СОДЕРЖАНИЕ

1 ЛАБОРАТОРНЫЕ РАБОТЫ

1.1 Задание к лабораторным работам

Выполнение лабораторных работ по курсу «Базы данных и базы знаний» включает следующие этапы:

1) проектирование структуры базы данных по выбранной предметной области;
2) реализация базы данных средствами СУБД MySQL.

При проектирование структуры базы данных необходимо выполнить следующие этапы проектирования:

- построить модель предметной области;
- выполнить проектирование логической модели данных и провести нормализацию полученных отношений;
- выполнить проектирования физической структуры и ее реализацию средствами СУБД.

1.2 План выполнения лабораторных работ

При выполнении лабораторных работ следует придерживаться следующего календарного плана-графика:

№	Тема лабораторной работы	Срок выполнения	Отчет по работе
1	Концептуальное проектирование данных	12.04.2020	Диаграмма сущность-связь
2	Проектирование логической структуры данных	12.04.2020	Набор отношений
3	Создание базы данных в среде MySQL. Изучение операторов выборки, вставки, удаления и изменения данных	17.04.2020	Скрипт создания базы данных MySQL и скрипты запросов к данным
4	Создание хранимых процедур, триггеров, представлений и настройка прав доступа	24.04.2020	Скрипт базы данных MySQL

1.3 Учебный пример разработки базы данных

Особенности диалекта SQL в СУБД MySQL рассмотрим на примере учебной базы данных book Интернет-магазина, торгующего компьютерной литературой. В базе данных должна поддерживаться следующая информация:

- тематические каталоги, по которым сгруппированы книги;
- предлагаемые книги (название, автор, год издания, цена, имеющееся на складе количество);

- зарегистрированные покупатели (имя, отчество, фамилия, телефон, адрес электронной почты, статус – авторизованный, неавторизованный, заблокированный, активный с хорошей кредитной историей);
- покупки, совершенные в магазине (время совершения покупки, число приобретенных экземпляров книги).

Логическая модель данных предметной области в стандарте IDEF1X представлена на рис. 1.

Выделены сущности КАТАЛОГ, КНИГА, КЛИЕНТ, ЗАКАЗ, между которыми установлены неидентифицирующие связи мощностью один-ко-многим, определенные спецификой предметной области.

Рис. 1. Логическая модель данных предметной области

Физическая модель данных предметной области в стандарте IDEF1X для целевой СУБД MySQL представлена на рис. 2.

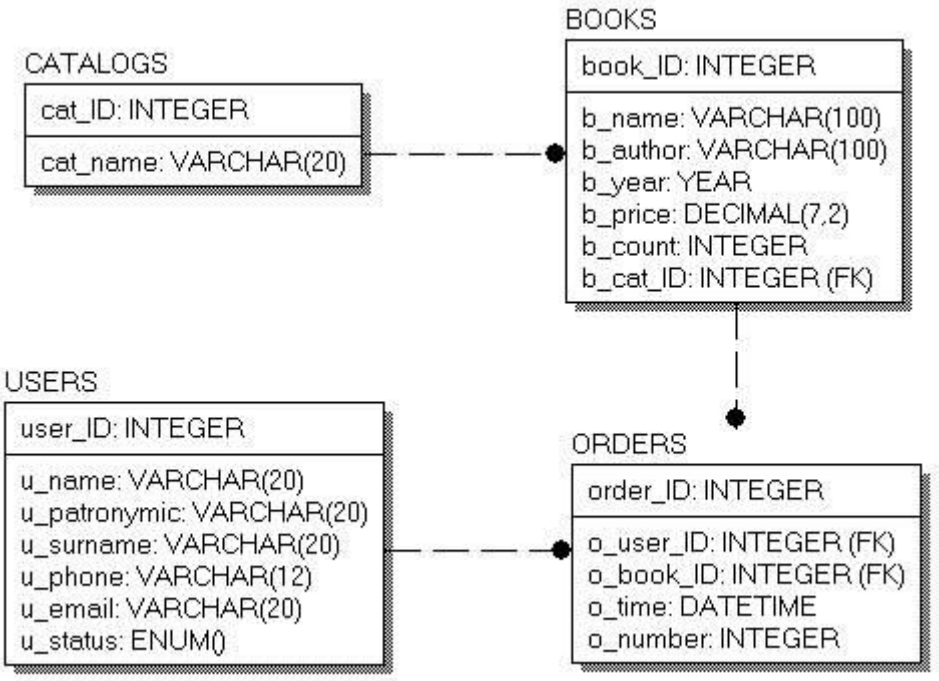

Рис. 2. Физическая модель предметной области

База данных book состоит из четырех таблиц:

- catalogs – список торговых каталогов;
- books – список предлагаемых книг;
- users – список зарегистрированных пользователей магазина;
- orders – список заказов (осуществленных сделок).

Таблица catalogs состоит из двух полей:

- cat_ID – уникальный код каталога;
- cat_name – имя каталога.

Оба поля должны быть снабжены атрибутом not null, поскольку неопределенное значение для них недопустимо.

Таблица books состоит из семи полей:

- book_ID – уникальный код книги;
- b_name – название книги;
- b_author – автор книги;
- b_year – год издания;
- b_price – цена книги;
- b_count – количество книг на складе;
- b_cat_ID – код каталога из таблицы catalogs.

Цена книги b_price и количество экземпляров на складе b_count могут иметь атрибут null. На момент доставки часто неизвестны количество товара и его цена, но отразить факт наличия товара в прайс-листе необходимо.

Поле b_cat_ID устанавливает связь между таблицами catalogs и books. Это поле должно быть объявлено как внешний ключ (FK)с правилом каскадного удаления и обновления. Обновление таблицы catalogs вызовет автоматическое обновление таблицы books. Удаление каталога в таблице catalogs приведет к автоматическому удалению всех записей в таблице books, соответствующих каталогу.

Таблица users состоит из семи полей:

- user_ID – уникальный код покупателя;
- u_name – имя покупателя;
- u_patronymic – отчество покупателя;
- u_surname – фамилия покупателя;
- u_phone – телефон покупателя (если имеется);
- u_email – e-mail покупателя (если имеется);
- u_status – статус покупателя.

Статус покупателя представлен полем типа enum, которое может принимать одно из четырех значений:

- active – авторизованный покупатель, который может осуществлять покупки через Интернет;
- passive – неавторизованный покупатель (значение по умолчанию), который осуществил процедуру регистрации, но не подтвердил ее и пока не может осуществлять покупки через Интернет, однако ему доступны каталоги для просмотра;
- lock – заблокированный покупатель, не может осуществлять покупки и просматривать каталоги магазина;
- gold – активный покупатель с хорошей кредитной историей, которому предоставляется скидка при следующих покупках в магазине.

Поля u_phone и u_email могут быть снабжены атрибутом null. Остальные поля должны получить атрибут not null.

Таблица orders включает пять полей:

- order_ID – уникальный номер сделки;
- o_user_ID – номер пользователя из таблицы users;
- o_book_ID – номер товарной позиции из таблицы books;
- o_time – время совершения сделки;
- o_number – число приобретенных товаров.

Поля таблицы orders должны быть снабжены атрибутом not null, т. к. при совершении покупки вся информация должна быть занесена в таблицу.

В таблице orders устанавливается связь с таблицами users (за счет поля o_user_id) и books (за счет поля o_book_id). Эти поля объявлены как внешние ключи (FK)с правилом каскадного удаления и обновления. Обновление таблиц users и books приведет к автоматическому обновлению таблицы orders. Удаление любого пользователя в таблице users приведет к автоматическому удалению всех записей в таблице orders, соответствующих этому пользователю.

2 МЕТОДИЧЕСКИЙ МАТЕРИАЛ

2.1 Лекция 1. Создание базы данных в среде MySQL

Рассмотрим следующие вопросы:

- создание и выбор базы данных;
- создание таблиц;
- столбцы и типы данных в MySQL;
- создание индексов;
- удаление таблиц, индексов и баз данных;
- изменение структуры таблиц.

Базы данных, таблицы и индексы легко создаются в рамках графического интерфейса MySQL, но мы будем использовать монитор MySQL (клиент командной строки), чтобы лучше понять структуру БД, таблиц и индексов.

2.1.1 Чувствительность к регистру и идентификаторы

Имена БД подчиняются тем же правилам зависимости от регистра символов, каким следуют каталоги операционной системы. Имена таблиц следуют тем же правилам, что и имена файлов. Все остальное не зависит от регистра.

Все идентификаторы, кроме имен псевдонимов, могут содержать до 64 символов. Имена псевдонимов могут иметь до 255 символов.

Идентификаторы могут содержать любые допустимые символы, но имена баз данных не могут содержать символы /, \ и . , а имена таблиц – символы . и /.

Зарезервированные слова можно использовать для идентификаторов, если заключить их в кавычки.

2.1.2 Комментарий в SQL

Начинается с двух дефисов (--), за которыми должен следовать пробел. Кроме того, MySQL содержит ряд собственных комментариев. Shell-комментарий # действует аналогично – все, что расположено правее его, является текстом комментария. С-комментарий /* */ является многострочным – комментарий начинается с /* и заканчивается, когда встретится завершение */.

2.1.3 Создание и выбор базы данных

Создание и выбор базы данных осуществляется с помощью оператора

```
create database имя_базы_данных;
```

Убедиться в том, что оператор выполнил задачу, можно с помощью оператора

```
show databases;
```

Теперь имеется пустая БД, ожидающая создания таблиц. Прежде чем работать с БД, необходимо выбрать эту БД с помощью оператора

```
use имя_базы_данных;
```

Теперь все действия по умолчанию будут применяться именно к этой БД.

2.1.4 Создание таблиц

Используется оператор CREATE TABLE, который в общем виде выглядит следующим образом:

```
create [temporary] table [if not exists]
имя_таблицы (определение таблицы)
[type=тип_таблицы];
```

Ключевое слово TEMPORARY используется для создания таблиц, которые будут существовать только в текущем сеансе работы с БД и будут автоматически удалены, когда сеанс завершится.

При использовании выражения IF NOT EXISTS таблица будет создана только в том случае, если еще нет таблицы с указанным именем.

Создать таблицу с такой же схемой, как у существующей, позволяет команда

```
create [temporary] table [if not exists]
имя_таблицы LIKE имя_старой_таблицы;
```

После имени таблицы в скобках объявляются имена столбцов, их типы и другая информация. В определение столбца можно добавить следующие описания.

- Объявить для любого столбца NOT NULL или NULL (столбцу запрещено или не запрещено содержать значения NULL). По умолчанию – NULL.
- Объявить для столбца значение по умолчанию, используя ключевое слово DEFAULT, за которым должно следовать значение по умолчанию.
- Использовать ключевое слово AUTO_INCREMENT, чтобы генерировать порядковый номер. Автоматически генерируемое значение будет на единицу большим, чем наибольшее значение в таблице. Первая введенная строка будет иметь порядковый номер 1. В таблице можно иметь не более одного столбца AUTO_INCREMENT, и он должен индексироваться.
- Объявить столбец первичным ключом таблицы с помощью выражения PRIMARY KEY.
- Объявить столбец внешним ключом, используя выражение FOREIGN KEY,с ссылкой на соответствующую таблицу с помощью выражения REFERENCES.
- Индексировать столбец с помощью слов INDEX или KEY (синонимы). Такие столбцы не обязательно должны содержать уникальные значения.
- Индексировать столбец с помощью слова UNIQUE, которое используется для указания того, что столбец должен содержать уникальные значения.
- Создать полнотекстовые индексы на основе столбцов типа TEXT, CHAR или VARCHAR с помощью слова FULLTEXT (только с таблицами MyISAM).

После закрывающей скобки можно указать тип таблицы:

- MyISAM – таблицы этого типа являются «родными» для MySQL, работают очень быстро и поддерживают полнотекстовую индексацию;
- InnoDB – ACID-совместимый механизм хранения, поддерживающий транзакции, внешние ключи, каскадное удаление и блокировки на уровне строк;
- BDB (Berkeley DB) – является механизмом хранения, который обеспечивает поддержку транзакций и блокировки на уровне страниц;

- MEMORY (HEAP) – таблицы целиком хранятся в оперативной памяти и никогда не записываются на диск, поэтому работают очень быстро, но ограничены в размерах и не допускают возможности восстановления в случае отказа системы;
- MERGE – тип позволяет объединить несколько таблиц MyISAM с одной структурой, чтобы к ним можно было направлять запросы как к одной таблице;
- NDB Cluster – тип предназначен для организации кластеровMySQL, когда таблицы распределены между несколькими компьютерами, объединенными в сеть;
- ARCHIVE – тип введен для хранения большого объема данных в сжатом формате; таблицы поддерживают только два SQL-оператора: INSERT и SELECT, причем оператор SELECT выполняется по методу полного сканирования таблицы;
- CSV – формат представляет собой обычный текстовый фал, записи в котором хранятся в строках, а поля разделены точкой с запятой (широко распространен в компьютерном мире, любая программа, поддерживающая CSV-формат, может открыть такой файл);
- FEDERATED – тип позволяет хранить данные в таблицах на другой машине сети (при создании таблицы в локальной директории создается только файл определения структуры таблицы, а все данные хранятся на удаленной машине).

MySQL поддерживает следующие типы данных, допустимые для столбцов:

- числовые;
- строковые;
- календарные;
- null – специальный тип, обозначающий отсутствие информации.

Числовые типы используются для хранения чисел и представляют два подтипа:

- точные числовые типы;
- приближенные числовые типы.

К точным числовым типам (табл. 1) относятся целый тип INTEGER и его вариации, а также вещественный тип decimal (синонимы numeric и dec). Последний используется для представления денежных данных.

Числовые типы могут характеризоваться максимальной длиной M. Для типа decimal параметр м задает число символов для отображения всего числа, а d – для его дробной части. Например: b_price DECIMAL (5, 2). Цифра 5 определяет общее число символов под число, а цифра 2 – количество знаков после запятой (интервал величин от –99.99 до 99.99). Можно не использовать параметры вообще, указать только общую длину или указать длину и число десятичных разрядов.

Объявления точных числовых типов можно завершать ключевыми словами UNSIGNED и (или) ZEROFILL. Ключевое слово UNSIGNED указывает, что столбец содержит только положительные числа или нули. Ключевое слово ZEROFILL означает, что число будет отображаться с ведущими нулями.

Таблица 1

ТИП	**ОБЪЕМ ПАМЯТИ**	**ДИАПАЗОН**
TINYINT (M) TINYINT unsigned	1 байт	от -128 до 127 (от -2^7 до 2^7-1) от 0 до 255 (от 0 до 2^8-1)

ТИП	ОБЪЕМ ПАМЯТИ	ДИАПАЗОН
SMALLINT (M) SMALLINT unsigned	2 байта	от -32 768 до 32 767 (от -2^{15} до 2^{15}-1) от 0 до 65 535 (от 0 до 2^{16}-1)
MEDIUMINT (M) MEDIUMINT unsigned	3 байта	от -8 388 608 до 8 388 607 (от -2^{23} до 2^{23}-1) от 0 до 16 777 215 (от 0 до 2^{24}-1)
INT (INTEGER) (M) INT unsigned	4 байта	от -2 147 683 648 до 2 147 683 647 (от -2^{31} до 2^{31}-1) от 0 до 4 294 967 295 (от 0 до 2^{32}-1)
BIGINT (M) BIGINT unsigned	8 байт	(от-2^{63} до2^{63}-1) (от 0 до 2^{64} -1)
BIT (M)	(M+7)/8 байт	От 1 до 64 битов, в зависимости от значения M
BOOL, BOOLEAN	1 байт	0 (false) либо 1 (true)
DECIMAL (M, D), NUMERUC (M, D)	M + 2 байта	Повышенная точность, зависит от параметров M и D

К приближенным числовым типам (табл. 2) относятся:

- FLOAT – представление чисел с плавающей запятой с обычной точностью;
- DOUBLE – представление чисел с плавающей запятой с двойной точностью.

Таблица 2

ТИП	ОБЪЕМ ПАМЯТИ	ДИАПАЗОН
FLOAT (M, D)	4 байта	Минимальное по модулю значение $1.175494351*10^{-39}$ Максимальное по модулю значение $3.402823466*10^{38}$
DOUBLE (M, D), REAL (M,D), DOUBLE PRECISION (M,D)	8 байт	Минимальное по модулю значение $2.2250738585072014*10^{-308}$ Максимальное по модулю значение $1.797693134862315*10^{308}$

Числовые типы с плавающей точкой также могут иметь параметр unsigned. Атрибут предотвращает хранение в столбце отрицательных величин, но максимальный интервал величин столбца остается прежним.

Приближенные числовые данные могут задаваться в обычной форме (например, 45.67) и в форме с плавающей точкой (например, 5.456E-02 или 4.674E+04).

Текстовые типы и строки (табл. 3):

- CHAR – хранение строк фиксированной длины;
- VARCHAR – хранение строк переменной длины;
- TEXT, BLOB и их вариации – хранение больших фрагментов текста;
- ENUM и SET – хранение значений из заданного списка.

Таблица 3

ТИП	ОБЪЕМ ПАМЯТИ	МАКСИМАЛЬНЫЙ РАЗМЕР
CHAR(M)	M символов	M символов
VARCHAR(M)	L+1 символов	M символов
TINYBLOB, TINYTEXT	L+1 символов	2^8-1 символов
BLOB, TEXT	L+2 символов	2^{16}-1 символов

MEDIUMBLOB, MEDIUMTEXT	L+3 символов	2^{24}-1 символов
LONGBLOB, LONGTEXT	L+4 символов	2^{32}-1 символов
ENUM('value 1', 'value2 ', ...)	1 или 2 байта	65 535 элементов
SET('value 1', 'value2', ...)	1, 2, 3, 4 или 8 байт	64 элемента

Здесь l – длина хранимой в ячейке строки, а приплюсованные к l байты – накладные расходы для хранения длины строки.

Для строк varchar требуется количество символов, равное длине строки плюс 1 байт, тогда как тип char(m), независимо от длины строки, использует для ее хранения все м символов. Тип char обрабатывается эффективнее переменных типов. Нельзя смешивать в таблице столбцы charи varchar. Если есть столбец переменной длины, все столбцы типа char будут приведены к типу varchar.

Типы blob и text аналогичны и отличаются в деталях. При выполнении операций над столбцами типа text учитывается кодировка, а типа blob – нет. Тип text используется для хранения больших объемов текста, тип blob – для больших двоичных объектов (электронные документы, изображения, звук). Основное отличие text от char и varchar – поддержка полнотекстового поиска.

Строки типов данных enum и set принимают значения из заданного списка. Значение типа enum должно содержать точно одно значение из указанного множества, тогда как столбцы set могут содержать любой или все элементы заданного множества одновременно. Для типа set (как и для enum) при объявлении задается список возможных значений, но ячейка может принимать любое значение из списка, а пустая строка означает, что ни один из элементов списка не выбран.

Типы enum и set задаются списком строк, но во внутреннем представлении элементы множеств сохраняются в виде чисел. Элементы типа enumнумеруются последовательно, начиная с 1. Под столбец может отводиться 1 байт (до 256 элементов в списке) или 2 байта (от 257 до 65536 элементов в списке). Элементы типа set обрабатываются как биты, размер типа определяется числом элементов в списке: 1 байт (от 1 до 8 элементов), 2 байта (от 9 до 16 элементов), 3 байта (от 17 до 24 элементов), 4 байта (от 25 до 32 элементов) и 8 байт (от 33 до 64 элементов).

Календарные типы данных (табл. 4):

- Date – для хранения даты (формат YYYY-MM-DD для дат вида 2009-10-15 и формат YY-MM-DD для дат вида 09-10-15);
- Time – для хранения времени суток (формат hh:mm:ss, где hh – часы, mm – минуты, ss – секунды, например, 10:48:56);
- Datetime– для представления и даты, и времени суток;
- Timestamp – если в соответствующем столбце строки не указать конкретное значение или NULL, там будет записано время, когда соответствующая строка была создана или в последний раз изменена (в формате DATETIME);
- Year – позволяет хранить только год.

Таблица 4

ТИП	**ОБЪЕМ ПАМЯТИ**	**ДИАПАЗОН**
DATE	3 байта	от '1000-01-01' до '9999-12-31'
TIME	3 байта	от '-828:59:59' до '828:59:59'
DATATIME	8 байт	от '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
TIMESTAMP (M)	4 байта	от '1970-01-01 00:00:00' до '2038-12-31 59:59:59'

YEAR(2) YEAR(4)	1 байт	формат YY, диапазон – от 1970 до 2069 формат YYYY, диапазон – от 1901 до 2155

Дни, месяцы, часы, минуты и секунды можно записывать как с ведущим нулем, так и без него. Например, все следующие записи идентичны:

```
'2009-04-06 02:04:08' '2009-4-06 02:04:08' '2009-4-6 02:04:08'
'2009-4-6 2:04:08' '2009-4-6 2:4:08' '2009-4-6 2:4:8'
```

В качестве разделителя между годами, месяцами, днями, часами, минутами, секундами может выступать любой символ, отличный от цифры. Так, следующие значения идентичны:

```
'09-12-31 11:30:45' '09.12.31 11+30+45' '09/12/31 11*30*45'
```

При указании времени после секунд через точку можно указать микросекунды, т. е. использовать расширенный формат вида hh:mm:ss.ffffff, например '10:25:14.000001'. Кроме того, можно использовать краткие форматы HH:MM и HH (вместо пропущенных величин будут подставлены нулевые значения).

Если время задается в недопустимом формате, то в поле записывается нулевое значение. Нулевое значение присваивается полям временного типа по умолчанию, когда им не присваивается инициирующее значение (табл. 5).

Таблица 5

ТИП	НУЛЕВОЕ ЗНАЧЕНИЕ
DATE	'0000-00-00'
TIME	'00:00:00'
DATATIME	'0000-00-00 00:00:00'
TIMESTAMP	
YEAR	

Формат timestamp совпадает с datetime, но во внутреннем представлении дата хранится как число секунд, прошедших с полуночи 1 января 1970 г. (такое исчисление принято в операционной системе UNIX, а дата 01.01.1970 считается началом эпохи UNIX и днем рождения операционной системы).

Если в таблице несколько столбцов timestamp, при модификации записи текущее время будет записываться только в один из них (первый). Можно явно указать столбец, которому необходимо назначать текущую дату при создании или изменении записи. Чтобы поля принимали текущую дату при создании записи, следует после определения столбца добавить default current_timestamp. Если текущее время должно выставляться при модификации записи, при использовании оператора update следует добавить on update current_timestamp.

Тип данных NULL используется, когда информации недостаточно и для части данных нельзя определить, какое значение они примут. Для указания того, что поле может принимать неопределенное значение, в определении столбца после типа данных следует указать ключевое слово null. Если поле не должно принимать значение null, следует указать ключевое слово not null.

2.1.5 Рекомендации по выбору типа данных

- Обработка числовых данных происходит быстрее строковых. Так как типы enum и set имеют внутреннее числовое представление, им следует отдавать предпочтение перед другими видами строковых данных, если это возможно.
- Производительность можно увеличить за счет представления строк в виде чисел. Пример – преобразование IP-адреса из строки в bigint. Это позволит уменьшить

размер таблицы и значительно увеличить скорость при сортировке и выборке данных, но потребует дополнительных преобразований.

- Базы данных хранятся на жестком диске, и чем меньше места они занимают, тем быстрее происходит поиск и извлечение. Если есть возможность, следует выбирать типы данных, занимающие меньше места.
- Типы фиксированной длины обрабатываются быстрее типов переменной длины, т. к. в последнем случае при частых удалениях и модификациях таблицы происходит ее фрагментация.
- Если применение столбцов с данными переменной длины неизбежно, для дефрагментации таблицы следует применять команду optimize table.

2.1.6 Обеспечение ссылочной целостности

Обеспечение ссылочной целостности задается конструкцией:

```
foreign key [name_key] (col1, ... ) REFERENCES tbl (tbl_col, ... )
[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}] [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]
```

Конструкция позволяет задать внешний ключ с необязательным именем name_key на столбцах, которые задаются в круглых скобках (один или несколько). Ключевое слово references указывает таблицу tbl, на которую ссылается внешний ключ, в круглых скобках указываются имена столбцов. Необязательные конструкции ON DELETE и ON UPDATE позволяют задать поведение СУБД при удалении и обновлении строк из таблицы-предка. Параметры, следующие за этими ключевыми словами, имеют следующие значения:

- CASCADE – при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, записи со ссылками на это значение в таблице-потомке удаляются или обновляются автоматически;
- SET NULL – при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, в таблице-потомке значения внешнего ключа, ссылающегося на таблицу-предка, устанавливаются в null;
- NO ACTION – при удалении или обновлении записей, содержащих первичный ключ, с таблицей-потомком никаких действий не производится;
- RESTRICT – если в таблице-потомке имеются записи, ссылающиеся на первичный ключ таблицы-предка, при удалении или обновлении записей с таким первичным ключом возвращается ошибка;
- SET DEFAULT – согласно стандарту SQL, при удалении или обновлении первичного ключа в таблице-потомке для ссылающихся на него записей в поле внешнего ключа должно устанавливаться значение по умолчанию (в MySQL это ключевое слово зарезервировано, но не обрабатывается).

2.1.7 Создание индексов

Индексы играют большую роль в БД, т. к. это основной способ ускорения их работы. Записи в таблице располагаются хаотически. Чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит много времени. Идея индексов состоит в том, чтобы создать для столбца копию, которая постоянно будет поддерживаться в отсортированном состоянии. Это позволяет быстро осуществлять поиск по такому столбцу.

Все необходимые индексы формируются при создании таблицы. Индексированы будут все столбцы, объявленные как PRIMARY KEY, KEY, UNIQUE или INDEX. Индекс также можно добавить с помощью оператора CREATE INDEX. Перед выполнением оператор преобразуется в оператор ALTER TABLE. Например, создание индекса с именем name на основе поля u_name из таблицы users:

```
create index name on users (u_name);
```

Перед ключевым словом index может присутствовать UNIQUE, требующее уникальности ограничения.

Корректность таблиц в БД можно проверить с помощью оператора

```
SHOW TABLES;
```

Более подробную информацию о структуре таблицы дает команда

```
DESCRIBE имя_таблицы;
```

2.1.8 Изменение БД

Специального оператора переименования БД нет, но можно переименовать каталог БД в системном каталоге (…\DATA).

Удаление БД. Удалить всю БД вместе с ее содержимым можно командой:

```
drop database [IF EXISTS] имя_базы_данных;
```

Удаление таблиц и индексов. Удалить таблицу можно с помощью оператора:

```
drop table [IF EXISTS] имя_таблицы;
```

Удалить индекс можно с помощью оператора:

```
drop index имя_индекса on имя_таблицы;
```

2.1.9 Изменение структуры таблиц

Изменить структуру существующей таблицы можно с помощью оператора ALTER TABLE. Например, можно создать индекс name для таблицы users следующим образом:

```
alter table users add index name (u_name);
```

Оператор ALTER TABLE является исключительно гибким, поэтому он имеет огромное множество дополнительных ключевых слов.

2.1.10 Контрольные вопросы

- Синтаксическая форма оператора создания и выбор базы данных
- Синтаксическая форма оператора создание таблиц
- Обеспечение ссылочной целостности в MySQL
- Назначение индексов. Создание индексов
- Синтаксическая форма оператора изменения базы данных
- Синтаксическая форма оператора изменения структуры таблиц

2.2 Лекция 2. Вставка, удаление и обновление данных

Рассмотрим следующие вопросы:

- вставка данных с помощью оператора INSERT;
- удаление данных операторами DELETE и TRUNCATE;

- обновление данных с помощью оператора UPDATE.

После создания БД и таблиц перед разработчиком встает задача заполнения таблиц данными. В реляционных БД традиционно применяют три подхода:

- однострочный оператор insert – добавляет в таблицу новую запись;
- многострочный оператор insert – добавляет в таблицу несколько записей;
- пакетная загрузка LOAD DATA INFILE – добавление данных из файла.

2.2.1 Вставка данных

Вставка данных с помощью оператора INSERT. Однострочный оператор insert может использоваться в нескольких формах. Упрощенный синтаксис первой формы:

```
insert [IGNORE] [INTO] имя_таблицы [(имя_столбца, ... )]
VALUES (выражение, ... );
```

Оператор вставляет новую запись в таблицу имя_таблицы. Значения полей записи перечисляются в списке (выражение, ...). Порядок следования столбцов задается списком (имя_столбца, ...). Список столбцов (имя_столбца, ...) позволяет менять порядок следования столбцов при добавлении.

Первичный ключ таблицы является уникальным, и попытка добавить уже существующее значение приведет к ошибке. Чтобы новые записи с дублирующим ключом отбрасывались без генерации ошибки, следует добавить после оператора insert ключевое слово IGNORE.

Другая форма оператора insert предполагает использование слова set:

```
insert [IGNORE] [INTO] имя_таблицы
SET имя_столбца1 = выражение1, имя_столбца2 = выражение2, ... ;
```

Оператор заносит в таблицу имя_таблицы новую запись, столбец имя_столбца в которой получает значение выражение.

Многострочный оператор INSERT совпадает по форме с однострочным оператором, но после ключевого слова values добавляется через запятую несколько списков (выражение, ...).

Практические примеры использования оператора insert для заполнения учебной БД book см. ниже, в пункте «Пример выполнения работы».

2.2.2 Удаление данных

Для удаления записей из таблиц предусмотрены:

- оператор DELETE;
- оператор TRUNCATE TABLE.

Оператор DELETE имеет следующий синтаксис:

```
DELETE FROM имя_таблицы
[Where условие]
[ORDER BY имя_поля]
[LIMIT число_строк];
```

Оператор удаляет из таблицы имя_таблицы записи, удовлетворяющие условию. В следующем примере из таблицы catalogs удаляются записи, имеющие значение первичного ключа catalog_id больше двух.

```
mysql> DELETE FROM catalogs WHERE cat_ID>2;
Query OK, 3 rows affected (0.05 sec)

mysql> SELECT * FROM catalogs;
+--------+------------------+
| cat_ID | cat_name         |
+--------+------------------+
|      1 | Программирование |
|      2 | Интернет         |
+--------+------------------+
2 rows in set (0.00 sec)
```

Если в операторе отсутствует условие where, удаляются все записи таблицы.

```
mysql> DELETE FROM catalogs;
Query OK, 2 rows affected (0.03 sec)

mysql> SELECT * FROM catalogs;
Empty set (0.00 sec)
```

Ограничение limit позволяет задать максимальное число записей, которые могут быть удалены. Следующий запрос удаляет все записи таблицы orders, но не более 3 записей.

```
mysql> DELETE FROM orders LIMIT 3;
Query OK, 3 rows affected (0.01 sec)

mysql> SELECT * FROM orders;
+----------+-----------+-----------+---------------------+----------+
| order_ID | o_user_ID | o_book_ID | o_time              | o_number |
+----------+-----------+-----------+---------------------+----------+
|        4 |         4 |        20 | 2009-03-10 18:20:00 |        1 |
|        5 |         3 |        20 | 2009-03-17 19:15:36 |        1 |
+----------+-----------+-----------+---------------------+----------+
2 rows in set (0.00 sec)
```

Конструкция order by обычно применяется вместе с ключевым словом limit. Например, если необходимо удалить 20 первых записей таблицы, то производится сортировка по полю типа datetime – тогда в первую очередь будут удалены самые старые записи.

Оператор truncate table полностью очищает таблицу и не допускает условного удаления. Он аналогичен оператору delete без условия where и ограничения limit. Удаление происходит гораздо быстрее, т. к. осуществляется не перебор записей, а полное очищение таблицы.

```
mysql> TRUNCATE TABLE orders;
Query OK, 5 rows affected (0.03 sec)

mysql> SELECT * FROM orders;
Empty set (0.00 sec)
```

2.2.3 Обновление данных

Обновление данных (изменение значений полей в существующих записях) обеспечивают:

- оператор Update;

- оператор Replace.

Оператор UPDATE позволяет обновлять отдельные поля в существующих записях. Имеет следующий синтаксис

```
Update [IGNORE] имя_таблицы
SET имя_столбца1= выражение1 [, имя_столбца2 = выражение2 … ]
[WHERE условие]
[ORDER BY имя_поля ]
[LIMIT число_строк] ;
```

После ключевого слова update указывается таблица, которая изменяется. В предложении set указывается, какие столбцы обновляются и устанавливаются их новые значения. Необязательное условие WHERE позволяет задать критерий отбора строк (обновляться будут только строки, удовлетворяющие условию).

Если указывается необязательное ключевое слово ignore, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, породившие конфликтные ситуации, обновлены не будут.

Запрос, изменяющий в таблице catalogs «Сети» на «Компьютерные сети».

```
mysql> UPDATE catalogs SET cat_name='Компьютерные сети'
    -> WHERE cat_name='Сети';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM catalogs;
+--------+--------------------+
| cat_ID | cat_name           |
+--------+--------------------+
|      1 | Программирование   |
|      2 | Интернет           |
|      3 | Базы данных        |
|      4 | Компьютерные сети  |
|      5 | Мультимедиа        |
+--------+--------------------+
5 rows in set (0.00 sec)
```

Обновлять можно всю таблицу. Пусть требуется уменьшить на 5 % цену на все книги. Для этого следует старую цену в рублях умножить на 0,95.

```
mysql> UPDATE books SET b_price=b_price*0.95;
Query OK, 30 rows affected (0.03 sec)
Rows matched: 30  Changed: 30  Warnings: 0

mysql> SELECT book_ID, b_name, b_price FROM books;
+---------+-----------------------------------------+---------+
| book_ID | b_name                                  | b_price |
+---------+-----------------------------------------+---------+
|       1 | JavaScript в кармане                    |   39.90 |
|       2 | Visual FoxPro 9.0                       |  627.00 |
|       3 | C++ Как он есть                         |  207.10 |
|       4 | Создание приложений с помощью C#        |  160.55 |
|       5 | Delphi. Народные советы                 |  230.85 |
|       6 | Delphi. Полное руководство              |  475.00 |
|       7 | Профессиональное программирование на PHP |  293.55 |
|       8 | Совершенный код                         |  732.45 |
|       9 | Практика программирования               |  203.30 |
|      10 | Принципы маршрутизации в Internet       |  406.60 |
|      11 | Поиск в Internet                        |  101.65 |
|      12 | Web-конструирование                     |  168.15 |
|      13 | Самоучитель Интернет                    |  114.95 |
|      14 | Популярные интернет-браузеры            |   77.90 |
|      15 | Общение в Интернете                     |   80.75 |
|      16 | Базы данных                             |  309.70 |
|      17 | Базы данных. Разработка приложений      |  179.55 |
|      18 | Раскрытие тайн SQL                      |  190.00 |
|      19 | Практикум по Access                     |   82.65 |
|      20 | Компьютерные сети                       |  598.50 |
|      21 | Сети. Поиск неисправностей              |  412.30 |
|      22 | Безопасность сетей                      |  438.90 |
|      23 | Анализ и диагностика компьютерных сетей |  326.80 |
|      24 | Локальные вычислительные сети           |   77.90 |
|      25 | Цифровая фотография                     |  141.55 |
|      26 | Музыкальный компьютер для гитариста     |  206.15 |
|      27 | Видео на ПК                             |  219.45 |
|      28 | Мультипликация во Flash                 |  200.45 |
|      29 | Запись CD и DVD                         |  158.65 |
|      30 | Запись и обработка звука на компьютере  |   48.45 |
+---------+-----------------------------------------+---------+
30 rows in set (0.00 sec)
```

Инструкции limit и order by позволяют ограничить число изменяемых записей. При этом за один запрос можно обновить несколько столбцов таблицы. Например, необходимо в таблице books для десяти самых дешевых товарных позиций уменьшить количество книг на складе на единицу, а цену – на 5 %.

```
mysql> UPDATE books SET b_price=b_price*0.95,b_count=b_count-1
    -> ORDER BY b_price LIMIT 10;
Query OK, 10 rows affected (0.05 sec)
Rows matched: 10  Changed: 10  Warnings: 0

mysql> SELECT book_ID, b_name, b_price, b_count FROM books;
+---------+-----------------------------------------+---------+---------+
| book_ID | b_name                                  | b_price | b_count |
+---------+-----------------------------------------+---------+---------+
|       1 | JavaScript в кармане                    |   39.90 |       9 |
|       2 | Visual FoxPro 9.0                       |  660.00 |       2 |
|       3 | C++ Как он есть                         |  218.00 |       4 |
|       4 | Создание приложений с помощью C#        |  169.00 |       1 |
|       5 | Delphi. Народные советы                 |  243.00 |       6 |
|       6 | Delphi. Полное руководство              |  500.00 |       6 |
|       7 | Профессиональное программирование на PHP |  309.00 |       5 |
|       8 | Совершенный код                         |  771.00 |       1 |
|       9 | Практика программирования               |  214.00 |      12 |
|      10 | Принципы маршрутизации в Internet       |  428.00 |       4 |
|      11 | Поиск в Internet                        |  101.65 |       1 |
|      12 | Web-конструирование                     |  177.00 |       6 |
|      13 | Самоучитель Интернет                    |  114.95 |       3 |
|      14 | Популярные интернет-браузеры            |   77.90 |       5 |
|      15 | Общение в Интернете                     |   80.75 |       4 |
|      16 | Базы данных                             |  326.00 |       2 |
|      17 | Базы данных. Разработка приложений      |  189.00 |       6 |
|      18 | Раскрытие тайн SQL                      |  200.00 |       3 |
|      19 | Практикум по Access                     |   82.65 |       5 |
|      20 | Компьютерные сети                       |  630.00 |       6 |
|      21 | Сети. Поиск неисправностей              |  434.00 |       4 |
|      22 | Безопасность сетей                      |  462.00 |       5 |
|      23 | Анализ и диагностика компьютерных сетей |  344.00 |       3 |
|      24 | Локальные вычислительные сети           |   77.90 |       7 |
|      25 | Цифровая фотография                     |  141.55 |      19 |
|      26 | Музыкальный компьютер для гитариста     |  217.00 |      15 |
|      27 | Видео на ПК                             |  231.00 |      10 |
|      28 | Мультипликация во Flash                 |  211.00 |      20 |
|      29 | Запись CD и DVD                         |  158.65 |      11 |
|      30 | Запись и обработка звука на компьютере  |   48.45 |       7 |
+---------+-----------------------------------------+---------+---------+
30 rows in set (0.00 sec)
```

Оператор REPLACE работает как оператор insert, за исключением того, что старая запись с тем же значением индекса unique или primary keyперед внесением новой будет удалена. Если не используются индексы unique или primary key, то применение оператора replace не имеет смысла.

Синтаксис оператора REPLACE аналогичен синтаксису оператора insert:

```
REPLACE [INTO] имя_таблицы [(имя_столбца, ... )]
VALUES (выражение, ... )
```

В таблицу вставляются значения, определяемые в списке после ключевого слова VALUES. Задать порядок столбцов можно при помощи необязательного списка, следующего за именем таблицы. Как и оператор Insert, оператор replace допускает многострочный формат.

2.2.4 Оператор SELECT

При выполнении лабораторной работы необходимо для заданной предметной области средствами MySQL:

- заполнить согласованными данными таблицы БД;
- при необходимости исправить введенную информацию.

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 *.

```
mysql> SELECT * FROM orders;
+----------+-----------+-----------+---------------------+----------+
| order_ID | o_user_ID | o_book_ID | o_time              | o_number |
+----------+-----------+-----------+---------------------+----------+
|        1 |         3 |         8 | 2009-01-04 10:39:38 |        1 |
|        2 |         6 |        10 | 2009-02-10 09:40:29 |        2 |
|        3 |         1 |        20 | 2009-02-18 13:41:05 |        4 |
|        4 |         4 |        20 | 2009-03-10 18:20:00 |        1 |
|        5 |         3 |        20 | 2009-03-17 19:15:36 |        1 |
+----------+-----------+-----------+---------------------+----------+
5 rows in set (0.02 sec)
```

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

```
mysql> SELECT cat_name, cat_ID FROM catalogs;
+------------------+--------+
| cat_name         | cat_ID |
+------------------+--------+
| Программирование |      1 |
| Интернет         |      2 |
| Базы данных      |      3 |
| Сети             |      4 |
| Мультимедиа      |      5 |
+------------------+--------+
5 rows in set (0.01 sec)
```

2.2.5 Условия выборки

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

```
mysql> SELECT user_ID, u_surname FROM users
    -> WHERE u_status='active';
+---------+-----------+
| user_ID | u_surname |
+---------+-----------+
|       1 | Иванов    |
|       3 | Симонов   |
|       4 | Кузнецов  |
+---------+-----------+
3 rows in set (0.03 sec)
```

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

```
mysql> SELECT DISTINCT u_status FROM users;
+----------+
| u_status |
+----------+
| active   |
| passive  |
| lock     |
| gold     |
+----------+
4 rows in set (0.01 sec)
```

2.2.6 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:

```
mysql> SELECT * FROM orders ORDER BY o_user_ID;
+----------+-----------+-----------+---------------------+----------+
| order_ID | o_user_ID | o_book_ID | o_time              | o_number |
+----------+-----------+-----------+---------------------+----------+
|        3 |         1 |        20 | 2009-02-18 13:41:05 |        4 |
|        1 |         3 |         8 | 2009-01-04 10:39:38 |        1 |
|        5 |         3 |        20 | 2009-03-17 19:15:36 |        1 |
|        4 |         4 |        20 | 2009-03-10 18:20:00 |        1 |
|        2 |         6 |        10 | 2009-02-10 09:40:29 |        2 |
+----------+-----------+-----------+---------------------+----------+
5 rows in set (0.00 sec)
```

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:

```
mysql> SELECT o_time FROM orders ORDER BY o_time DESC;
+---------------------+
| o_time              |
+---------------------+
| 2009-03-17 19:15:36 |
| 2009-03-10 18:20:00 |
| 2009-02-18 13:41:05 |
| 2009-02-10 09:40:29 |
| 2009-01-04 10:39:38 |
+---------------------+
5 rows in set (0.27 sec)
```

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

2.2.7 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:

```
mysql> SELECT book_ID, b_count FROM books
    -> ORDER BY b_count DESC
    -> LIMIT 5;
+---------+---------+
| book_ID | b_count |
+---------+---------+
|      28 |      20 |
|      25 |      20 |
|      26 |      15 |
|      29 |      12 |
|       9 |      12 |
+---------+---------+
5 rows in set (0.03 sec)
```

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:

```
mysql> SELECT book_ID, b_count FROM books
    -> ORDER BY b_count DESC
    -> LIMIT 5,5;
+---------+---------+
| book_ID | b_count |
+---------+---------+
|       1 |      10 |
|      27 |      10 |
|      24 |       8 |
|      30 |       8 |
|      20 |       6 |
+---------+---------+
5 rows in set (0.00 sec)
```

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

Группировка записей. Конструкция GROUP BY 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:

```
mysql> SELECT COUNT(book_ID) FROM books;
+----------------+
| COUNT(book_ID) |
+----------------+
|             30 |
+----------------+
1 row in set (0.16 sec)
```

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:

```
mysql> SELECT COUNT(DISTINCT b_cat_ID) FROM books;
+--------------------------+
| COUNT(DISTINCT b_cat_ID) |
+--------------------------+
|                        5 |
+--------------------------+
1 row in set (0.02 sec)
```

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

```
mysql> SELECT COUNT(order_ID) AS total FROM orders;
+-------+
| total |
+-------+
|     5 |
+-------+
1 row in set (0.05 sec)
```

Использование функций в конструкции where приведет к ошибке. В следующем примере показана попытка извлечения из таблицы catalogsзаписи с максимальным значением поля cat_ID:

```
mysql> SELECT * FROM catalogs WHERE cat_ID=MAX(cat_ID);
ERROR 1111 (HY000): Invalid use of group function
```

Решение задачи следует искать в использовании конструкции order by:

```
mysql> SELECT * FROM catalogs ORDER BY cat_ID DESC LIMIT 1;
+--------+-------------+
| cat_ID | cat_name    |
+--------+-------------+
|      5 | Мультимедиа |
+--------+-------------+
1 row in set (0.00 sec)
```

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

```
mysql> SELECT b_cat_ID FROM books
    -> GROUP BY b_cat_ID ORDER BY b_cat_ID;
+----------+
| b_cat_ID |
+----------+
|        1 |
|        2 |
|        3 |
|        4 |
|        5 |
+----------+
5 rows in set (0.03 sec)
```

При использовании group by возможно использование условия where:

```
mysql> SELECT b_cat_ID, COUNT(b_cat_ID) FROM books
    -> WHERE b_cat_ID > 2
    -> GROUP BY b_cat_ID
    -> ORDER BY b_cat_ID;
+----------+-----------------+
| b_cat_ID | COUNT(b_cat_ID) |
+----------+-----------------+
|        3 |               4 |
|        4 |               5 |
|        5 |               6 |
+----------+-----------------+
3 rows in set (0.02 sec)
```

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:

```
mysql> SELECT b_cat_ID, COUNT(b_cat_ID) AS total FROM books
    -> GROUP BY b_cat_ID
    -> HAVING total > 5
    -> ORDER BY b_cat_ID;
+----------+-------+
| b_cat_ID | total |
+----------+-------+
|        1 |     9 |
|        2 |     6 |
|        5 |     6 |
+----------+-------+
3 rows in set (0.00 sec)
```

Запрос, извлекающий уникальные значения столбца b_cat_ID, большие двух:

```
mysql> SELECT b_cat_ID, COUNT(b_cat_ID) FROM books
    -> GROUP BY b_cat_ID
    -> HAVING b_cat_ID > 2
    -> ORDER BY b_cat_ID;
+----------+-----------------+
| b_cat_ID | COUNT(b_cat_ID) |
+----------+-----------------+
|        3 |               4 |
|        4 |               5 |
|        5 |               6 |
+----------+-----------------+
3 rows in set (0.00 sec)
```

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.

2.2.8 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:

```
mysql> SELECT VERSION();
+-----------------------+
| VERSION()             |
+-----------------------+
| 5.0.51b-community-nt  |
+-----------------------+
1 row in set (0.00 sec)
```

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

```
mysql> SELECT 2+3;
+-----+
| 2+3 |
+-----+
|   5 |
+-----+
1 row in set (0.00 sec)
```

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.

2.2.9 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.

2.2.10 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.

2.2.11 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:

```
mysql> SELECT NULL=NULL;
+-----------+
| NULL=NULL |
+-----------+
|      NULL |
+-----------+
1 row in set (0.02 sec)
```

Корректнее использовать следующий запрос:

```
mysql> SELECT NULL IS NULL;
+--------------+
| NULL IS NULL |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)
```

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.

2.2.12 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.

2.2.13 Создание простых запросов на выборку

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

2.2.14 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 selectc использованием оператора присваивания := .

Например:

```
mysql> SELECT @total := COUNT(*) FROM books;
+---------------------+
| @total := COUNT(*) |
+---------------------+
|                  30 |
+---------------------+
1 row in set (0.42 sec)

mysql> SELECT @total;
+--------+
| @total |
+--------+
| 30     |
+--------+
1 row in set (0.02 sec)
```

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:

```
mysql> SET @last=CURDATE()-INTERVAL 7 DAY;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT CURDATE(), @last;
+------------+------------+
| CURDATE()  | @last      |
+------------+------------+
| 2009-12-22 | 2009-12-15 |
+------------+------------+
1 row in set (0.00 sec)
```

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.

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.

2.2.15 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:

```
mysql> SELECT book.users.u_surname FROM users;
+-----------+
| u_surname |
+-----------+
| Иванов    |
| Лосев     |
| Симонов   |
| Кузнецов  |
| Петров    |
| Корнеев   |
+-----------+
6 rows in set (0.00 sec)
```

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:

```
mysql> SELECT orders.order_ID, users.u_surname
    -> FROM orders, users
    -> WHERE orders.o_user_ID=users.user_ID
    -> ORDER BY orders.order_ID;
+----------+-----------+
| order_ID | u_surname |
+----------+-----------+
|        1 | Симонов   |
|        2 | Корнеев   |
|        3 | Иванов    |
|        4 | Кузнецов  |
|        5 | Симонов   |
+----------+-----------+
5 rows in set (0.00 sec)
```

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:

```
mysql> SELECT catalogs.cat_name
    -> FROM catalogs,books,orders
    -> WHERE o_time='2009-02-10 09:40:29'
    -> AND catalogs.cat_ID=books.b_cat_ID
    -> AND orders.o_book_ID=books.book_ID;
+----------+
| cat_name |
+----------+
| Интернет |
+----------+
1 row in set (0.09 sec)
```

2.2.16 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.

```
mysql> SELECT b2.b_name
    -> FROM books b1, books b2
    -> WHERE b1.b_name='Компьютерные сети'
    -> AND b1.b_cat_ID=b2.b_cat_ID;
+----------------------------------------+
| b_name                                 |
+----------------------------------------+
| Компьютерные сети                      |
| Сети. Поиск неисправностей             |
| Безопасность сетей                     |
| Анализ и диагностика компьютерных сетей |
| Локальные вычислительные сети          |
+----------------------------------------+
5 rows in set (0.02 sec)
```

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.

2.2.17 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:

```
mysql> SELECT orders.order_ID, users.u_surname
    -> FROM orders JOIN users
    -> ON orders.o_user_ID=users.user_ID
    -> ORDER BY orders.order_ID;
+----------+-----------+
| order_ID | u_surname |
+----------+-----------+
|        1 | Симонов   |
|        2 | Корнеев   |
|        3 | Иванов    |
|        4 | Кузнецов  |
|        5 | Симонов   |
+----------+-----------+
5 rows in set (0.03 sec)
```

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:

```
mysql> SELECT catalogs.cat_name, COUNT(book_ID)
    -> FROM catalogs JOIN books ON catalogs.cat_ID=books.b_cat_ID
    -> GROUP BY books.b_cat_ID;
+------------------+----------------+
| cat_name         | COUNT(book_ID) |
+------------------+----------------+
| Программирование |              9 |
| Интернет         |              6 |
| Базы данных      |              4 |
| Сети             |              5 |
| Мультимедиа      |              6 |
+------------------+----------------+
5 rows in set (0.05 sec)
```

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

```
mysql> INSERT INTO catalogs VALUES (NULL,'Компьютеры');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM catalogs;
+--------+------------------+
| cat_ID | cat_name         |
+--------+------------------+
|      1 | Программирование |
|      2 | Интернет         |
|      3 | Базы данных      |
|      4 | Сети             |
|      5 | Мультимедиа      |
|      6 | Компьютеры       |
+--------+------------------+
6 rows in set (0.00 sec)
```

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):

```
mysql> SELECT catalogs.cat_name, COUNT(book_ID)
    -> FROM catalogs LEFT JOIN books ON catalogs.cat_ID=books.b_cat_ID
    -> GROUP BY books.b_cat_ID;
+------------------+----------------+
| cat_name         | COUNT(book_ID) |
+------------------+----------------+
| Компьютеры       |              0 |
| Программирование |              9 |
| Интернет         |              6 |
| Базы данных      |              4 |
| Сети             |              5 |
| Мультимедиа      |              6 |
+------------------+----------------+
6 rows in set (0.00 sec)
```

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

```
mysql> SELECT users.u_surname,users.u_name,users.u_patronymic,
    -> COUNT(orders.order_ID) AS total
    -> FROM users JOIN orders ON users.user_ID=orders.o_user_ID
    -> GROUP BY users.user_ID
    -> ORDER BY total DESC;
+-----------+-----------+---------------+-------+
| u_surname | u_name    | u_patronymic  | total |
+-----------+-----------+---------------+-------+
| Симонов   | Игорь     | Николаевич    |     2 |
| Иванов    | Александр | Валерьевич    |     1 |
| Кузнецов  | Максим    | Петрович      |     1 |
| Корнеев   | Александр | Александрович |     1 |
+-----------+-----------+---------------+-------+
4 rows in set (0.02 sec)
```

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):

```
mysql> SELECT users.u_surname,users.u_name,users.u_patronymic,
    -> COUNT(orders.order_ID) AS total
    -> FROM users LEFT JOIN orders ON users.user_ID=orders.o_user_ID
    -> GROUP BY users.user_ID
    -> ORDER BY total DESC;
+-----------+-----------+---------------+-------+
| u_surname | u_name    | u_patronymic  | total |
+-----------+-----------+---------------+-------+
| Симонов   | Игорь     | Николаевич    |     2 |
| Иванов    | Александр | Валерьевич    |     1 |
| Корнеев   | Александр | Александрович |     1 |
| Кузнецов  | Максим    | Петрович      |     1 |
| Петров    | Анатолий  | Юрьевич       |     0 |
| Лосев     | Сергей    | Иванович      |     0 |
+-----------+-----------+---------------+-------+
6 rows in set (0.02 sec)
```

2.2.18 Вложенный запрос

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:

```
mysql> SELECT b_name,b_price FROM books
    -> WHERE b_cat_ID=(SELECT cat_ID FROM catalogs
    ->                 WHERE cat_name='Базы данных')
    -> ORDER BY b_price;
+-----------------------------------+---------+
| b_name                            | b_price |
+-----------------------------------+---------+
| Практикум по Access               |   87.00 |
| Базы данных. Разработка приложений |  189.00 |
| Раскрытие тайн SQL                |  200.00 |
| Базы данных                       |  326.00 |
+-----------------------------------+---------+
4 rows in set (0.03 sec)
```

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:

```
mysql> SELECT cat_name FROM catalogs
    -> WHERE cat_ID=(SELECT b_cat_ID FROM books
    -> WHERE b_price=(SELECT MAX(b_price) FROM books));
+-----------------+
| cat_name        |
+-----------------+
| Программирование |
+-----------------+
1 row in set (0.00 sec)
```

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

Однако следующий вложенный запрос вернет ошибку:

```
mysql> SELECT cat_name FROM catalogs
    -> WHERE cat_ID=(SELECT b_cat_ID FROM books);
ERROR 1242 (21000): Subquery returns more than 1 row
```

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

```
mysql> SELECT cat_name FROM catalogs
    -> WHERE cat_ID IN (SELECT b_cat_ID FROM books GROUP BY b_cat_ID);
+-----------------+
| cat_name        |
+-----------------+
| Программирование |
| Интернет        |
| Базы данных     |
| Сети            |
| Мультимедиа     |
+-----------------+
5 rows in set (0.17 sec)
```

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:

```
mysql> SELECT u_name, u_surname FROM users
    -> WHERE user_ID=ANY(SELECT o_user_ID FROM orders);
+----------+-----------+
| u_name   | u_surname |
+----------+-----------+
| Александр | Иванов   |
| Игорь    | Симонов   |
| Максим   | Кузнецов  |
| Александр | Корнеев  |
+----------+-----------+
4 rows in set (0.03 sec)
```

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:

```
mysql> SELECT b_name, b_price FROM books
    -> WHERE b_price>ALL(SELECT AVG(b_price) FROM books
    -> GROUP BY b_cat_ID);
+-----------------------------------+---------+
| b_name                            | b_price |
+-----------------------------------+---------+
| Visual FoxPro 9.0                 |  660.00 |
| Delphi. Полное руководство        |  500.00 |
| Совершенный код                   |  771.00 |
| Принципы маршрутизации в Internet |  428.00 |
| Компьютерные сети                 |  630.00 |
| Сети. Поиск неисправностей        |  434.00 |
| Безопасность сетей                |  462.00 |
+-----------------------------------+---------+
7 rows in set (0.03 sec)
```

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:

```
mysql> SELECT u_name, u_surname FROM users
    -> WHERE EXISTS (SELECT * FROM orders
    -> WHERE orders.o_user_ID=users.user_ID);
+----------+-----------+
| u_name   | u_surname |
+----------+-----------+
| Александр | Иванов   |
| Игорь    | Симонов   |
| Максим   | Кузнецов  |
| Александр | Корнеев  |
+----------+-----------+
4 rows in set (0.00 sec)
```

2.2.19 Контрольные вопросы

- Вставка данных
- Удаление данных
- Обновление данных
- Оператор SELECT

- Условия выборки в операторе SELECT
- Сортировка данных в операторе SELECT
- Ограничение объема выборки
- Использование функций, операторов и арифметических операций
- Создание простых запросов на выборку
- Назначение временных таблиц
- Использование абсолютных ссылок на базы данных и таблицы
- Самообъединение таблиц и основное объединение
- Вложенные запросы.

2.2.20 Задание к лабораторной работе 3

На этапе проектирования физической структуры необходимо для заданной предметной области средствами СУБД:

- создать базу данных;
- создать таблицы, определить поля таблиц, индексы;
- определить связи между таблицами и ограничения целостности;
- построить не менее двух простых запросов на выборку с использованием операторов и функций;
- построить многотабличный запрос на выборку с использованием объединения;
- построить запрос на выборку, содержащий вложенный запрос;
- оформить отчет по лабораторной работе.

2.3 Лекция 3. Создание хранимых процедур

На практике часто требуется повторять последовательность одинаковых запросов. Хранимые процедуры позволяют объединить последовательность таких запросов и сохранить их на сервере. После этого клиентам достаточно послать один запрос на выполнение хранимой процедуры.

Хранимые процедуры обладают следующими преимуществами.

- Повторное использование кода – после создания хранимой процедуры ее можно вызывать из любых приложений и SQL-запросов.
- Сокращение сетевого трафика – вместо нескольких запросов экономнее послать серверу запрос на выполнение хранимой процедуры и сразу получить ответ.
- Безопасность – действия не приведут к нарушению целостности данных, т.к. для выполнения хранимой процедуры пользователь должен иметь привилегию.
- Простота доступа – хранимые процедуры позволяют инкапсулировать сложный код и оформить его в виде простого вызова.
- Выполнение бизнес-логики – хранимые процедуры позволяют перенести код сохранения целостности БД из прикладной программы на сервер БД. Бизнес-логика в виде хранимых процедур не зависит от языка разработки приложения.

2.3.1 Создание хранимых процедур

Реализуется оператором

```
CREATE PROCEDURE имя_процедуры ( [ параметр [, ... ] ] )
[характеристика ...] тело_процедуры
```

В скобках передается необязательный список параметров, перечисленных через запятую. Каждый параметр позволяет передать в процедуру (из процедуры) входные данные (результат работы) и имеет следующий синтаксис:

```
[ IN | OUT | INOUT ] имя_параметра тип
```

Ключевые слова in, out, inout задают направление передачи данных:

- in – данные передаются строго внутрь хранимой процедуры; если параметру с данным модификатором присваивается новое значение, при выходе из процедуры оно не сохраняется и параметр принимает значение, которое он имел до вызова;
- out – данные передаются строго из хранимой процедуры, если параметр имеет какое-то начальное значение, то внутри хранимой процедуры это значение во внимание не принимается;
- inout – значение этого параметра как принимается во внимание внутри процедуры, так и сохраняет свое значение при выходе из нее.

Список аргументов, заключенных в круглые скобки, присутствует всегда. Если аргументы отсутствуют, следует использовать пустой список. Если ни один из модификаторов не указан, считается, что параметр объявлен с ключевым словом in.

Телом процедуры является составной оператор begin ... end, внутри которого могут располагаться другие операторы:

```
[ label: ] BEGIN
statements
END [ label ]
```

Оператор, начинающийся с необязательной метки label (любое уникальное имя), может заканчиваться выражением end label. Внутри составного оператора begin ... end может находиться другой составной оператор. Если хранимая процедура содержит один запрос, то составной оператор можно не использовать.

При работе с хранимыми процедурами символ точки с запятой в конце запроса воспринимается консольным клиентом как сигнал к отправке запроса на сервер. Поэтому следует переопределить разделитель запросов – например, вместо точки с запятой использовать последовательность // :

```
mysql> DELIMITER //
mysql> SELECT VERSION( )//
+-----------------------+
| VERSION( )            |
+-----------------------+
| 5.0.51b-community-nt |
+-----------------------+
1 row in set (0.00 sec)
```

Пример создания простейшей хранимой процедуры:

```
mysql> CREATE PROCEDURE my_version()
    -> BEGIN
    -> SELECT VERSION();
    -> END //
Query OK, 0 rows affected (0.00 sec)
```

Чтобы вызвать хранимую процедуру, необходимо применить оператор call, после которого помещается имя процедуры и ее параметры в круглых скобках:

```
mysql> CALL my_version()//
+----------------------+
| VERSION()            |
+----------------------+
| 5.0.51b-community-nt |
+----------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Рекомендуется избегать использования названий хранимых процедур, совпадающих с именами встроенных функций MySQL. В теле хранимой процедуры можно использовать многострочный комментарий, который начинается с последовательности /* и заканчивается последовательностью */ .

Рассмотрим хранимые процедуры с параметрами. Создадим и вызовем процедуру, которая присваивает пользовательской переменной @x новое значение:

```
mysql> CREATE PROCEDURE set_x(IN value INT)
    -> BEGIN
    -> SET @x = value;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> CALL set_x(123456)//
Query OK, 0 rows affected (0.00 sec)
```

Через параметр value процедуре передается числовое значение 123456, которое она присваивает пользовательской переменной @x. Модификатор in сообщает, что данные передаются внутрь функции. Проверим корректность работы процедуры:

```
mysql> SELECT @x//
+--------+
| @x     |
+--------+
| 123456 |
+--------+
1 row in set (0.00 sec)
```

В отличие от пользовательской переменной @x, которая является глобальной и доступна как внутри хранимой процедуры set_x ( ), так и вне ее, параметры процедуры являются локальными и доступны для использования только внутри нее.

Создадим процедуру numcatalogs(), которая подсчитывает число записей в таблице catalogs базы данных book:

```
mysql> CREATE PROCEDURE numcatalogs(OUT total INT)
    -> BEGIN
    -> SELECT COUNT(*) INTO total FROM catalogs;
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL numcatalogs(@a)//
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT @a//
+------+
| @a   |
+------+
| 5    |
+------+
1 row in set (0.00 sec)
```

Хранимая процедура numcatalogs() имеет один целочисленный параметр total, в котором сохраняется число записей в таблице catalogs. Осуществляется это при помощи оператора select ... into ... from. В качестве параметра функции numcatalogs() передается пользовательская переменная @a.

Создадим хранимую процедуру catalogname(), которая будет возвращать по первичному ключу catID название каталога cat_name. Для этого потребуется определить параметр id с атрибутом in, и catalog с атрибутом OUT.

```
mysql> CREATE PROCEDURE catalogname(IN id INT, OUT catalog TINYTEXT)
    -> BEGIN
    -> SELECT cat_name INTO catalog FROM catalogs
    -> WHERE catID = id;
    -> END //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SET @id = 5//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL catalogname(@id, @name)//
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @id, @name//
+------+-------------+
| @id  | @name       |
+------+-------------+
| 5    | Мультимедиа |
+------+-------------+
1 row in set (0.00 sec)
```

2.3.2 Операторы управления потоком данных

Хранимые процедуры позволяют реализовать сложную логику с помощью операторов ветвления и циклов. Вне хранимых процедур эти операторы применять нельзя. Ветвление программы по условию позволяет реализовать оператор:

```
IF лог_выражение THEN оператор
[ELSEIF лог_выражение THEN оператор] ...
[ELSE оператор]
END IF ;
```

Логическое выражение может принимать два значения:

- 0 (ложь);
- значение, отличное от нуля (истина).

Если логическое выражение истинно, то выполняется оператор в блоке THEN, иначе выполняется список операторов в блоке else (если блок else имеется). В логических выражениях можно использовать операторы сравнения (= , >, >=, <> , <, <=). Логические выражения можно комбинировать с помощью операторов && (И), а также || (ИЛИ). Если в блоках if, elseif и else – два или более операторов, необходимо использовать составной оператор begin ... end.

Множественный выбор позволяет осуществить оператор:

```
CASE выражение
WHEN значение THEN оператор
[WHEN значение THEN оператор] ...
[ELSE оператор]
```

```
END CASE ;
```

2.3.3 Выражение сравнивается со значениями

Как только найдено соответствие, выполняется соответствующий оператор. Если соответствия не найдены, выполняется оператор, размещенный после ключевого слова else (если оно присутствует).

В MySQL имеется несколько операторов, позволяющих реализовать циклы. Первый оператор цикла имеет следующий синтаксис:

```
[ label: ] WHILE условие DO
операторы
END WHILE [ label ] ;
```

Операторы выполняются в цикле, пока истинно условие. При каждой итерации условие проверяется, и если при очередной проверке оно будет ложным (0), цикл завершится. Если условие ложно с самого начала, то цикл не выполнится ни разу. Если в цикле выполняется более одного оператора, не обязательно заключать их в блок begin ... end, т. к. эту функцию выполняет сам оператор while.

Досрочный выход из цикла обеспечивает оператор:

```
LEAVE label ;
```

Оператор прекращает выполнение блока, помеченного меткой label (например, прекращает выполнение цикла по достижении критического числа итераций).

Досрочное прекращение цикла также обеспечивает оператор

```
ITERATE label ;
```

В отличие от оператора leave оператор iterate не прекращает выполнение цикла, он лишь выполняет досрочное прекращение текущей итерации. Оператор leave эквивалентен оператору BREAK, а оператор iterate эквивалентен оператору continue в С-подобных языках программирования.

Второй оператор цикла имеет следующий вид:

```
[ label: ] REPEAT
операторы UNTIL условие END REPEAT [label] ;
```

Условие проверяется не в начале, а в конце оператора цикла. Таким образом, цикл выполняет по крайней мере одну итерацию независимо от условия. Следует отметить, что цикл выполняется, пока условие ложно. Оператор цикла может быть снабжен необязательной меткой label, по которой можно осуществлять досрочный выход из цикла при помощи операторов leave и iterate.

Реализовать бесконечный цикл позволяет оператор

```
[ label: ] LOOP
операторы END LOOP [ label ] ;
```

Цикл loop (в отличие от операторов while и repeat) не имеет условий выхода. Поэтому данный цикл должен обязательно иметь в составе оператор leave.

Осуществлять безусловный переход позволяет оператор

```
GOTO label ;
```

Оператор осуществляет переход к оператору, помеченному меткой label. Это может быть как оператор begin, так и любой из циклов: while, repeat и loop. Кроме того, метка может быть не привязана ни к одному из операторов, а объявлена при помощи оператора

```
LABEL label ;
```

Использовать оператор goto для реализации циклов не рекомендуется, т. к. обычные циклы гораздо нагляднее и проще поддаются модификации, в них сложнее допустить логическую ошибку.

2.3.4 Удаление хранимых процедур

Для удаления процедур используется оператор

```
DROP PROCEDURE [IF EXISTS] имя_процедуры ;
```

Если удаляемой процедуры с таким именем не существует, оператор возвращает ошибку, которую можно подавить, если использовать необязательное ключевое слово if exists.

2.3.5 Обработчики ошибок

При выполнении хранимых процедур могут возникать ошибки. MySQL позволяет каждой возникающей в хранимой процедуре ошибке назначить свой обработчик, который в зависимости от ситуации и серьезности ошибки может как прекратить, так и продолжить выполнение процедуры.

Для объявления такого обработчика предназначен оператор

```
DECLARE тип_обработчика HANDLER FOR код_ошибки [, ... ] выражение;
```

Выражение содержит SQL-запрос, который выполняется при срабатывании обработчика. Тип обработчика может принимать одно из трех значений:

- continue – выполнение текущей операции продолжается после выполнения оператора обработчика;
- exit – выполнение составного оператора begin ... end, в котором объявлен обработчик, прекращается;
- undo – данный вид обработчика в текущей версии не поддерживается.

Обработчик может быть привязан сразу к нескольким ошибкам, для этого их коды следует перечислить через запятую. Код ошибки, для которой будет происходить срабатывание обработчика, может принимать следующие значения:

- sqlstate [value] значение – значение sqlstate является пятисимвольным кодом ошибки в шестнадцатеричном формате (стандарт в SQL); примеры кодов – 'hy000', 'hy001', '42000' и т. д.; один код обозначает сразу несколько ошибок;
- sqlwarning – любое предупреждение MySQL; это ключевое слово позволяет назначить обработчик для всех предупреждений; обрабатываются любые события, для которых код sqlstate начинается с 01;
- not found – любая ошибка, связанная с невозможностью найти объект (таблицу, процедуру, функцию, столбец и т. п.); обрабатываются любые события, для которых код sqlstate начинается с 02;
- sqlexception – ошибки, не охваченные ключевыми словами sqlwarning и not found;
- mysql_error_code – обычные четырехзначные ошибки MySQL, такие как 1020, 1232, 1324 и т. п.;
- именованное условие (см. ниже).

При указании кода ошибки можно использовать не только целочисленные коды, но и именованные условия, которые объявляются при помощи оператора

```
DECLARE именованное условие CONDITION FOR код ошибки;
```

Оператор объявляет именованное условие для кода ошибки. Так, для обрабатываемой ошибки 1062 (23000) – дублирование уникального индекса, оператор может выглядеть следующим образом:

```
DECLARE 'violation' CONDITION FOR SQLSTATE '23000';
DECLARE 'violation' CONDITION FOR 1062;
```

Первое объявление охватывает все ошибки со статусом 23000, второй вид ошибок более узкий и включает только дублирование уникального индекса.

2.3.6 Курсоры

Если результирующий запрос возвращает одну запись, поместить результаты в промежуточные переменные можно с помощью оператора select ... into ... from. Однако результирующие таблицы чаще содержат несколько записей, и использование такой конструкции приводит к возникновению ошибки 1172: «Результат содержит более чем одну строку».

Избежать ошибки можно, добавив предложение limit 1 или назначив continue-обработчик ошибок. Однако такая процедура реализует не то поведение, которое ожидает пользователь. Кроме того, существуют ситуации, когда требуется обработать именно многострочную результирующую таблицу.

Например, пусть требуется вернуть записи одной таблицы, отвечающие определенному условию, и на основании этих записей создать новую таблицу. Решить эту задачу можно с помощью курсоров, которые позволяют в цикле просмотреть каждую строку результирующей таблицы запросов. Работа с курсорами похожа на работу с файлами – сначала открытие курсора, затем чтение и после закрытие.

Работа с курсорами происходит по следующему алгоритму:

1. При помощи инструкции DECLARE курсор CURSOR FOR связывается имя курсора с выполняемым запросом.

2. Оператор open выполняет запрос, связанный с курсором, и устанавливает курсор перед первой записью результирующей таблицы.

3. Оператор fetch помещает курсор на первую запись результирующей таблицы и извлекает данные из записи в локальные переменные хранимой процедуры. Повторный вызов оператора fetch приводит к перемещению курсора к следующей записи, и так до тех пор, пока записи в результирующей таблице не будут исчерпаны. Эту операцию удобно осуществлять в цикле.

4. Оператор close прекращает доступ к результирующей таблице и ликвидирует связь между курсором и результирующей таблицей.

2.3.7 Пример

1. Создадим хранимую процедуру, которая выводит число заказов покупателя по вводимому в качестве параметра процедуры коду покупателя.

```
mysql> CREATE PROCEDURE num(OUT total INT, IN user_kod INT)
    -> BEGIN
    -> SELECT COUNT(*) INTO total FROM orders WHERE o_user_ID=user_kod;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)
```

Параметр total является выходным, его значение равно числу заказов покупателя, код которого записывается во входной параметр user_kod.Процедура считает все строки, где код клиента совпадает с параметром user_kod.

До вызова процедуры присваиваем параметру процедуры значение кода клиента. Затем вызываем процедуру оператором CALL. Для вывода результата можно воспользоваться оператором SELECT.

```
mysql> SET @user_kod=3//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL num(@total,@user_kod)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @total,@user_kod//
+--------+-----------+
| @total | @user_kod |
+--------+-----------+
| 2      | 3         |
+--------+-----------+
1 row in set (0.00 sec)
```

2. Создадим хранимую процедуру, которая записывает в новую таблицу fevral все заказы, сделанные в феврале 2009 г. Предварительно необходимо создать новую пустую таблицу fevral со структурой, аналогичной структуре таблицы orders.

```
mysql> CREATE TABLE fevral(
    ->   f_order_ID int(6) NOT NULL,
    ->   f_o_user_ID int NOT NULL,
    ->   f_o_book_ID int NOT NULL,
    ->   f_o_time datetime NOT NULL,
    ->   f_o_number int(6) NOT NULL,
    ->   PRIMARY KEY (f_order_ID)
    -> )TYPE=InnoDB//
Query OK, 0 rows affected, 1 warning (0.08 sec)
```

Хранимая процедура ord_fevr ( ) использует курсор curf, который в цикле читает данные из таблицы orders и добавляет их в таблицу fevral.

```
mysql> CREATE PROCEDURE ord_fevr()
    -> BEGIN
    -> DECLARE id int;
    -> DECLARE _end int DEFAULT 0;
    -> DECLARE userID int;
    -> DECLARE bookID int;
    -> DECLARE tim datetime;
    -> DECLARE num int;
    -> DECLARE curf CURSOR FOR SELECT * FROM orders
    -> WHERE o_time BETWEEN '2009.02.01' AND '2009.02.28';
    -> DECLARE CONTINUE HANDLER FOR NOT FOUND SET _end=1;
    -> OPEN curf;
    -> wet: LOOP
    -> FETCH curf INTO id,userID,bookID,tim,num;
    -> IF _end THEN LEAVE wet;
    -> END IF;
    -> INSERT INTO fevral VALUES(id,userID,bookID,tim,num);
    -> END LOOP wet;
    -> CLOSE curf;
    -> END
    -> //
Query OK, 0 rows affected (0.03 sec)
```

Вызов процедуры осуществляется оператором call. Для просмотра результата выполнения процедуры используем полную выборку из таблицыfevral.

```
mysql> CALL ord_fevr//
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM fevral//
+------------+-------------+-------------+---------------------+------------+
| f_order_ID | f_o_user_ID | f_o_book_ID | f_o_time            | f_o_number |
+------------+-------------+-------------+---------------------+------------+
|          2 |           6 |          10 | 2009-02-10 09:40:29 |          2 |
|          3 |           1 |          20 | 2009-02-18 13:41:05 |          4 |
+------------+-------------+-------------+---------------------+------------+
2 rows in set (0.00 sec)
```

2.3.8 Контрольные вопросы

- Создание хранимых процедур
- Операторы управления потоком данных
- Удаление хранимых процедур
- Обработчики ошибок
- Курсоры и последовательный доступ к данным.

2.4 Лекция 4. Создание триггеров

Рассмотрим следующие вопросы:

- понятие триггера;
- создание триггеров с помощью оператора CREATE trigger;
- удаление триггеров с помощью оператора DROP trigger.

Триггер – эта та же хранимая процедура, но привязанная к событию изменения содержимого конкретной таблицы.

Возможны три события, связанных с изменением содержимого таблицы, к которым можно привязать триггер:

- · insert – вставка новых данных в таблицу;
- · delete – удаление данных из таблицы;
- · Update – обновление данных в таблице.

Например, при оформлении нового заказа, т. е. при добавлении новой записи в таблицу orders, можно создать триггер, автоматически вычитающий число заказанных товарных позиций в таблице books.

2.4.1 Создание триггеров

Создать новый триггер позволяет оператор:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt ;
```

Оператор создает триггер с именем trigger_name, привязанный к таблице tbl_name. Не допускается привязка триггера к временной таблице или представлению. Конструкция trigger_time указывает момент выполнения триггера и может принимать два значения:

- before – действия триггера производятся до выполнения операции изменения таблицы;
- after – действия триггера производятся после выполнения операции изменения таблицы.

Конструкция trigger_event показывает, на какое событие должен реагировать триггер, и может принимать три значения:

- · insert – триггер привязан к событию вставки новой записи в таблицу;
- · update – триггер привязан к событию обновления записи таблицы;
- · delete – триггер привязан к событию удаления записей таблицы.

Для таблицы tbl_name может быть создан только один триггер для каждого из событий trigger_event и момента trigger_time. Таким образом, для каждой из таблиц может быть создано всего шесть триггеров.

Конструкция trigger_stmt представляет тело триггера – оператор, который необходимо выполнить при возникновении события trigger_event в таблице tbl_name.

Если требуется выполнить несколько операторов, то необходимо использовать составной оператор begin ... end. Синтаксис и допустимые операторы такие же, как и у хранимых процедур. Внутри составного оператора begin ... end допускаются все специфичные для хранимых процедур операторы и конструкции:

- · другие составные операторы begin ... end;
- · операторы управления потоком (if, case, while, loop, repeat, leave, iterate);
- · объявления локальных переменных при помощи оператора declare и назначение им значений при помощи оператора set;
- · именованные условия и обработчики ошибок.

В MySQL триггеры нельзя привязать к каскадному обновлению или удалению записей из таблицы типы InnoDB по связи первичный ключ/внешний ключ.

Триггеры сложно использовать, не имея доступа к новым записям, которые вставляются в таблицу, или старым записям, которые обновляются или удаляются. Для доступа к новым и старым записям используются префиксы new и old соответственно. Если в таблице обновляется поле total, то получить доступ к старому значению можно по имени old.total, а к новому – new.total.

Пример простейшего триггера для учебной БД book см. в пункте «Пример выполнения работы» (пример 1). Он демонстрирует работу триггеров после добавления записи в таблицу без вмешательства в запрос. Рассмотрим триггер, который будет включаться до вставки новых записей в таблицу orders и ограничивает число заказываемых товаров до 1:

```
mysql> CREATE TRIGGER restrict_count BEFORE INSERT ON orders
    -> FOR EACH ROW
    -> BEGIN
    -> SET NEW.o_number=1;
    -> END//
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> INSERT INTO orders VALUES (NULL,1,2,NOW(),10)//
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM orders WHERE orderID = LAST_INSERT_ID()//
+---------+----------+----------+---------------------+----------+
| orderID | o_userID | o_bookID | o_time              | o_number |
+---------+----------+----------+---------------------+----------+
|      16 |        1 |        2 | 2009-10-23 20:26:19 |        1 |
+---------+----------+----------+---------------------+----------+
1 row in set (0.00 sec)
```

Часто при обновлении полей таблицы производится попытка добавления некорректных значений. Пример триггера, который при добавлении нового покупателя преобразует полные

имена и отчества в инициалы, см. в пункте «Пример выполнения работы» (пример 2). Он привязан к событию INSERT. Чтобы имя и отчество не могло быть отредактировано при помощи оператора update, можно создать триггер, привязанный к событию update.

2.4.2 Удаление триггеров

Удалить существующий триггер позволяет оператор

```
DROP TRIGGER trigger_name;
```

2.4.3 Контрольные вопросы

- Создание триггеров
- Удаление триггеров

2.5 Лекция 5. Представления

Рассмотрим следующие вопросы:

- создание представлений с помощью оператора CREATE VIEW;
- удаление представлений с помощью оператора DROP VIEW.

Основная структурная единица реляционных БД – таблицы, но язык SQL предоставляет еще один способ организации данных. Представление – это запрос на выборку, которому присваивается уникальное имя и который можно сохранять или удалять из БД как хранимую процедуру. Представления позволяют увидеть результаты сохраненного запроса так, как будто это полноценная таблица. MySQL, встретив в запросе ссылку на представление, ищет его определение в БД. Пользовательский запрос с участием представления преобразуется в эквивалентный запрос к исходным таблицам. Если определение представления простое, то каждая строка представления формируется «на лету». Если определение сложное, MySQL материализует представление в виде временной таблицы. Клиент, обращаясь к представлению, будет видеть только столбцы результирующей таблицы. Не имеет значения, сколько столбцов в исходной таблице и является ли запрос, лежащий в основе представления, одно- или многотабличным. Клиенту можно запретить обращаться к исходным таблицам, но снабдить привилегиями обращения к представлениям. На одном наборе таблиц можно создать гибкие системы доступа.

Преимущества представлений:

- · безопасность – каждый пользователь имеет доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать;
- · простота запросов – с помощью представления можно извлечь данные из нескольких таблиц и представить их как одну таблицу (запрос ко многим таблицам заменяется однотабличным запросом к представлению);
- · простота структуры – представления позволяют создать для каждого пользователя собственную структуру БД (отображаются данные, которые ему нужны);
- · защита от изменений – таблицы и их структура могут постоянно изменяться и переименовываться; представления позволяют создавать виртуальные таблицы со старыми именами и структурой, позволяя избежать модификации приложений.

Недостатки представлений:

- · производительность – представления создают видимость существования таблицы, и MySQL приходится преобразовывать запрос к представлению в запрос к исходным

таблицам; если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением;

- · ограничение на обновление – когда пользователь пытается обновить строки представления, MySQL необходимо обновить строки в исходных таблицах; это возможно только для простых представлений, сложные представления доступны только для выборки.

Поэтому не стоит везде применять представления вместо исходных таблиц.

2.5.1 Создание представлений

Создание представлений осуществляется при помощи оператора

```
CRFATE [OR REPLACE] [ALGORITHM = {UNDEFINED / MERGE / TEMPTABLE}]
VIEW view_name [(column_list)] AS select_statement
[WITH [CASCADED / LOCAL] CHECK OPTION];
```

Оператор создает представление view_name со столбцами, перечисленными в column_list, на основании запроса select_statement. Рассмотрим создание представления cat, которое дублирует таблицу catalogs базы данных book:

```
mysql> CREATE VIEW cat AS  SELECT  *  FROM catalogs;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT  *  FROM cat;
+-------+------------------+
| catID | cat_name         |
+-------+------------------+
|     1 | Программирование |
|     2 | Интернет         |
|     3 | Базы данных      |
|     4 | Сети             |
|     5 | Мультимедиа      |
+-------+------------------+
5 rows in set (0.00 sec)
```

Представление рассматривается как полноценная таблица и может быть просмотрено в списке таблиц БД при помощи оператора show tables:

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_book |
+---------------+
| books          |
| cat            |
| catalogs       |
| orders         |
| users          |
+---------------+
5 rows in set (0.00 sec)
```

При создании представления можно явно указать список столбцов, изменить их названия и порядок следования, например:

```
mysql> CREATE OR REPLACE VIEW cat (catalog, id)
    -> AS SELECT cat_name, catID FROM catalogs;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT * FROM cat;
+------------------+----+
| catalog          | id |
+------------------+----+
| Программирование |  1 |
| Интернет         |  2 |
| Базы данных      |  3 |
| Сети             |  4 |
| Мультимедиа      |  5 |
+------------------+----+
5 rows in set (0.00 sec)
```

Представления, не содержащие дополнительных столбцов, называются вертикальными представлениями. Они применяются для ограничения доступа пользователей к столбцам таблицы. Пример вертикального представления см. ниже.

Кроме вертикальных представлений используются горизонтальные представления, которые делают видимыми только те строки, с которыми работают пользователи. Например, чтобы в электронном магазине каждый менеджер видел только те товарные позиции, за которые отвечает, можно создать представления для менеджеров. Учетные записи менеджеров следует лишить привилегии доступа к таблице и разрешить просматривать только свои представления.

Создадим представление manager1 для менеджера, работающего с каталогами «Интернет» и «Сети»:

```
mysql> CREATE VIEW manager1
    -> AS SELECT * FROM books
    -> WHERE b_catID IN (SELECT catID
    -> FROM catalogs
    -> WHERE cat_name = 'Интернет' OR cat_name = 'Сети')
    -> ORDER BY b_name;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT b_name, b_price, b_count FROM manager1;
+----------------------------------------+---------+---------+
| b_name                                 | b_price | b_count |
+----------------------------------------+---------+---------+
| Web-конструирование                    |  177.00 |       6 |
| Анализ и диагностика компьютерных сетей|  344.00 |       3 |
| Безопасность сетей                     |  462.00 |       5 |
| Компьютерные сети                      |  630.00 |       6 |
| Общение в Интернете                    |   85.00 |       5 |
| Принципы маршрутизации в Internet      |  428.00 |       4 |
| Поиск в Internet                       |  107.00 |       2 |
| Популярные интернет-браузеры           |   82.00 |       6 |
| Локальные вычислительные сети          |   82.00 |       8 |
| Сети. Поиск неисправностей             |  434.00 |       4 |
| Самоучитель Интернет                   |  121.00 |       4 |
+----------------------------------------+---------+---------+
11 rows in set (0.05 sec)
```

Наиболее удобно использовать представления для формирования сгруппированных таблиц. При работе с такими таблицами MySQL самостоятельно формирует временную таблицу.

2.5.2 Удаление представлений

Удаление представлений выполняется с помощью оператора:

```
DROP VIEW [IF EXISTS] view_name [, view_name] … ;
```

Оператор позволяет уничтожить одно или несколько представлений, например:

```
mysql> DROP VIEW cat, list_user, price;
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW TABLES;
+---------------+
| Tables_in_book |
+---------------+
| books          |
| catalogs       |
| orders         |
| users          |
+---------------+
4 rows in set (0.00 sec)
```

2.5.3 Контрольные вопросы

- Создание представлений
- Удаление представлений

2.6 Лекция 6. Права доступа

СУБД MySQL является многопользовательской средой, поэтому для доступа к таблицам БД могут быть созданы различные учетные записи с разным уровнем привилегий. Учетной записи редактора можно предоставить привилегии на просмотр таблицы, добавление новых записей и обновление уже существующих. Администратору БД можно предоставить более широкие полномочия (возможность создания таблиц, редактирования и удаления уже существующих). Для пользователя БД достаточно лишь просмотра таблиц.

Рассмотрим следующие вопросы:

- создание, редактирование и удаление учетных записей пользователей;
- назначение и отмена привилегий.

Учетная запись является составной и принимает форму 'username' @ 'host', где username – имя пользователя, а host – наименование хоста, с которого пользователь может обращаться к серверу. Например, записи 'root' @ '127.0.0.1' и 'wet' @ '62.78.56.34' означают, что пользователь с именем root может обращаться с хоста, на котором расположен сервер, а wet – только с хоста с IP-адресом 62.78.56.34.

IP-адрес 127.0.0.1 всегда относится к локальному хосту. Если сервер и клиент установлены на одном хосте, то сервер слушает соединения по этому адресу, а клиент отправляет на него SQL-запросы.

IP-адрес 127.0.0.1 имеет псевдоним localhost, поэтому учетные записи вида 'root' @ '127.0.0.1' можно записывать в виде 'root' @ 'localhost'.

Число адресов, с которых необходимо обеспечить доступ пользователю, может быть значительным. Для задания диапазона в имени хоста используется специальный символ "%". Так, учетная запись 'wet' @ '%' позволяет пользователю wet обращаться к серверу MySQL с любых компьютеров сети.

Все учетные записи хранятся в таблице user системной базы данных с именем mysql. После первой инсталляции содержимое таблицы userвыглядит так, как показано в листинге.

```
mysql> SELECT Host,User,Password FROM mysql.user;
+-----------------------+------+----------+
| Host                  | User | Password |
+-----------------------+------+----------+
| localhost             | root |          |
| production.mysql.com  | root |          |
| 127.0.0.1             | root |          |
| localhost             |      |          |
| production.mysql.com  |      |          |
+-----------------------+------+----------+
5 rows in set (0.27 sec)
```

2.6.1 Создание новой учетной записи

Создать учетную запись позволяет оператор

```
CREATE USER 'username' @ 'host' [IDENTIFIED BY [PASSWORD] 'пароль'];
```

Оператор создает новую учетную запись с необязательным паролем. Если пароль не указан, в его качестве выступает пустая строка. Разумно хранить пароль в виде хэш-кода, полученного в результате необратимого шифрования. Чтобы воспользоваться этим механизмом авторизации, необходимо поместить между ключевым словом identified by и паролем ключевое слово password.

2.6.2 Удаление учетной записи

Удалить учетную запись позволяет оператор

```
DROP USER 'username' @ 'host';
```

Изменение имени пользователя в учетной записи. Осуществляется с помощью оператора

```
RENAME USER старое_имя TO новое_имя;
```

2.6.3 Назначение привилегий

Рассмотренные выше операторы позволяют создавать, удалять и редактировать учетные записи, но они не позволяют изменять привилегии пользователя – сообщать MySQL, какой пользователь имеет право только на чтение информации, какой на чтение и редактирование, а кому предоставлены права изменять структуру БД и создавать учетные записи.

Для решения этих задач предназначены операторы grant (назначает привилегии) и revoke (удаляет привилегии). Если учетной записи, котораяпоказана в операторе grant, не существует, то она автоматически создается. Удаление всех привилегий с помощью оператора revoke не приводит к автоматическому уничтожению учетной записи. Для удаления пользователя необходимо воспользоваться оператором drop user.

В простейшем случае оператор grant выглядит следующим образом:

```
mysql> GRANT ALL ON *.* TO 'wet'@'localhost' IDENTIFIED BY 'pass';
Query OK, 0 rows affected (0.17 sec)
```

Данный запрос создает пользователя с именем wet и паролем pass, который может обращаться к серверу с локального хоста (localhost) и имеет все права (all) для всех баз данных (*.*). Если такой пользователь существует, то его привилегии будут изменены на all.

Вместо ключевого слова all можно использовать любое из ключевых слов, представленных в табл. 9. Ключевое слово on в операторе grantзадает уровень привилегий, которые могут быть заданы на одном из четырех уровней, представленных в табл. 10. Для таблиц можно установить только следующие типы привилегий: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT OPTION, INDEX и ALTER. Это следует учитывать при использовании конструкции grant all, которая назначает привилегии на текущем уровне. Так, запрос уровня базы данных grant all on db.* не предоставляет никаких глобальных привилегий.

Отмена привилегий. Для отмены привилегий используется оператор revoke:

```
mysql> REVOKE DELETE, UPDATE ON *.* FROM 'wet'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

Оператор revoke отменяет привилегии, но не удаляет учетные записи, для их удаления необходимо воспользоваться оператором drop USER.

Таблица 9

ПРИВИЛЕГИЯ	ОПЕРАЦИЯ, РАЗРЕШЕННАЯ ПРИВИЛЕГИЕЙ
ALL [PRIVILEGES]	Комбинация всех привилегий, за исключением привилегии GRANT option, которая задается отдельно
ALTER	Позволяет редактировать таблицу с помощью оператора ALTER TABLE
ALTER ROUTINE	Позволяет редактировать или удалять хранимую процедуру
CREATE	Позволяет создавать таблицу при помощи оператора CREATE TABLE
CREATE ROUTINE	Позволяет создавать хранимую процедуру
CREATE TEMPORARY TABLES	Позволяет создавать временные таблицы
CREATE USER	Позволяет работать с учетными записями с помощью CREATE USER, DROP USER, RENAME USER и REVOKE ALL PRIVILEGES
CREATE VIEW	Позволяет создавать представление с помощью оператора CREATE VIEW
DELETE	Позволяет удалять записи при помощи оператора delete
DROP	Позволяет удалять таблицы при помощи оператора DROP TABLE
EXECUTE	Позволяет выполнять хранимые процедуры
INDEX	Позволяет работать с индексами, в частности, использовать операторы CREATE INDEX и DROP INDEX
INSERT	Позволяет добавлять в таблицу новые записи оператором insert
LOCK TABLES	Позволяет осуществлять блокировки таблиц при помощи операторов LOCK TABLES и UNLOCK TABLES. Для вступления в действие этой привилегии должна быть установлена привилегия SELECT
select	Позволяет осуществлять выборки таблиц оператором SELECT
Show databases	Позволяет просматривать список всех таблиц на сервере при помощи оператора show databases
Show view	Позволяет использовать оператор show create view
UPDATE	Позволяет обновлять содержимое таблиц оператором UPDATE
USAGE	Синоним для статуса «отсутствуют привилегии»
GRANT OPTION	Позволяет управлять привилегиями других пользователей, без данной привилегии невозможно выполнить операторы grant и REVOKE

Таблица 10

КЛЮЧЕВОЕ СЛОВО ON	УРОВЕНЬ

ON *.*	Глобальный уровень – пользователь с полномочиями на глобальном уровне может обращаться ко всем БД и таблицам, входящим в их состав
ON db.*	Уровень базы данных – привилегии распространяются на таблицы базы данных db
ON db.tbl	Уровень таблицы – привилегии распространяются на таблицу tbl базы данных db
ON db.tbl	Уровень столбца – привилегии касаются отдельных столбцов в таблице tbl базы данных db. Список столбцов указывается в скобках через запятую после ключевых слов select, insert, update

2.6.4 Контрольные вопросы

- Создание новой учетной записи
- Удаление учетной записи
- Назначение привилегий

2.6.5 Задание к лабораторной работе 4

На этапе проектирования физической структуры необходимо для заданной предметной области средствами СУБД:

- создать представления;
- написать две хранимые процедуры и включить их в БД;
- два триггера для разных таблиц базы данных;
- задать права доступа к объектам базы данных;
- оформить отчет по лабораторной работе.