МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ

ТИПЫ ДАННЫХ MYSQL

Учебно-методическое пособие
по дисциплине
«ВЕБ-ПРОГРАММИРОВАНИЕ»

Набережные Челны
2018

Галиуллин Л.А. Типы данных MySQL: учебно-методическое пособие по дисциплине «Веб-программирование» [Электронный ресурс] / Казанский федеральный университет, Электронный архив, 2018.

Рассматривается архитектура СУБД MySQL. Представлены методики создания и удаления таблиц, описаны типы данных MySQL, основные команды SQL-запросов. Приведены контрольные вопросы. Для студентов направлений подготовки «Информатика и вычислительная техника», «Программная инженерия».

Введение

SQL больше напоминает естественный человеческий, а не компьютерный язык. SQL добивается этого сходства благодаря четкой императивной структуре. Во многом походя на предложение английского языка, отдельные команды SQL, называемые запросами, могут быть разбиты на части речи.

Большинство реализаций SQL, включая MySQL, нечувствительны к регистру: неважно, в каком регистре вы вводите ключевые слова SQL, если орфография верна.

Нечувствительность к регистру относится только к ключевым словам SQL. В MySQL имена баз данных, таблиц и колонок к регистру чувствительны. Но это характерно не для всех СУБД. Поэтому, если вы пишете приложение, которое должно работать с любыми СУБД, не следует использовать имена, различающиеся одним только регистром.

Первый элемент SQL-запроса - всегда глагол. Глагол выражает действие, которое должно выполнить ядро базы данных. Хотя остальная часть команды зависит от глагола, она всегда следует общему формату: указывается имя объекта, над которым осуществляется действие, а затем описываются используемые при действии данные. Например, в запросе CREATE TABLE people (char(10)) используется глагол CREATE, за которым следует дополнение (объект) TABLE. Оставшаяся часть запроса описывает таблицу, которую нужно создать.

SQL-запрос исходит от клиента - приложения, с помощью которого пользователь взаимодействует с базой данных. Клиент составляет запрос, основываясь на действиях пользователя, и посылает его серверу SQL. После этого сервер должен обработать запрос и выполнить указанные действия. Сделав свою работу, сервер возвращает клиенту одно или несколько значений.

Поскольку основная задача SQL - сообщить серверу баз данных о том, какие действия необходимо выполнить, он не обладает гибкостью языка общего назначения. Большинство функций SQL связано с вводом и выводом из базы: добавление, изменение, удаление и чтение данных. SQL предоставляет и другие возможности, но всегда с оглядкой на то, как они могут

использоваться для манипулирования данными в базе.

Создание и удаление таблиц

Успешно установив MySQL, вы можете приступить к созданию своей первой таблицы. *Таблица,* структурированное вместилище данных, является основным понятием реляционных баз. Прежде чем начать вводить данные в таблицу, вы должны определить ее структуру. Рассмотрим следующую раскладку.

people	
name	char(10) not null
address	text(100)
id	int

Таблица содержит не только имена колонок, но и тип каждого поля, а также возможные дополнительные сведения о полях. Тип данных поля определяет, какого рода данные могут в нем содержаться. Типы данных SQL сходны с типами данных в других языках программирования. Полный стандарт SQL допускает большое разнообразие типов данных. MySQL реализует большую их часть.

Общий синтаксис для создания таблиц следующий:

CREATE TABLE *table_name (column_name1 type [modifiers] [, column_name2 type [modifiers]]*)

Какие идентификаторы - имена таблиц и колонок - являются допустимыми, зависит от конкретной СУБД. В MySQL длина идентификатора может быть до 64 символов, допустим символ «$», и первым символом может быть цифра. Более важно, однако, что MySQL допускает использование любых символов из установленного в системе локального набора. Для хорошей переносимости SQL избегайте имен, начинающихся не с допустимой буквы.

Колонка - это отдельная единица данных в таблице. В таблице может содержаться произвольное число колонок, но использование больших таблиц бывает неэффективным. Создав правильно нормализованные таблицы, можно объединять их («join») для осуществления поиска в данных, размещенных в нескольких таблицах. Механику объединения таблиц мы

обсудим позднее.

Как и бывает в жизни, разрушить легче, чем создать. Следующая команда удаляет таблицу:

DROP TABLE *table_name*

MySQL уничтожит все данные удаленной таблицы. Если у вас не осталось резервной копии, нет абсолютно никакого способа отменить действие данной операции. Поэтому всегда храните резервные копии и будьте очень внимательны при удалении таблиц. В один «прекрасный» день это вам пригодится.

В MySQL можно одной командой удалить несколько таблиц, разделяя их имена запятыми. Например, DROP TABLE *people, animals, plants* удалит эти три таблицы. Можно также использовать модификатор IF EXISTS для подавления ошибки в случае отсутствия удаляемой таблицы. Этот модификатор полезен в больших сценариях, предназначенных для создания базы данных и всех ее таблиц. Прежде чем создавать таблицу, выполните команду DROP TABLE *table_name* IF EXISTS.

Типы данных в SQL

Каждая колонка таблицы имеет тип. Типы данных SQL сходны с типами данных традиционных языков программирования. В то время как во многих языках определен самый минимум типов, необходимых для работы, в SQL для удобства пользователей определены дополнительные типы, такие как MONEY и DATE. Данные типа MONEY можно было бы хранить и как один из основных числовых типов данных, однако использование типа, специально учитывающего особенности денежных расчетов, повышает легкость использования SQL.

Таблица 1. *Наиболее употребительные типы данных, поддерживаемые MySQL*

Тип данных	**Описание**
INT	Целое число, может быть со знаком или без знака.
REAL	Число с плавающей запятой. Этот тип допускает

	больший диапазон значений, чем INT, но не обладает его точностью.
CHAR(length)	Символьная величина фиксированной длины. Поля типа CHAR не могут содержать строки длины большей, чем указанное значение. Поля меньшей длины дополняются пробелами.
TEXT(length)	Символьная величина переменной длины. TEXT - лишь один из нескольких типов данных переменного размера.
DATE	Стандартное значение даты.
TIME	Стандартное значение времени. Этот тип используется для хранения времени дня безотносительно какой-либо даты. При использовании вместе с датой позволяет хранить конкретную дату и время. Есть дополнительный тип DATETIME для совместного хранения даты и времени в одном поле.

MySQL поддерживает атрибут UNSIGNED для всех числовых типов. Этот модификатор позволяет вводить в колонку только положительные (беззнаковые) числа. Беззнаковые поля имеют верхний предел значений вдвое больший, чем у соответствующих знаковых типов. Беззнаковый TINYINT - однобайтовый числовой тип MySQL - имеет диапазон от 0 до 255, а не от -127 до 127, как у своего знакового аналога.

MySQL имеет больше типов, чем перечислено выше. Однако на практике в основном используются перечисленные типы. Размер данных, которые вы собираетесь хранить, играет гораздо большую роль при разработке таблиц MySQL.

Числовые типы данных

Прежде чем создавать таблицу, вы должны хорошо представить себе, какого рода данные вы будете в ней хранить. Помимо очевидного решения о том, будут это числовые или

символьные данные, следует выяснить примерный размер хранимых данных. Если это числовое поле, то каким окажется максимальное значение? Может ли оно измениться в будущем? Если минимальное значение всегда положительно, следует рассмотреть использование беззнакового типа. Всегда следует выбирать самый маленький числовой тип, способный хранить самое большое мыслимое значение. Если бы требовалось хранить в поле численность населения штата, следовало бы выбрать беззнаковый INT. В штате не может быть отрицательной численности населения, а чтобы беззнаковое поле типа INT не могло вместить число, представляющее его население, численность населения штата должна примерно равняться численности населения всей Земли.

Символьные типы

С символьными типами работать немного труднее. Вы должны подумать не только о максимальной и минимальной длине строки, но также о среднем размере, частоте отклонения от него и необходимости в индексировании. В данном контексте мы называем *индексом* поле или группу полей, в которых вы собираетесь осуществлять поиск — в основном, в предложении WHERE. Индексирование, однако, значительно сложнее, чем такое упрощенное определение, и мы займемся им далее. Здесь важно лишь отметить, что индексирование по символьным полям происходит значительно быстрее, если они имеют фиксированную длину. Если длина строк не слишком колеблется или, что еще лучше, постоянна, то, вероятно, лучше выбрать для поля тип CHAR. Хороший кандидат на тип CHAR - код страны. Стандартом ISO определены двух символьные коды для всех стран. CHAR(2) будет правильным выбором для данного поля.

Чтобы подходить для типа CHAR, поле необязательно должно быть фиксированной длины, но длина не должна сильно колебаться. Телефонные номера, к примеру, можно смело хранить в поле CHAR(13), хотя длина номеров различна в разных странах. Просто различие не столь велико, поэтому нет смысла делать поле для номера телефона переменным по длине.

В отношении поля типа CHAR важно помнить, что, вне зависимости от реальной длины хранимой строки, в поле будет ровно столько символов, сколько указано в его размере - не больше и не меньше. Разность в длине между размером сохраняемого текста и размером поля заполняется пробелами. Не стоит беспокоиться по поводу нескольких лишних символов при хранении телефонных номеров, но не хотелось бы тратить много места в некоторых других случаях. Для этого существуют текстовые поля переменной длины.

Хороший пример поля, для которого требуется тип данных с переменной длиной, дает URL Интернет. По большей части адреса Web занимают сравнительно немного места - *http://www.ora.com, http://www.hughes.com.au, http://www.mysql.com* - и не представляют проблемы. Иногда, однако, можно наткнуться на адреса подобного вида:

http://www.winespectator.com/Wine/Spectator/notes\5527293926834323221480431354?XvlI=&Xr5=&Xvl =&type-region-search-code=&Xa14=flora+springs&Xv4=.

Если создать поле типа CHAR длины, достаточной для хранения этого URL, то почти для каждого другого хранимого URL будет напрасно тратиться весьма значительное пространство. Поля переменной длины позволяют задать такую длину, что оказывается возможным хранение необычно длинных значений, и в то же время не расходуется напрасно место при хранении обычных коротких величин.

Поля переменной длины

Преимуществом текстовых полей переменной длины является то, что они используют ровно столько места, сколько необходимо для хранения отдельной величины. Например, поле типа VARCHAR(255) , в котором хранится строка «hello, world», занимает только двенадцать байтов (по одному байту на каждый символ плюс еще один байт для хранения длины). В отличие от стандарта ANSI, в MySQL поля типа VARCHAR не дополняются пробелами. Перед записью из строки удаляются лишние пробелы.

Сохранить строки, длина которых больше, чем заданный

размер поля, нельзя. В поле VARCHAR(4) можно сохранить строку не длиннее 4 символов. Если вы попытаетесь сохранить строку «happy birthday», MySQL сократит ее до «happ». Недостатком подхода MySQL к хранению полей переменной длины является то, что не существует способа сохранить необычную строку, длина которой превосходит заданное вами значение. В таблице 3.5 показан размер пространства, необходимого для хранения 144-символьного URL, продемонстрированного выше, и обычного, 30-символьного URL.

Таблица 2. *Пространство памяти, необходимое для различных символьных типов MySQL*

Тип данных	Пространство для хранения строки из 144 символов	Пространство для хранения строки из 30 символов	Максимальная длина строки
CHAR(150)	150	150	255
VARCHAR(150)	145	31	255
TINYTEXT(150)	145	31	255
TEXT(150)	146	32	65535
MEDIUMTEXT(150)	147	33	16777215
LONGTEXT(150)	148	34	4294967295

Если через годы работы со своей базой данных вы обнаружите, что мир изменился, и поле, уютно чувствовавшее себя в типе VARCHAR(25), должно теперь вмещать строки длиной 30 символов, не все потеряно. В MySQL есть команда ALTER TABLE , позволяющая переопределить размер поля без потери данных.

ALTER TABLE *mytable* MODIFY *mycolumn LONGTEXT*

Двоичные типы данных

В MySQL есть целый ряд двоичных типов данных,

соответствующих своим символьным аналогам. Двоичными типами, поддерживаемыми MySQL, являются CHAR BINARY, VARCHAR BINARY, TINYBLOB, BLOB, MEDIUMBLOB и LONGBLOB. Практическое отличие между символьными типами и их двоичными аналогами основано на принципе кодировки. *Двоичные данные* просто являются куском данных, которые MySQL не пытается интерпретировать. Напротив, *символьные данные* предполагаются представляющими текстовые данные из используемых человеком алфавитов. Поэтому они кодируются и сортируются, основываясь на правилах, соответствующих рассматриваемому набору символов. Двоичные же данные MySQL сортирует в порядке ASCII без учета регистра.

Перечисления и множества

MySQL предоставляет еще два особых типа данных. Тип ENUM позволяет при создании таблицы указать список возможных значений некоторого поля. Например, если бы у вас была колонка с именем «фрукт», в которую вы разрешили бы помещать только значения «яблоко», «апельсин», «киви» и «банан», ей следовало бы присвоить тип ENUM:

```
CREATE TABLE meal(meal_id INT NOT NULL PRIMARY KEY,
фрукт ENUM('яблоко', 'апельсин', 'киви', 'банан'))
```

При записи значения в эту колонку оно должно быть одним из перечисленных фруктов. Поскольку MySQL заранее знает, какие значения допустимы для этой колонки, она может абстрагировать их каким-либо числовым типом. Иными словами, вместо того, чтобы хранить в колонке «яблоко» в виде строки, MySQL заменяет его однобайтовым числом, а «яблоко» вы видите, когда обращаетесь к таблице или выводите из нее результаты.

Тип MySQL SET позволяет одновременно хранить в поле несколько значений.

Другие типы данных

Любые мыслимые данные можно хранить с помощью числовых или символьных типов. В принципе, даже числа можно хранить в символьном виде. Однако то, что это можно сделать, не означает, что это нужно делать. Рассмотрим, к примеру, как хранить в базе данных денежные суммы. Можно делать это, используя INT или REAL. Хотя интуитивно REAL может показаться более подходящим - в конце концов, в денежных суммах нужны десятичные знаки, - на самом деле более правильно использовать INT. В полях, содержащих значения с плавающей запятой, таких как REAL, часто невозможно найти число с точным десятичным значением. Например, если вы вводите число 0.43, которое должно представлять сумму $0.43, MySQL может записать его как 0.42999998. Это небольшое отличие может вызвать проблемы при совершении большого числа математических операций.

MySQL предоставляет специальные типы данных для таких денежных сумм. Одним из них является тип MONEY, другим - DATE.

Индексы

Хотя MySQL обеспечивает более высокую производительность, чем любые большие серверы баз данных, некоторые задачи все же требуют осторожности при проектировании базы данных. Например, если таблица содержит миллионы строк, поиск нужной строки в ней наверняка потребует длительного времени. В большинстве баз данных поиск облегчается благодаря средству, называемому индексом.

Индексы способствуют хранению данных в базе, таким образом, который позволяет осуществлять быстрый поиск. К несчастью, ради скорости поиска приходится жертвовать дисковым пространством и скоростью изменения данных. Наиболее эффективно создавать индексы для тех колонок, в которых вы чаще всего собираетесь осуществлять поиск. MySQL поддерживает следующий синтаксис для создания

индексов:

CREATE INDEX *index_name ON tablename (column1, column2, columnN)*

MySQL позволяет также создавать индекс одновременно с созданием таблицы, используя следующий синтаксис:

CREATE TABLE *materials (id* INT NOT NULL,
name CHAR(50) NOT NULL,
resistance INT,
melting_pt REAL,
INDEX *index1 (id, name),*
UNIQUE INDEX *index2 (name))*

В этом примере для таблицы создается два индекса. Первый индекс index1 состоит из полей id и name. Второй индекс включает в себя только поле name и указывает, что значения поля name должны быть уникальными. Если вы попытаетесь вставить в поле name значение, которое уже есть в этом поле в какой-либо строке, операция не будет осуществлена. Все поля, указанные в уникальном индексе, должны быть объявлены как NOT NULL .

Хотя мы создали отдельный индекс для поля name, отдельно для поля id мы не создавали индекса. Если такой индекс нам понадобится, создавать его не нужно - он уже есть. Когда индекс содержит более одной колонки (например, name, rank, и serial_number), MySQL читает колонки в порядке слева направо. Благодаря используемой MySQL структуре индекса всякое подмножество колонок с левого края автоматически становится индексом внутри «главного» индекса. Например, когда вы создаете индекс name, rank, serial_number, создаются также «свободные» индексы name и name вместе с rank. Однако индексы rank или name и serial_number не создаются, если не потребовать этого явно.

MySQL поддерживает также семантику ANSI SQL для особого индекса, называемого первичным ключом. В MySQL первичный ключ - это уникальный индекс с именем PRIMARY. Назначив при создании таблицы колонку первичным ключом, вы делаете ее уникальным индексом, который будет поддерживать объединения таблиц. В следующем примере создается таблица cities с первичным ключом id.

CREATE TABLE cities (id INT NOT NULL PRIMARY KEY,
name VARCHAR(100),
pop MEDIUMINT,
founded DATE)

Прежде чем создавать таблицу, нужно решить, какие поля будут ключами (и будут ли вообще ключи). Как уже говорилось, любые поля, которые будут участвовать в объединении таблиц, являются хорошими кандидатами на роль первичного ключа.

Последовательности и автоинкрементирование

Лучше всего, когда первичный ключ не имеет в таблице никакого иного значения, кроме значения первичного ключа. Для достижения этого лучшим способом является создание числового первичного ключа, значение которого увеличивается при добавлении в таблицу новой строки. Если вернуться к примеру с таблицей cities, то первый введенный вами город должен иметь id, равный 1, второй - 2, третий - 3, и т. д. Чтобы успешно управлять такой последовательностью первичных ключей, нужно иметь какое-то средство, гарантирующее, что в данный конкретный момент только один клиент может прочесть число и увеличить его на единицу. В базе данных с транзакциями можно создать таблицу, скажем, с именем sequence, содержащую число, представляющее очередной id. Когда необходимо добавить новую строку в таблицу, вы читаете число из этой таблицы и вставляете число на единицу большее. Чтобы эта схема работала, нужно быть уверенным, что никто другой не сможет произвести чтение из таблицы, пока вы не ввели новое число. В противном случае два клиента могут прочесть одно и то же значение и попытаться использовать его в качестве значения первичного ключа в одной и той же таблице.

MySQL не поддерживает транзакции, поэтому описанный механизм нельзя использовать для генерации уникальных чисел. Использовать для этих целей команду MySQL LOCK TABLE обременительно. Тем не менее СУБД предоставляет свой вариант понятия последовательности, позволяющий генерировать уникальные идентификаторы, не беспокоясь о транзакциях.

Контрольные вопросы

1. Что Вы знаете о SQL?
2. Что Вы знаете об архитектуре MySQL?
3. Что Вы знаете о создании и удалении таблиц?
4. Что Вы знаете о типах данных в SQL?
5. Что Вы знаете о числовых типах данных?
6. Что Вы знаете о символьных типах данных?
7. Что Вы знаете о полях переменной длины?
8. Что Вы знаете о двоичных типах данных?
9. Что Вы знаете о перечислениях и множествах?
10. Что Вы знаете об индексах?

Рекомендуемые источники

1. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2015. - 416 с. [Электронный ресурс]. http://znanium.com/bookread.php?book=336649.
2. Гагарина Л.Г. Технология разработки программного обеспечения: Учеб. пос. / Л.Г.Гагарина, Е.В.Кокорева, Б.Д.Виснадул; Под ред. проф. Л.Г.Гагариной - М.: ИД ФОРУМ: НИЦ Инфра-М, 2017. - 400 с. [Электронный ресурс]. http://znanium.com/bookread.php?book=389963.
3. Голицына О. Л. Программирование на языках высокого уровня: Учебное пособие / О.Л. Голицына, И.И. Попов. - М.: Форум, 2016. - 496 с. [Электронный ресурс]. http://znanium.com/bookread.php?book=139428.