отклонение в аргументе *expr* |
| *STDDEV\_SAMP (expr)* | Возвращает выборочное среднеквадратичное отклонение в аргументе *expr* |
| *SUM ( [DISTINCT]  expr)* | Возвращает сумму величин в столбце *expr*. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |

Использование ключевого слова *distinct* с функцией *count( )* позволяет вернуть число уникальных значений *b\_cat\_ID* в таблице *books*, например:



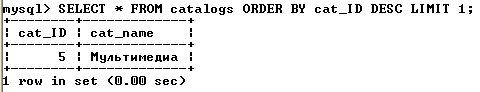
В *SELECT*-запросе столбцу можно назначить новое имя с помощью оператора *as*. Например, результату функции *count( )* присваивается псевдоним *total*:



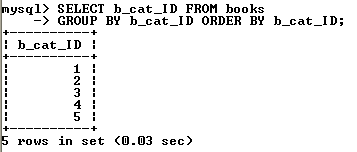
Использование функций в конструкции *where* приведет к ошибке. В следующем примере показана попытка извлечения из таблицы *catalogs* записи с максимальным значением поля *cat\_ID*:



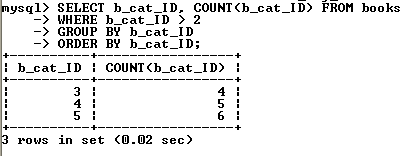
Решение задачи следует искать в использовании конструкции *order by*:



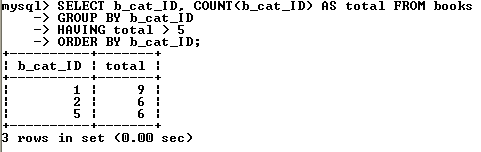
Для извлечения уникальных записей используют конструкцию *group by* с именем столбца, по которому группируется результат:



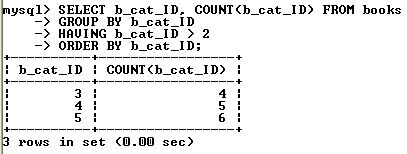
При использовании *group by* возможно использование условия *where*:



Часто при задании условий требуется ограничить выборку по результату функ­ции (например, выбрать каталоги, где число товарных позиций больше 5). Использование для этих целей конструкции *where* приводит к ошибке. Для решения этой проблемы вместо ключевого слова *where* используется ключевое слово *having*, располагающееся за конструкцией *group by*:



Запрос, извлекающий уникальные значения столбца *b\_cat\_ID*, большие двух:



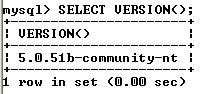
При этом в случае использования ключевого слова *where* сначала производится выборка из таблицы с применением условия и лишь затем группировка результата, а в случае использования ключевого слова *having* сначала происходит группировка таблицы и лишь затем выборка с применением условия. Допускается использование условия *having* без группировки *group by*.

**Использование функций**. Для решения специфических задач при выборке удобны встроенные функции MySQL. Большинство функций предназначено для использования в выражениях *SELECT* и *WHERE*. Существуют также специальные функции группировки для использования в выражении *GROUP BY* (см. выше).

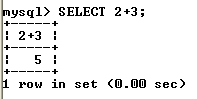
Каждая функция имеет уникальное имя и может иметь несколько аргументов (перечисляются через запятую в круглых скобках). Если аргументы отсутствуют, круглые скобки все равно следует указывать. Пробелы между именем функции и круглыми скобками недопустимы.

Число доступных для использования функций велико, в приложениях приведены наиболее полезные из них.

Пример использования функции, возвращающей версию сервера MySQL:



Отметим также возможность использования оператора *SELECT* без таблиц вообще. В такой форме *SELECT* можно использовать как калькулятор:



Можно вычислить любое выражение без указания таблиц, получив доступ ко всему разнообразию математических и других операторов и функций. Возможность выполнять математические расчеты на уровне *SELECT* позволяет проводить финансовый анализ значений таблиц и отображать полученные результаты в отчетах. Во всех выражениях MySQL (как в любом языке программирования) можно использовать скобки, чтобы контролировать порядок вычислений.

**Операторы**. Под операторами подразумеваются конструкции языка, которые производят преобразование данных. Данные, над которыми совершается операция, называются операндами.

В MySQL используются три типа операторов:

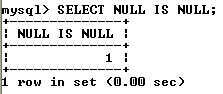
* арифметические операторы;
* операторы сравнения;
* логические операторы.

*Арифметические операции.*В MySQL используются обычные арифметические операции: сложение (+), вычитание (–), умножение (\*), деление (/) и целочисленное деление *DIV* (деление и отсечение дробной части). Деление на 0 дает безопасный результат *NULL*.

*Операторы сравнения.* При работе с операторами сравнения необходимо помнить о том, что, за ис­ключением нескольких особо оговариваемых случаев, сравнение чего-либо со зна­чением *NULL* дает в результате *NULL*. Это касается и сравнения значения *NULL* со значением *NULL*:



Корректнее использовать следующий запрос:



Поэтому следует быть предельно внимательными при работе с операторами сравнения, если операнды могут принимать значения *NULL*.

Наиболее часто используемые операторы сравнения приведены в табл. 7.

*Логические операторы.* MySQL поддерживает все обычные логические операции, которые можно использовать в выражениях. Логические выражения в MySQL могут принимать значения 1 (истина), 0 (ложь) или *NULL*.

Кроме того, следует учитывать, что MySQL интерпретирует любое ненулевое значение, отличное от *NULL*, как значение «истина». Основные логические операторы приведены в табл. 8.

**Практическая работа**

При выполнении лабораторной работы необходимо:

* для заданной предметной области построить не менее десяти простых запросов на выборку с использованием операторов и функций MySQL;

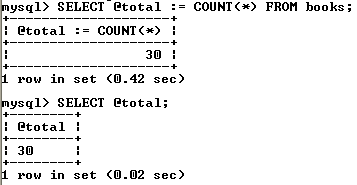
Таблица 7

|  |  |
| --- | --- |
| **Оператор** | **Значение** |
| **=** | Оператор равенства. Возвращает 1 (истина), если операнды равны, и 0 (ложь), если не равны |
| <=> | Оператор эквивалентности. Аналогичен обычному равенству, но возвращает только два значения: 1 (истина) и 0 (ложь). *NULL* не возвращает |
| <> | Оператор неравенства. Возвращает 1 (истина), если операнды не равны, и 0 (ложь), если равны |
| < | Оператор «меньше». Возвращает 1 (истина), если левый операнд меньше правого, и 0 (ложь) – в противном случае |
| <= | Оператор «меньше или равно». Возвращает 1 (истина), если левый операнд меньше правого или они равны, и 0 (ложь) – в противном случае |
| > | Оператор «больше». Возвращает 1 (истина), если левый операнд больше правого, и 0 (ложь) – в противном случае |
| >= | Оператор «больше или равно». Возвращает 1 (истина), если левый операнд больше правого или они равны, и 0 (ложь) – в противном случае |
| *n BETWEEN min   AND max* | Проверка диапазона. Возвращает 1 (истина), если проверяемое значение *n* находится между *min* и *max*, и 0 (ложь) – в противном случае |
| *IS NULL* и  *IS NOT NULL* | Позволяют прове­рить, является ли значение значением *NULL* или нет |
| *n IN (множество)* | Принадлежность к множеству. Возвращает 1 (истина), если проверяемое значение *n* входит в список, и 0 (ложь) – в противном случае. В качестве множества может использоваться список литеральных значений или выражений или подзапрос |

Таблица 8

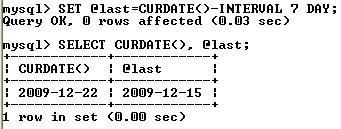
|  |  |  |
| --- | --- | --- |
| **Оператор** | **Пример** | **Значение** |
| *AND* | *n AND m* | Логическое *И*: истина *AND* истина = истина,  ложь *AND* любое = ложь. Все остальные выражения оцениваются как *NULL* |
| *OR* | *n OR m* | Логическое *ИЛИ*: истина *OR* любое = истина,  *NULL* *OR* ложь = *NULL,*  *NULL OR NULL* = *NULL,* ложь *OR* ложь = ложь |
| *NOT* | *NOT n* | Логическое НЕТ: *NOT* истина = ложь, *NOT* ложь = истина.  *NOT NULL = NULL* |
| *XOR* | *n XOR m* | Логическое *исключающее ИЛИ*: истина *XOR* истина = ложь,  истина *XOR* ложь = истина, ложь *XOR* истина = истина,  ложь *XOR* ложь = ложь,  *NULL XOR* любое *= NULL*, любое *XOR NULL = NULL* |

**Переменные SQL и временные таблицы**. Часто результаты запроса необходимо использовать в последующих запросах. Для этого полученные данные необходимо сохранить во временных структурах. Эту задачу решают переменные SQL и временные таблицы. Объявление переменной начинается с символа @, за которым следует имя переменной. Значения переменным присваиваются посредством оператора *select* с использованием оператора присваивания := . Например:



Объявляется переменная @*total*, которой присваивается число записей в таблице *books*. Затем в рамках текущего сеанса в последующих запросах появляется возможность использования данной переменной. Переменная действует только в рамках одного сеанса соединения с сервером MySQL и прекращает свое существование после разрыва соединения.

Переменные также могут объявляться при помощи оператора *set*:



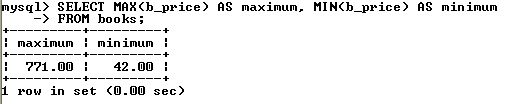
При использовании оператора *set* в качестве оператора присваивания может выступать обычный знак равенства =. Оператор *set* удобен тем, что он не возвращает результирующую таблицу. Не рекомендуется одновременно присваивать переменной некоторое значение и использовать эту переменную в одном запросе.

Переменная SQL позволяет сохранить одно промежуточное значение. Когда необходимо сохранить результирующую таблицу, прибегают к временным таблицам. Создание временных таблиц осуществляется при помощи оператора *CREATE temporary table,* синтаксискоторого ничем не отличается от синтаксиса оператора *CREATE table*.

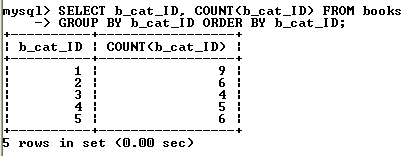
Временная таблица автоматически удаляется по завершении соединения с сервером, а ее имя действительно только в течение данного соединения. Это означает, что два разных клиента могут использовать временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем.

**Пример выполнения работы**

1. Создадим простой запрос на выборку к таблице *books*, который выводит максимальную и минимальную цены товарных позиций, присваивая им соответственно псевдонимы *maximum* и *minimum*:



2. Создадим простой запрос на выборку к таблице *books*, который выводит количество записей, соответствующих каждому из уникальных значений *b\_cat\_ID.* Для этого используем функцию *count( )* вместе с выражением *group by*:



**2. Создание сложных запросов на выборку**

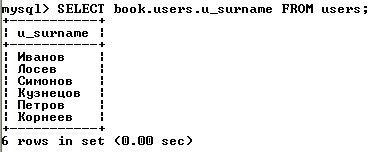
**Теоретические сведения**

Рассмотрим следующие вопросы:

* использование объединений в запросах к нескольким таблицам;
* создание вложенных запросов.

В реаль­ных приложениях часто требуется использовать сразу несколько таблиц БД. Запросы, которые обращаются одновременно к нескольким таблицам, называются многотабличными или сложными запросами.

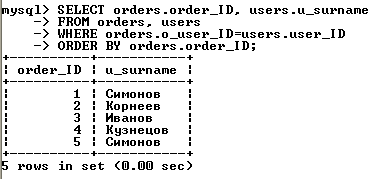
**Абсолютные ссылки на базы данных и таблицы**. В запросе мож­но прямо указывать необходимую БД и таблицу. Напри­мер, можно представить ссылку на столбец *u\_surname* из таблицы *users* в виде *users.u\_surname*. Аналогично можно уточнить БД, таблица из которой упоминается в запросе. Если необходимо, то вместе с БД и таблицей можно указать и столбец, например:



При использовании сложных запросов это позволяет избежать двусмысленности при указании источника необходи­мой информации.

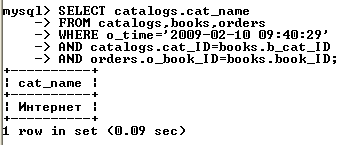
**Использование объединений для запросов к нескольким таблицам**. Хорошо спроектированная реляционная БД эффективна из-за связей между таблицами. При выборе информации из нескольких таблиц такие связи называют объединениями.

В качестве примера объединения двух таблиц рассмотрим запрос, извлекающий из БД *book* фамилии покупателей вместе с номерами сделанных ими заказов:

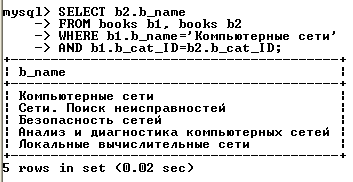


Выражение *WHERE* важно с точки зрения получения ре­зультата. Набор условий, используемых для объединения таб­лиц, называют условием объединения. В данном примере условие связы­вает таблицы *orders* и *users* по внешним ключам.

Объединение нескольких таблиц аналогично объединению двух таблиц. Например, необходимо выяснить, какому каталогу принадлежит товарная позиция из заказа, сделанного 10 февраля 2009 г. в 09:40:29:



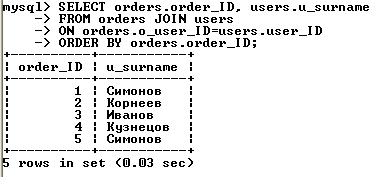
**Самообъединение таблиц**. Можно объединить таб­лицу саму с собой (когда интересуют связи между строками одной и той же таблицы). Пусть нужно выяснить, какие книги есть в каталоге, содержащем книгу с названием «Компьютерные сети». Для этого необходимо найти в таблице *books* номер каталога (*b\_cat\_ID*) с этой книгой, а затем посмотреть в таблице *books* книги этого каталога.



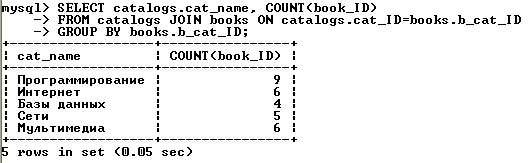
В этом запросе для таблицы *books* определены два разных псев­донима (две от­дельных таблицы *b1* и *b2*, которые должны содержать одни и те же данные). После этого они объединяются, как любые другие таблицы. Сна­чала ищется строка в таблице *b1*, а затем в таблице *b2* – строки с тем же значением номера каталога.

**Основное объединение**. Набор таблиц, перечисленных в выражении *FROM* и разделенных запятыми, – это декартово произведение (полное или перекрестное объединение), которое возвращает полный набор ком­бинаций. Добавление к нему условного выражения *WHERE* превраща­ет его в объединение по эквивалентности, ограни­чивающее число возвращаемых запросом строк.

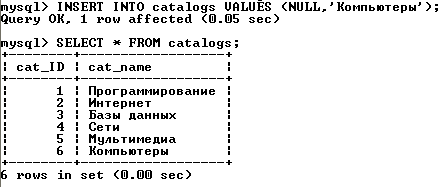
Вместо запятой в выражении *FROM* можно использовать ключевое слово *JOIN*. В этом случае вместо *WHERE* лучше использовать ключевое слово *ON*:



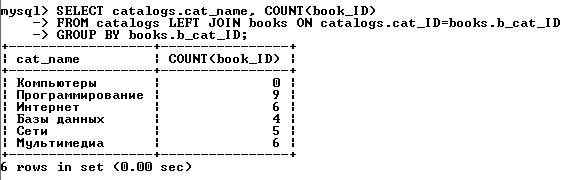
Вместо *JOIN* с тем же результатом можно использовать *CROSS JOIN* (перекрестное объединение) или *INNER JOIN* (внутреннее объединение). Пример запроса, выдающего число товарных позиций в каталогах:



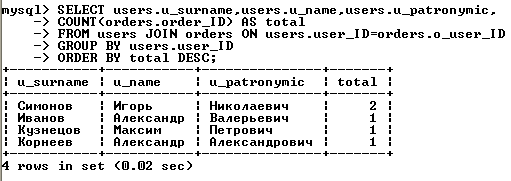
Допустим, происходит расширение ассортимента и в списке каталогов появляется новый каталог «Компьютеры»:



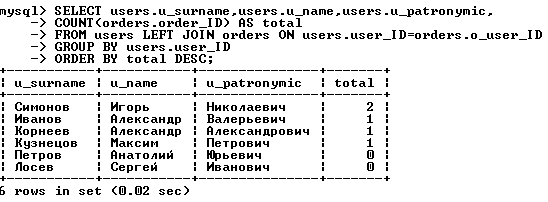
Предыдущий запрос не отразит наличие нового каталога (таблица *books* не содержит записей, относящихся к новому каталогу). Выходом является использование левого объединения (таблица *catalogs* должна быть левой таблицей):



Пусть нужно вывести список покупателей и число осуществленных ими покупок, причем покупателей необходимо отсортировать по убыванию числа заказов:

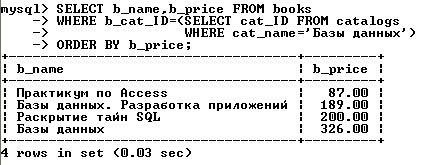


В список не входят покупатели, которые не сделали ни одной покупки. Чтобы вывести полный список покупателей, необходимо вместо перекрестного объединения таблиц *users* и *orders* использовать левое объединение (левой таблицей должна быть таблица *users*):



**Вложенный запрос**. Позволяет использовать результат, возвращаемый одним запросом, в другом запросе. Так как результат возвращает только оператор *select*, то в качестве вложенного запроса всегда выступает *SELECT*-запрос. В качестве внешнего запроса может выступать запрос с участием любого SQL-оператора: *select, insert, update, delete, create table* и др.

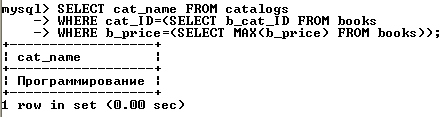
Пусть требуется вывести названия и цены товарных позиций из таблицы *books* для каталога «Базы данных» таблицы *catalogs*:



Получить аналогичный результат можно при помощи многотабличного запроса, но имеется ряд задач, которые решаются только при помощи вложенных запросов. Вложенный запрос может применяться не только с условием *WHERE*, но и в конструкциях *DISTINCT, GROUP BY, ORDER BY, LIMIT* и т. д. Различают:

* вложенные запросы, возвращающие одно значение;
* вложенные запросы, возвращающие несколько строк.

В первом случае вложенный запрос возвращает скалярное значение или литерал, которое используется во внешнем запросе (подставляет результат на место своего выполнения). Например, необходимо определить название каталога, содержащего самую дорогую товарную позицию:



Наиболее часто вложенные запросы используются в операциях сравнения в условиях, которые задаются ключевыми словами *WHERE, HAVING* или *ON*.

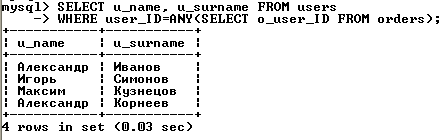
Однако следующий вложенный запрос вернет ошибку:



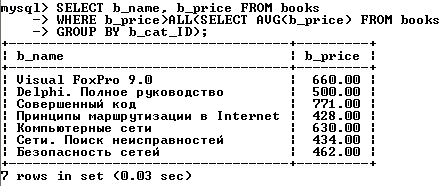
Чтобы выбрать строки из таблицы *catalogs*, у которых первичный ключ совпадает с одним из значений, возвращаемых вложенным запросом, следует воспользоваться конструкцией *IN*:



Ключевое слово *ANY* может применяться с использованием любого оператора сравнения. Используется логика *ИЛИ*, т. е. достаточно, чтобы срабатывало хотя бы одно из многих условий. Запрос вида *WHERE X > ANY (SELECT Y …)* можно интерпретировать как «где *X* больше хотя бы одного выбранного *Y*». Соответственно, запрос вида *WHERE X < ANY (SELECT Y …)* интерпретируется как «где *X* меньше хотя бы одного выбранного *Y*». Рассмотрим запрос, возвращающий имена и фамилии покупателей, совершивших хотя бы одну покупку:

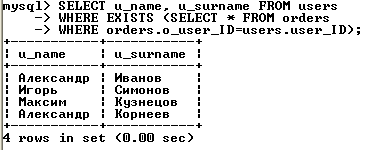


Ключевое слово *ALL* также может применяться с использованием любого оператора сравнения, но при этом используется логика *И*, то есть должны срабатывать все условия. Запрос вида *WHERE X > ALL (SELECT Y …)* интерпретируется как «где *X* больше любого выбранного *Y*». Соответственно, запрос вида *WHERE X < ALL (SELECT Y …)* интерпретируется как «где *X* меньше, чем все выбранные *Y*». Рассмотрим запрос, возвращающий все товарные позиции, цена которых превышает среднюю цену каждого из каталогов:



Результирующая таблица, возвращаемая вложенным запросом, может не содержать ни одной строки. Для проверки этого факта могут использоваться ключевые слова *EXISTS* и *NOT EXISTS*.

Запрос, формирующий список покупателей, совершивших хотя бы одну покупку, можно записать следующим образом:



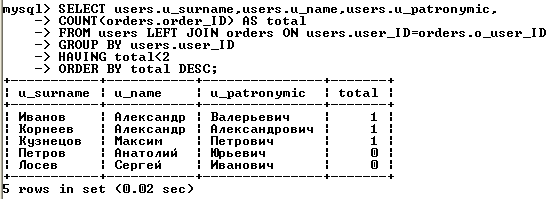
**Практическая работа**

При выполнении лабораторной работы необходимо:

* для заданной предметной области построить многотабличные запросы на выборку с использованием объединения;
* для заданной предметной области построить запросы на выборку, содержащие вложенные запросы;
* составить отчет по лабораторной работе.
* сложных многотабличных запросов должно быть не менее десяти

**Пример выполнения работы**

1. Создадим многотабличный запрос на выборку, который выводит фамилии, имена и отчества покупателей магазина, сделавших менее двух покупок:



2. Создадим запрос на выборку с вложенным запросом, выводящим перечень книг, которые не заказывались покупателями:

