关系数据库入门

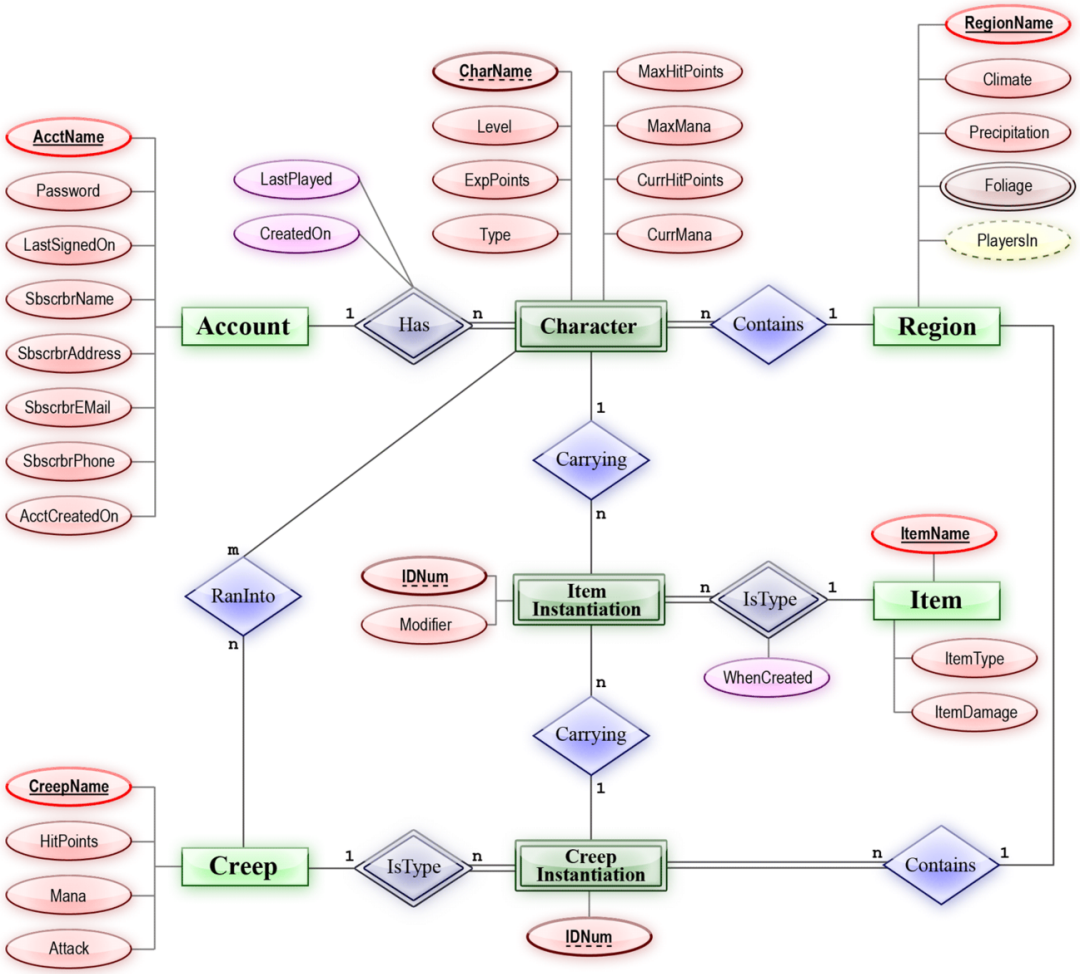
关系数据库概述

1. 数据持久化 - 将数据保存到能够长久保存数据的存储介质中，在掉电的情况下数据也不会丢失。
2. 数据库发展史 - 网状数据库、层次数据库、关系数据库、NoSQL数据库。

   > 1970年，IBM的研究员E.F.Codd在*Communication of the ACM*上发表了名为*A Relational Model of Data for Large Shared Data Banks*的论文，提出了关系模型的概念，奠定了关系模型的理论基础。后来Codd又陆续发表多篇文章，论述了范式理论和衡量关系系统的12条标准，用数学理论奠定了关系数据库的基础。

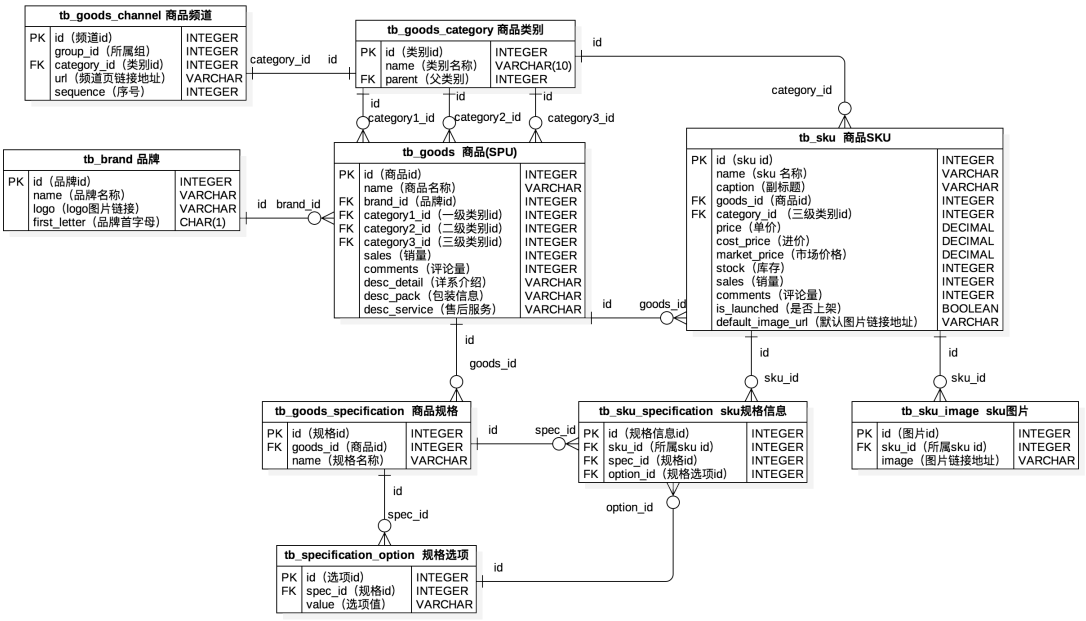
3. 关系数据库特点。
   - 理论基础：集合论和关系代数。
   - 具体表象：用二维表（有行和列）组织数据。
   - 编程语言：结构化查询语言（SQL）。
4. ER模型（实体关系模型）和概念模型图。

   ER模型，全称为**实体关系模型**（Entity-Relationship Model），由美籍华裔计算机科学家陈品山先生提出，是概念数据模型的高层描述方式，如下图所示。



   - 实体 - 矩形框
   - 属性 - 椭圆框
   - 关系 - 菱形框
   - 重数 - 1:1（一对一） / 1:N（一对多） / M:N（多对多）

实际项目开发中，我们可以利用数据库建模工具（如：PowerDesigner）来绘制概念数据模型（其本质就是ER模型），然后再设置好目标数据库系统，将概念模型转换成物理模型，最终生成创建二维表的SQL（很多工具都可以根据我们设计的物理模型图以及设定的目标数据库来导出SQL或直接生成数据表）。



5. 关系数据库产品。

   - Oracle - 目前世界上使用最为广泛的数据库管理系统，作为一个通用的数据库系统，它具有完整的数据管理功能；作为一个关系数据库，它是一个完备关系的产品；作为分布式数据库，它实现了分布式处理的功能。在Oracle最新的12c版本中，还引入了多承租方架构，使用该架构可轻松部署和管理数据库云。
   - DB2 - IBM公司开发的、主要运行于Unix（包括IBM自家的AIX）、Linux、以及Windows服务器版等系统的关系数据库产品。DB2历史悠久且被认为是最早使用SQL的数据库产品，它拥有较为强大的商业智能功能。
   - SQL Server - 由Microsoft开发和推广的关系型数据库产品，最初适用于中小企业的数据管理，但是近年来它的应用范围有所扩展，部分大企业甚至是跨国公司也开始基于它来构建自己的数据管理系统。
   - MySQL - MySQL是开放源代码的，任何人都可以在GPL（General Public License）的许可下下载并根据个性化的需要对其进行修改。MySQL因为其速度、可靠性和适应性而备受关注。
   - PostgreSQL - 在BSD许可证下发行的开放源代码的关系数据库产品。

MySQL简介

MySQL最早是由瑞典的MySQL AB公司开发的一个开放源码的关系数据库管理系统，该公司于2008年被昇阳微系统公司（Sun Microsystems）收购。在2009年，甲骨文公司（Oracle）收购昇阳微系统公司，因此在这之后MySQL成为了Oracle旗下产品。

MySQL在过去由于性能高、成本低、可靠性好，已经成为最流行的开源数据库，因此被广泛地应用于中小型网站开发。随着MySQL的不断成熟，它也逐渐被应用于更多大规模网站和应用，比如维基百科、谷歌（Google）、脸书（Facebook）、淘宝网等网站都使用了MySQL来提供数据持久化服务。

甲骨文公司收购后昇阳微系统公司，大幅调涨MySQL商业版的售价，且甲骨文公司不再支持另一个自由软件项目OpenSolaris的发展，因此导致自由软件社区对于Oracle是否还会持续支持MySQL社区版（MySQL的各个发行版本中唯一免费的版本）有所担忧，MySQL的创始人麦克尔·维德纽斯以MySQL为基础，成立分支计划MariaDB（以他女儿的名字命名的数据库）。有许多原来使用MySQL数据库的公司（例如：维基百科）已经陆续完成了从MySQL数据库到MariaDB数据库的迁移。

1. 安装和配置

> **说明**：下面的安装和配置都是以CentOS Linux环境为例，如果需要在其他系统下安装MySQL，读者可以自行在网络上查找对应的安装教程）。

- 刚才说过，MySQL有一个分支版本名叫MariaDB，该数据库旨在继续保持MySQL数据库在GNU GPL下开源。如果要使用MariaDB作为MySQL的替代品，可以使用下面的命令进行安装。

  ```
  yum install mariadb mariadb-server
  ```

- 如果要安装官方版本的MySQL，可以在MySQL官方网站下载安装文件。首先在下载页面中选择平台和版本，然后找到对应的下载链接。下面以MySQL 5.7.26版本和Red Hat Enterprise Linux为例，直接下载包含所有安装文件的归档文件，解归档之后通过包管理工具进行安装。

  ```
  wget https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.26-1.el7.x86_64.rpm-bundle.tar
  tar -xvf mysql-5.7.26-1.el7.x86_64.rpm-bundle.tar
  ```

  如果系统上有MariaDB相关的文件，需要先移除MariaDB相关的文件。

  ```
  yum list installed | grep mariadb | awk '{print $1}' | xargs yum erase -y
  ```

  接下来可以按照如下所示的顺序用RPM（Redhat Package Manager）工具安装MySQL。

  ```
  rpm -ivh mysql-community-common-5.7.26-1.el7.x86_64.rpm
  rpm -ivh mysql-community-libs-5.7.26-1.el7.x86_64.rpm
  rpm -ivh mysql-community-client-5.7.26-1.el7.x86_64.rpm
  rpm -ivh mysql-community-server-5.7.26-1.el7.x86_64.rpm
  ```

  可以使用下面的命令查看已经安装的MySQL相关的包。

  ```
  rpm -qa | grep mysql
  ```

- 配置MySQL。

  MySQL的配置文件在`/etc`目录下，名为`my.cnf`，默认的配置文件内容如下所示。如果对这个文件不理解并没有关系，什么时候用到这个配置文件什么时候再了解它就行了。

  ```
  cat /etc/my.cnf
  ```

  ```
  # For advice on how to change settings please see
  # http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html

  [mysqld]
  #
  # Remove leading # and set to the amount of RAM for the most important data
  # cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
  # innodb_buffer_pool_size = 128M
  #
  ```

```
# Remove leading # to turn on a very important data integrity option:
logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

# Disabling symbolic-links is recommended to prevent assorted security
risks
symbolic-links=0

log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

- 启动MySQL服务。

  可以使用下面的命令来启动MySQL。

  ```
  service mysqld start
  ```

  在CentOS 7中，更推荐使用下面的命令来启动MySQL。

  ```
  systemctl start mysqld
  ```

  启动MySQL成功后，可以通过下面的命令来检查网络端口使用情况，MySQL默认使用3306端口。

  ```
  netstat -ntlp | grep mysql
  ```

  也可以使用下面的命令查找是否有名为mysqld的进程。

  ```
  pgrep mysqld
  ```

- 使用MySQL客户端工具连接服务器。

  命令行工具：

  ```
  mysql -u root -p
  ```

  > 说明：启动客户端时，`-u`参数用来指定用户名，MySQL默认的超级管理账号为`root`；`-p`表示要输入密码（用户口令）；如果连接的是其他主机而非本机，可以用`-h`来指定连接主机的主机名或IP地址。

  如果是首次安装MySQL，可以使用下面的命令来找到默认的初始密码。

  ```
  cat /var/log/mysqld.log | grep password
  ```

上面的命令会查看MySQL的日志带有password的行，在显示的结果中`root@localhost:`后面的部分就是默认设置的初始密码。

修改超级管理员（root）的访问口令为`123456`。

```
set global validate_password_policy=0;
set global validate_password_length=6;
alter user 'root'@'localhost' identified by '123456';
```

> **说明**：MySQL较新的版本默认不允许使用弱口令作为用户口令，所以我们通过上面的前两条命令修改了验证用户口令的策略和口令的长度。事实上我们不应该使用弱口令，因为存在用户口令被暴力破解的风险。近年来，攻击数据库窃取数据和劫持数据库勒索比特币的事件屡见不鲜，要避免这些潜在的风险，最为重要的一点是不要让数据库服务器暴露在公网上（最好的做法是将数据库置于内网，至少要做到不向公网开放数据库服务器的访问端口），另外要保管好`root`账号的口令，应用系统需要访问数据库时，通常不使用`root`账号进行访问，而是创建其他拥有适当权限的账号来访问。

再次使用客户端工具连接MySQL服务器时，就可以使用新设置的口令了。在实际开发中，为了方便用户操作，可以选择图形化的客户端工具来连接MySQL服务器，包括：

- MySQL Workbench（官方提供的工具）
- Navicat for MySQL（界面简单优雅，功能直观强大）
- SQLyog for MySQL（强大的MySQL数据库管理员工具）

2. 常用命令。
    - 查看服务器版本。

      ```
      select version();
      ```

    - 查看所有数据库。

      ```
      show databases;
      ```

    - 切换到指定数据库。

      ```
      use mysql;
      ```

    - 查看数据库下所有表。

      ```
      show tables;
      ```

    - 获取帮助。

      ```
      ? contents;
      ? functions;
      ? numeric functions;
      ? round;

      ? data types;
      ? longblob;
      ```

SQL详解

基本操作

我们通常可以将SQL分为三类：DDL（数据定义语言）、DML（数据操作语言）和DCL（数据控制语言）。DDL主要用于创建（create）、删除（drop）、修改（alter）数据库中的对象，比如创建、删除和修改二维表；DML主要负责插入数据（insert）、删除数据（delete）、更新数据（update）和查询（select）；DCL通常用于授予权限（grant）和召回权限（revoke）。

> 说明：SQL是不区分大小写的语言，为了书写方便，下面的SQL都使用了小写字母来书写。

1. DDL（数据定义语言）

```
-- 如果存在名为school的数据库就删除它
drop database if exists school;

-- 创建名为school的数据库并设置默认的字符集和排序方式
create database school default charset utf8;

-- 切换到school数据库上下文环境
use school;

-- 创建学院表
create table tb_college
(
collid      int auto_increment comment '编号',
collname    varchar(50) not null comment '名称',
collintro   varchar(500) default '' comment '介绍',
primary key (collid)
);

-- 创建学生表
create table tb_student
(
stuid       int not null comment '学号',
stuname     varchar(20) not null comment '姓名',
stusex      boolean default 1 comment '性别',
stubirth    date not null comment '出生日期',
stuaddr     varchar(255) default '' comment '籍贯',
collid      int not null comment '所属学院',
primary key (stuid),
foreign key (collid) references tb_college (collid)
);

-- 创建教师表
create table tb_teacher
(
teaid       int not null comment '工号',
teaname     varchar(20) not null comment '姓名',
teatitle    varchar(10) default '助教' comment '职称',
collid      int not null comment '所属学院',
primary key (teaid),
foreign key (collid) references tb_college (collid)
);

-- 创建课程表
create table tb_course
(
couid       int not null comment '编号',
couname     varchar(50) not null comment '名称',
coucredit   int not null comment '学分',
```

```
teaid       int not null comment '授课老师',
primary key (couid),
foreign key (teaid) references tb_teacher (teaid)
);

-- 创建选课记录表
create table tb_record
(
recid       int auto_increment comment '选课记录编号',
sid         int not null comment '选课学生',
cid         int not null comment '所选课程',
seldate     datetime default now() comment '选课时间日期',
score       decimal(4,1) comment '考试成绩',
primary key (recid),
foreign key (sid) references tb_student (stuid),
foreign key (cid) references tb_course (couid),
unique (sid, cid)
);
```

上面的DDL有几个地方需要强调一下：

- 创建数据库时，我们通过`default charset utf8`指定了数据库默认使用的字符集，我们推荐使用该字符集，因为utf8能够支持国际化编码。如果将来数据库中用到的字符可能包括类似于Emoji这样的图片字符，也可以将默认字符集设定为utf8mb4（最大4字节的utf-8编码）。查看MySQL支持的字符集可以执行下面的语句。

  ```
  show character set;
  ```

  ```
  +----------+---------------------------------+---------------------+---
  -----+
  | Charset  | Description                     | Default collation   |
  Maxlen |
  +----------+---------------------------------+---------------------+---
  -----+
  | big5     | Big5 Traditional Chinese        | big5_chinese_ci     |
     2 |
  | dec8     | DEC West European               | dec8_swedish_ci     |
     1 |
  | cp850    | DOS West European               | cp850_general_ci    |
     1 |
  | hp8      | HP West European                | hp8_english_ci      |
     1 |
  | koi8r    | KOI8-R Relcom Russian           | koi8r_general_ci    |
     1 |
  | latin1   | cp1252 West European            | latin1_swedish_ci   |
     1 |
  | latin2   | ISO 8859-2 Central European     | latin2_general_ci   |
     1 |
  | swe7     | 7bit Swedish                    | swe7_swedish_ci     |
     1 |
  | ascii    | US ASCII                        | ascii_general_ci    |
     1 |
  | ujis     | EUC-JP Japanese                 | ujis_japanese_ci    |
     3 |
  | sjis     | Shift-JIS Japanese              | sjis_japanese_ci    |
     2 |
  ```

```
| hebrew   | ISO 8859-8 Hebrew               | hebrew_general_ci   |
  1 |
| tis620   | TIS620 Thai                     | tis620_thai_ci      |
  1 |
| euckr    | EUC-KR Korean                   | euckr_korean_ci     |
  2 |
| koi8u    | KOI8-U Ukrainian                | koi8u_general_ci    |
  1 |
| gb2312   | GB2312 Simplified Chinese       | gb2312_chinese_ci   |
  2 |
| greek    | ISO 8859-7 Greek                | greek_general_ci    |
  1 |
| cp1250   | Windows Central European        | cp1250_general_ci   |
  1 |
| gbk      | GBK Simplified Chinese          | gbk_chinese_ci      |
  2 |
| latin5   | ISO 8859-9 Turkish              | latin5_turkish_ci   |
  1 |
| armscii8 | ARMSCII-8 Armenian              | armscii8_general_ci |
  1 |
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |
  3 |
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |
  2 |
| cp866    | DOS Russian                     | cp866_general_ci    |
  1 |
| keybcs2  | DOS Kamenicky Czech-Slovak      | keybcs2_general_ci  |
  1 |
| macce    | Mac Central European            | macce_general_ci    |
  1 |
| macroman | Mac West European               | macroman_general_ci |
  1 |
| cp852    | DOS Central European            | cp852_general_ci    |
  1 |
| latin7   | ISO 8859-13 Baltic              | latin7_general_ci   |
  1 |
| utf8mb4  | UTF-8 Unicode                   | utf8mb4_general_ci  |
  4 |
| cp1251   | Windows Cyrillic                | cp1251_general_ci   |
  1 |
| utf16    | UTF-16 Unicode                  | utf16_general_ci    |
  4 |
| utf16le  | UTF-16LE Unicode                | utf16le_general_ci  |
  4 |
| cp1256   | Windows Arabic                  | cp1256_general_ci   |
  1 |
| cp1257   | Windows Baltic                  | cp1257_general_ci   |
  1 |
| utf32    | UTF-32 Unicode                  | utf32_general_ci    |
  4 |
| binary   | Binary pseudo charset           | binary              |
  1 |
| geostd8  | GEOSTD8 Georgian                | geostd8_general_ci  |
  1 |
| cp932    | SJIS for Windows Japanese       | cp932_japanese_ci   |
  2 |
| eucjpms  | UJIS for Windows Japanese       | eucjpms_japanese_ci |
  3 |
```

```
| gb18030  | China National Standard GB18030 | gb18030_chinese_ci  |
   4 |
+----------+---------------------------------+---------------------+---
-----+
41 rows in set (0.00 sec)
```

如果要设置MySQL服务启动时默认使用的字符集，可以修改MySQL的配置并添加以下内容

```
[mysqld]
character-set-server=utf8
```

- 在创建表的时候，我们可以在右圆括号的后面通过`engine=xxx`来指定表的存储引擎，MySQL支持多种存储引擎，可以通过`show engines`命令进行查看。MySQL 5.5以后的版本默认使用的存储引擎是InnoDB，它正好也就是我们推荐大家使用的存储引擎（因为InnoDB更适合互联网应用对高并发、性能以及事务支持等方面的需求）。

```
show engines\G
```

```
*************************** 1. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign
keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it
disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
```

```
*************************** 6. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
9 rows in set (0.00 sec)
```

下面的表格对MySQL几种常用的数据引擎进行了简单的对比。

特性	**InnoDB**	**MRG_MYISAM**	**MEMORY**	**MyISAM**
存储限制	有	没有	有	有
事务	支持			
锁机制	行锁	表锁	表锁	表锁
B树索引	支持	支持	支持	支持
哈希索引			支持	
全文检索	支持（5.6+）			支持
集群索引	支持			
数据缓存	支持		支持	
索引缓存	支持	支持	支持	支持
数据可压缩				支持

特性	InnoDB	MRG_MYISAM	MEMORY	MyISAM
内存使用	高	低	中	低
存储空间使用	高	低		低
批量插入性能	低	高	高	高
是否支持外键	支持			

通过上面的比较我们可以了解到，InnoDB是唯一能够支持外键、事务以及行锁的存储引擎，所以我们之前说它更适合互联网应用，而且它也是较新的MySQL版本中默认使用的存储引擎。

- 在定义表结构为每个字段选择数据类型时，如果不清楚哪个数据类型更合适，可以通过MySQL的帮助系统来了解每种数据类型的特性、数据的长度和精度等相关信息。

```
? data types
```

```
You asked for help about help category: "Data Types"
For more information, type 'help <item>', where <item> is one of the
following
topics:
   AUTO_INCREMENT
   BIGINT
   BINARY
   BIT
   BLOB
   BLOB DATA TYPE
   BOOLEAN
   CHAR
   CHAR BYTE
   DATE
   DATETIME
   DEC
   DECIMAL
   DOUBLE
   DOUBLE PRECISION
   ENUM
   FLOAT
   INT
   INTEGER
   LONGBLOB
   LONGTEXT
   MEDIUMBLOB
   MEDIUMINT
   MEDIUMTEXT
   SET DATA TYPE
   SMALLINT
   TEXT
   TIME
   TIMESTAMP
   TINYBLOB
   TINYINT
   TINYTEXT
   VARBINARY
   VARCHAR
```

```
   YEAR DATA TYPE
```

```
? varchar
```

```
Name: 'VARCHAR'
Description:
[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE
collation_name]

A variable-length string. M represents the maximum column length in
characters. The range of M is 0 to 65,535. The effective maximum length
of a VARCHAR is subject to the maximum row size (65,535 bytes, which is
shared among all columns) and the character set used. For example, utf8
characters can require up to three bytes per character, so a VARCHAR
column that uses the utf8 character set can be declared to be a maximum
of 21,844 characters. See
http://dev.mysql.com/doc/refman/5.7/en/column-count-limit.html.

MySQL stores VARCHAR values as a 1-byte or 2-byte length prefix plus
data. The length prefix indicates the number of bytes in the value. A
VARCHAR column uses one length byte if values require no more than 255
bytes, two length bytes if values may require more than 255 bytes.

*Note*:

MySQL follows the standard SQL specification, and does not remove
trailing spaces from VARCHAR values.

VARCHAR is shorthand for CHARACTER VARYING. NATIONAL VARCHAR is the
standard SQL way to define that a VARCHAR column should use some
predefined character set. MySQL uses utf8 as this predefined character
set. http://dev.mysql.com/doc/refman/5.7/en/charset-national.html.
NVARCHAR is shorthand for NATIONAL VARCHAR.

URL: http://dev.mysql.com/doc/refman/5.7/en/string-type-overview.html
```

在数据类型的选择上，保存字符串数据通常都使用VARCHAR和CHAR两种类型，前者通常称为变长字符串，而后者通常称为定长字符串；对于InnoDB存储引擎，行存储格式没有区分固定长度和可变长度列，因此VARCHAR类型好CHAR类型没有本质区别，后者不一定比前者性能更好。如果要保存的很大字符串，可以使用TEXT类型；如果要保存很大的字节串，可以使用BLOB（二进制大对象）类型。在MySQL中，TEXT和BLOB又分别包括TEXT、MEDIUMTEXT、LONGTEXT和BLOB、MEDIUMBLOB、LONGBLOB三种不同的类型，它们主要的区别在于存储数据的最大大小不同。保存浮点数可以用FLOAT或DOUBLE类型，而保存定点数应该使用DECIMAL类型。如果要保存时间日期，DATETIME类型优于TIMESTAMP类型，因为前者能表示的时间日期范围更大。

2. DML

```
-- 插入学院数据
insert into tb_college (collname, collintro) values
('计算机学院', '创建于1956年是我国首批建立计算机专业。学院现有计算机科学与技术一级学科和网络空间安全一级学科博士学位授予权，其中计算机科学与技术一级学科具有博士后流动站。计算机科学与技术一级学科在2017年全国第四轮学科评估中评为A；2019 U.S.News全球计算机学科排名26名；ESI学科排名0.945‰，进入全球前1‰，位列第43位。'),
```

```
('外国语学院', '1998年浙江大学、杭州大学、浙江农业大学、浙江医科大学四校合并，成立新的浙江大学。1999年原浙江大学外语系、原杭州大学外国语学院、原杭州大学大外部、原浙江农业大学公外部、原浙江医科大学外语教学部合并，成立浙江大学外国语学院。2003年学院更名为浙江大学外国语言文化与国际交流学院。'),
('经济管理学院', '四川大学经济学院历史悠久、传承厚重，其前身是创办于1905年的四川大学经济科,距今已有100多年的历史。已故著名经济学家彭迪先、张与九、蒋学模、胡寄窗、陶大镛、胡代光，以及当代著名学者刘诗白等曾先后在此任教或学习。在长期的办学过程中，学院坚持以马克思主义的立场、观点、方法为指导，围绕建设世界一流经济学院的奋斗目标，做实“两个伟大”深度融合，不断提高党的建设质量与科学推进一流事业深度融合。');

-- 插入学生数据
insert into tb_student (stuid, stuname, stusex, stubirth, stuaddr, collid)
values
(1001, '杨逍', 1, '1990-3-4', '四川成都', 1),
(1002, '任我行', 1, '1992-2-2', '湖南长沙', 1),
(1033, '王语嫣', 0, '1989-12-3', '四川成都', 1),
(1572, '岳不群', 1, '1993-7-19', '陕西咸阳', 1),
(1378, '纪嫣然', 0, '1995-8-12', '四川绵阳', 1),
(1954, '林平之', 1, '1994-9-20', '福建莆田', 1),
(2035, '东方不败', 1, '1988-6-30', null, 2),
(3011, '林震南', 1, '1985-12-12', '福建莆田', 3),
(3755, '项少龙', 1, '1993-1-25', null, 3),
(3923, '杨不悔', 0, '1985-4-17', '四川成都', 3),
(4040, '隔壁老王', 1, '1989-1-1', '四川成都', 2);

-- 删除学生数据
delete from tb_student where stuid=4040;

-- 更新学生数据
update tb_student set stuname='杨过', stuaddr='湖南长沙' where stuid=1001;

-- 插入老师数据
insert into tb_teacher (teaid, teaname, teatitle, collid) values
(1122, '张三丰', '教授', 1),
(1133, '宋远桥', '副教授', 1),
(1144, '杨逍', '副教授', 1),
(2255, '范遥', '副教授', 2),
(3366, '韦一笑', '讲师', 3);

-- 插入课程数据
insert into tb_course (couid, couname, coucredit, teaid) values
(1111, 'Python程序设计', 3, 1122),
(2222, 'Web前端开发', 2, 1122),
(3333, '操作系统', 4, 1122),
(4444, '计算机网络', 2, 1133),
(5555, '编译原理', 4, 1144),
(6666, '算法和数据结构', 3, 1144),
(7777, '经贸法语', 3, 2255),
(8888, '成本会计', 2, 3366),
(9999, '审计学', 3, 3366);

-- 插入选课数据
insert into tb_record (sid, cid, seldate, score) values
(1001, 1111, '2017-09-01', 95),
(1001, 2222, '2017-09-01', 87.5),
(1001, 3333, '2017-09-01', 100),
(1001, 4444, '2018-09-03', null),
(1001, 6666, '2017-09-02', 100),
```

```
(1002, 1111, '2017-09-03', 65),
(1002, 5555, '2017-09-01', 42),
(1033, 1111, '2017-09-03', 92.5),
(1033, 4444, '2017-09-01', 78),
(1033, 5555, '2017-09-01', 82.5),
(1572, 1111, '2017-09-02', 78),
(1378, 1111, '2017-09-05', 82),
(1378, 7777, '2017-09-02', 65.5),
(2035, 7777, '2018-09-03', 88),
(2035, 9999, default, null),
(3755, 1111, default, null),
(3755, 8888, default, null),
(3755, 9999, '2017-09-01', 92);
```

```
-- 查询所有学生信息
select * from tb_student;

-- 查询所有课程名称及学分(投影和别名)
select couname, coucredit from tb_course;
select couname as 课程名称, coucredit as 学分 from tb_course;

-- 查询所有学生的姓名和性别(条件运算)
select stuname as 姓名, case stusex when 1 then '男' else '女' end as 性别
from tb_student;
select stuname as 姓名, if(stusex, '男', '女') as 性别 from tb_student;

-- 查询所有女学生的姓名和出生日期(筛选)
select stuname, stubirth from tb_student where stusex=0;

-- 查询所有80后学生的姓名、性别和出生日期(筛选)
select stuname, stusex, stubirth from tb_student where stubirth>='1980-1-1'
and stubirth<='1989-12-31';
select stuname, stusex, stubirth from tb_student where stubirth between
'1980-1-1' and '1989-12-31';

-- 查询姓"杨"的学生姓名和性别(模糊)
select stuname, stusex from tb_student where stuname like '杨%';

-- 查询姓"杨"名字两个字的学生姓名和性别(模糊)
select stuname, stusex from tb_student where stuname like '杨_';

-- 查询姓"杨"名字三个字的学生姓名和性别(模糊)
select stuname, stusex from tb_student where stuname like '杨__';

-- 查询名字中有"不"字或"嫣"字的学生的姓名(模糊)
select stuname, stusex from tb_student where stuname like '%不%' or stuname
like '%嫣%';

-- 查询没有录入家庭住址的学生姓名(空值)
select stuname from tb_student where stuaddr is null;

-- 查询录入了家庭住址的学生姓名(空值)
select stuname from tb_student where stuaddr is not null;

-- 查询学生选课的所有日期(去重)
select distinct seldate from tb_record;
```

```sql
-- 查询学生的家庭住址(去重)
select distinct stuaddr from tb_student where stuaddr is not null;

-- 查询男学生的姓名和生日按年龄从大到小排列(排序)
select stuname as 姓名, datediff(curdate(), stubirth) div 365 as 年龄 from
tb_student where stusex=1 order by 年龄 desc;

-- 查询年龄最大的学生的出生日期(聚合函数)
select min(stubirth) from tb_student;

-- 查询年龄最小的学生的出生日期(聚合函数)
select max(stubirth) from tb_student;

-- 查询男女学生的人数(分组和聚合函数)
select stusex, count(*) from tb_student group by stusex;

-- 查询课程编号为1111的课程的平均成绩(筛选和聚合函数)
select avg(score) from tb_record where cid=1111;

-- 查询学号为1001的学生所有课程的平均分(筛选和聚合函数)
select avg(score) from tb_record where sid=1001;

-- 查询每个学生的学号和平均成绩(分组和聚合函数)
select sid as 学号, avg(score) as 平均分 from tb_record group by sid;

-- 查询平均成绩大于等于90分的学生的学号和平均成绩
-- 分组以前的筛选使用where子句 / 分组以后的筛选使用having子句
select sid as 学号, avg(score) as 平均分 from tb_record group by sid having 平
均分>=90;

-- 查询年龄最大的学生的姓名(子查询/嵌套的查询)
select stuname from tb_student where stubirth=( select min(stubirth) from
tb_student );

-- 查询年龄最大的学生姓名和年龄(子查询+运算)
select stuname as 姓名, datediff(curdate(), stubirth) div 365 as 年龄 from
tb_student where stubirth=( select min(stubirth) from tb_student );

-- 查询选了两门以上的课程的学生姓名(子查询/分组条件/集合运算)
select stuname from tb_student where stuid in ( select stuid from tb_record
group by stuid having count(stuid)>2 );

-- 查询学生姓名、课程名称以及成绩(连接查询)
select stuname, couname, score from tb_student t1, tb_course t2, tb_record
t3 where stuid=sid and couid=cid and score is not null;

-- 查询学生姓名、课程名称以及成绩按成绩从高到低查询第11-15条记录(内连接+分页)
select stuname, couname, score from tb_student inner join tb_record on
stuid=sid inner join tb_course on couid=cid where score is not null order by
score desc limit 5 offset 10;

select stuname, couname, score from tb_student inner join tb_record on
stuid=sid inner join tb_course on couid=cid where score is not null order by
score desc limit 10, 5;

-- 查询选课学生的姓名和平均成绩(子查询和连接查询)
select stuname, avgmark from tb_student, ( select sid, avg(score) as avgmark
from tb_record group by sid ) temp where stuid=sid;
```

```
select stuname, avgmark from tb_student inner join ( select sid, avg(score)
as avgmark from tb_record group by sid ) temp on stuid=sid;

-- 查询每个学生的姓名和选课数量(左外连接和子查询)
select stuname, ifnull(total, 0) from tb_student left outer join ( select
sid, count(sid) as total from tb_record group by sid ) temp on stuid=sid;
```

上面的DML有几个地方需要加以说明：

1. MySQL中支持多种类型的运算符，包括：算术运算符（+、-、*、/、%）、比较运算符（=、<>、<=>、<、<=、>、>=、BETWEEN...AND...、IN、IS NULL、IS NOT NULL、LIKE、RLIKE、REGEXP）、逻辑运算符（NOT、AND、OR、XOR）和位运算符（&、|、^、~、>>、<<），我们可以在DML中使用这些运算符处理数据。
2. 在查询数据时，可以在SELECT语句及其子句（如WHERE子句、ORDER BY子句、HAVING子句等）中使用函数，这些函数包括字符串函数、数值函数、时间日期函数、流程函数等，如下面的表格所示。

   常用字符串函数。

函数	功能
CONCAT	将多个字符串连接成一个字符串
FORMAT	将数值格式化成字符串并指定保留几位小数
FROM_BASE64 / TO_BASE64	BASE64解码/编码
BIN / OCT / HEX	将数值转换成二进制/八进制/十六进制字符串
LOCATE	在字符串中查找一个子串的位置
LEFT / RIGHT	返回一个字符串左边/右边指定长度的字符
LENGTH / CHAR_LENGTH	返回字符串的长度以字节/字符为单位
LOWER / UPPER	返回字符串的小写/大写形式
LPAD / RPAD	如果字符串的长度不足，在字符串左边/右边填充指定的字符
LTRIM / RTRIM	去掉字符串前面/后面的空格
ORD / CHAR	返回字符对应的编码/返回编码对应的字符
STRCMP	比较字符串，返回-1、0、1分别表示小于、等于、大于
SUBSTRING	返回字符串指定范围的子串

   常用数值函数。

函数	功能
ABS	返回一个数的绝度值
CEILING / FLOOR	返回一个数上取整/下取整的结果
CONV	将一个数从一种进制转换成另一种进制
CRC32	计算循环冗余校验码
EXP / LOG / LOG2 / LOG10	计算指数/对数
POW	求幂
RAND	返回[0,1)范围的随机数
ROUND	返回一个数四舍五入后的结果
SQRT	返回一个数的平方根
TRUNCATE	截断一个数到指定的精度
SIN / COS / TAN / COT / ASIN / ACOS / ATAN	三角函数

常用时间日期函数。

函数	功能
CURDATE / CURTIME / NOW	获取当前日期/时间/日期和时间
ADDDATE / SUBDATE	将两个日期表达式相加/相减并返回结果
DATE / TIME	从字符串中获取日期/时间
YEAR / MONTH / DAY	从日期中获取年/月/日
HOUR / MINUTE / SECOND	从时间中获取时/分/秒
DATEDIFF / TIMEDIFF	返回两个时间日期表达式相差多少天/小时
MAKEDATE / MAKETIME	制造一个日期/时间

常用流程函数。

函数	功能
IF	根据条件是否成立返回不同的值
IFNULL	如果为NULL则返回指定的值否则就返回本身
NULLIF	两个表达式相等就返回NULL否则返回第一个表达式的值

其他常用函数。

函数	功能
MD5 / SHA1 / SHA2	返回字符串对应的哈希摘要
CHARSET / COLLATION	返回字符集/校对规则
USER / CURRENT_USER	返回当前用户
DATABASE	返回当前数据库名
VERSION	返回当前数据库版本
FOUND_ROWS / ROW_COUNT	返回查询到的行数/受影响的行数
LAST_INSERT_ID	返回最后一个自增主键的值
UUID / UUID_SHORT	返回全局唯一标识符

3. DCL

```
-- 创建可以远程登录的root账号并为其指定口令
create user 'root'@'%' identified by '123456';

-- 为远程登录的root账号授权操作所有数据库所有对象的所有权限并允许其将权限再次赋予其他用户
grant all privileges on *.* to 'root'@'%' with grant option;

-- 创建名为hellokitty的用户并为其指定口令
create user 'hellokitty'@'%' identified by '123123';

-- 将对school数据库所有对象的所有操作权限授予hellokitty
grant all privileges on school.* to 'hellokitty'@'%';

-- 召回hellokitty对school数据库所有对象的insert/delete/update权限
revoke insert, delete, update on school.* from 'hellokitty'@'%';
```

> 说明：创建一个可以允许任意主机登录并且具有超级管理员权限的用户在现实中并不是一个明智的决定，因为一旦该账号的口令泄露或者被破解，数据库将会面临灾难级的风险。

索引

索引是关系型数据库中用来提升查询性能最为重要的手段。关系型数据库中的索引就像一本书的目录，我们可以想象一下，如果要从一本书中找出某个知识点，但是这本书没有目录，这将是意见多么可怕的事情（我们估计得一篇一篇的翻下去，才能确定这个知识点到底在什么位置）。创建索引虽然会带来存储空间上的开销，就像一本书的目录会占用一部分的篇幅一样，但是在牺牲空间后换来的查询时间的减少也是非常显著的。

MySQL中，所有数据类型的列都可以被索引，常用的存储引擎InnoDB和MyISAM能支持每个表创建16个索引。InnoDB和MyISAM使用的索引其底层算法是B-tree（B树），B-tree是一种自平衡的树，类似于平衡二叉排序树，能够保持数据有序。这种数据结构能够让查找数据、顺序访问、插入数据及删除的操作都在对数时间内完成。

接下来我们通过一个简单的例子来说明索引的意义，比如我们要根据学生的姓名来查找学生，这个场景在实际开发中应该经常遇到，就跟通过商品名称查找商品道理是一样的。我们可以使用MySQL的`explain`关键字来查看SQL的执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb_student
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 11
     filtered: 10.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

在上面的SQL执行计划中，有几项值得我们关注：

1. type：MySQL在表中找到满足条件的行的方式，也称为访问类型，包括：ALL（全表扫描）、index（索引全扫描）、range（索引范围扫描）、ref（非唯一索引扫描）、eq_ref（唯一索引扫描）、const/system、NULL。在所有的访问类型中，很显然ALL是性能最差的，它代表了全表扫描是指要扫描表中的每一行才能找到匹配的行。
2. possible_keys：MySQL可以选择的索引，但是**有可能不会使用。**
3. key：MySQL真正使用的索引。
4. rows：执行查询需要扫描的行数，这是一个**预估值。**

从上面的执行计划可以看出，当我们通过学生名字查询学生时实际上是进行了全表扫描，不言而喻这个查询性能肯定是非常糟糕的，尤其是在表中的行很多的时候。如果我们需要经常通过学生姓名来查询学生，那么就应该在学生姓名对应的列上创建索引，通过索引来加速查询。

```
create index idx_student_name on tb_student(stuname);
```

再次查看刚才的SQL对应的执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb_student
   partitions: NULL
         type: ref
possible_keys: idx_student_name
          key: idx_student_name
      key_len: 62
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

可以注意到，在对学生姓名创建索引后，刚才的查询已经不是全表扫描而是基于索引的查询，而且扫描的行只有唯一的一行，这显然大大的提升了查询的性能。MySQL中还允许创建前缀索引，即对索引字段的前N个字符创建索引，这样的话可以减少索引占用的空间（但节省了空间很有可能会浪费时间，**时间和空间是不可调和的矛盾**），如下所示。

```
create index idx_student_name_1 on tb_student(stuname(1));
```

上面的索引相当于是根据学生姓名的第一个字来创建的索引，我们再看看SQL执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb_student
   partitions: NULL
         type: ref
possible_keys: idx_student_name
          key: idx_student_name
      key_len: 5
          ref: const
         rows: 2
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

不知道大家是否注意到，这一次扫描的行变成了2行，因为学生表中有两个姓“林”的学生，我们只用姓名的第一个字作为索引的话，在查询时通过索引就会找到这两行。

如果要删除索引，可以使用下面的SQL。

```
alter table tb_student drop index idx_student_name;
```

或者

```
drop index idx_student_name on tb_student;
```

我们简单的为大家总结一下索引的设计原则：

1. **最适合**索引的列是出现在**WHERE子句**和连接子句中的列。
2. 索引列的基数越大（取值多重复值少），索引的效果就越好。
3. 使用**前缀索引**可以减少索引占用的空间，内存中可以缓存更多的索引。
4. **索引不是越多越好**，虽然索引加速了读操作（查询），但是写操作（增、删、改）都会变得更慢，因为数据的变化会导致索引的更新，就如同书籍章节的增删需要更新目录一样。
5. 使用InnoDB存储引擎时，表的普通索引都会保存主键的值，所以**主键要尽可能选择较短的数据类型**，这样可以有效的减少索引占用的空间，利用提升索引的缓存效果。

最后，还有一点需要说明，InnoDB使用的B-tree索引，数值类型的列除了等值判断时索引会生效之外，使用>、<、>=、<=、BETWEEN...AND... 、<>时，索引仍然生效；对于字符串类型的列，如果使用不以通配符开头的模糊查询，索引也是起作用的，但是其他的情况会导致索引失效，这就意味着很有可能会做全表查询。

视图

视图是关系型数据库中将一组查询指令构成的结果集组合成可查询的数据表的对象。简单的说，视图就是虚拟的表，但与数据表不同的是，数据表是一种实体结构，而视图是一种虚拟结构，你也可以将视图理解为保存在数据库中被赋予名字的SQL语句。

使用视图可以获得以下好处：

1. 可以将实体数据表隐藏起来，让外部程序无法得知实际的数据结构，让访问者可以使用表的组成部分而不是整个表，降低数据库被攻击的风险。
2. 在大多数的情况下视图是只读的（更新视图的操作通常都有诸多的限制），外部程序无法直接透过视图修改数据。
3. 重用SQL语句，将高度复杂的查询包装在视图表中，直接访问该视图即可取出需要的数据；也可以将视图视为数据表进行连接查询。
4. 视图可以返回与实体数据表不同格式的数据，

创建视图。

```
create view vw_score
as
    select sid, round(avg(score), 1) as avgscore from tb_record group by sid;

create view vw_student_score
as
    select stuname, avgscore
        from tb_student, vw_score
        where stuid=sid;
```

> **提示**：因为视图不包含数据，所以每次使用视图时，都必须执行查询以获得数据，如果你使用了连接查询、嵌套查询创建了较为复杂的视图，你可能会发现查询性能下降得很厉害。因此，在使用复杂的视图前，应该进行测试以确保其性能能够满足应用的需求。

使用视图。

```
select stuname, avgscore from vw_student_score order by avgscore desc;
```

```
+--------------+----------+
| stuname      | avgscore |
+--------------+----------+
| 杨过         |     95.6 |
| 任我行       |     53.5 |
| 王语嫣       |     84.3 |
| 纪嫣然       |     73.8 |
| 岳不群       |     78.0 |
| 东方不败     |     88.0 |
| 项少龙       |     92.0 |
+--------------+----------+
```

既然视图是一张虚拟的表，那么视图的中的数据可以更新吗？视图的可更新性要视具体情况而定，以下类型的视图是不能更新的：

1. 使用了聚合函数（SUM、MIN、MAX、AVG、COUNT等）、DISTINCT、GROUP BY、HAVING、UNION或者UNION ALL的视图。
2. SELECT中包含了子查询的视图。
3. FROM子句中包含了一个不能更新的视图的视图。
4. WHERE子句的子查询引用了FROM子句中的表的视图。

删除视图。

```
drop view vw_student_score;
```

> **说明**：如果希望更新视图，可以先用上面的命令删除视图，也可以通过`create or replace view`来更新视图。

视图的规则和限制。

1. 视图可以嵌套，可以利用从其他视图中检索的数据来构造一个新的视图。视图也可以和表一起使用。
2. 创建视图时可以使用`order by`子句，但如果从视图中检索数据时也使用了`order by`，那么该视图中原先的`order by`会被覆盖。
3. 视图无法使用索引，也不会激发触发器（实际开发中因为性能等各方面的考虑，通常不建议使用触发器，所以我们也不对这个概念进行介绍）的执行。

存储过程

存储过程是事先编译好存储在数据库中的一组SQL的集合，调用存储过程可以简化应用程序开发人员的工作，减少与数据库服务器之间的通信，对于提升数据操作的性能也是有帮助的。其实迄今为止，我们使用的SQL语句都是针对一个或多个表的单条语句，但在实际开发中经常会遇到某个操作需要多条SQL语句才能完成的情况。例如，电商网站在受理用户订单时，需要做以下一系列的处理。

1. 通过查询来核对库存中是否有对应的物品以及库存是否充足。
2. 如果库存有物品，需要锁定库存以确保这些物品不再卖给别人， 并且要减少可用的物品数量以反映正确的库存量。
3. 如果库存不足，可能需要进一步与供应商进行交互或者至少产生一条系统提示消息。
4. 不管受理订单是否成功，都需要产生流水记录，而且需要给对应的用户产生一条通知信息。

我们可以通过存储过程将复杂的操作封装起来，这样不仅有助于保证数据的一致性，而且将来如果业务发生了变动，只需要调整和修改存储过程即可。对于调用存储过程的用户来说，存储过程并没有暴露数据表的细节，而且执行存储过程比一条条的执行一组SQL要快得多。

下面的存储过程实现了查询某门课程的最高分、最低分和平均分。

```
drop procedure if exists sp_score_by_cid;

delimiter $$

create procedure sp_score_by_cid(
    courseId int,
    out maxScore decimal(4,1),
    out minScore decimal(4,1),
    out avgScore decimal(4,1)
)
begin
    select max(score) into maxScore from tb_record
        where cid=courseId;
    select min(score) into minScore from tb_record
        where cid=courseId;
    select avg(score) into avgScore from tb_record
        where cid=courseId;
end $$

delimiter ;

call sp_score_by_cid(1111, @a, @b, @c);
select @a, @b, @c;
```

> 说明：在定义存储过程时，因为可能需要书写多条SQL，而分隔这些SQL需要使用分号作为分隔符，如果这个时候，仍然用分号表示整段代码结束，那么定义存储过程的SQL就会出现错误，所以上面我们用`delimiter $$`将整段代码结束的标记定义为`$$`，那么代码中的分号将不再表示整段代码的结束，需要马上执行，整段代码在遇到`end $$`时才输入完成并执行。在定义完存储过程后，通过`delimiter ;`将结束符重新改回成分号。

上面定义的存储过程有四个参数，其中第一个参数是输入参数，代表课程的编号，后面的参数都是输出参数，因为存储过程不能定义返回值，只能通过输出参数将执行结果带出，定义输出参数的关键字是`out`，默认情况下参数都是输入参数。

调用存储过程。

```
call sp_score_by_cid(1111, @a, @b, @c);
```

获取输出参数的值。

```
select @a as 最高分, @b as 最低分, @c as 平均分;
```

删除存储过程。

```
drop procedure sp_score_by_cid;
```

在存储过程中，我们可以定义变量、条件，可以使用分支和循环语句，可以通过游标操作查询结果，还可以使用事件调度器，这些内容我们暂时不在此处进行介绍。虽然我们说了很多存储过程的好处，但是在实际开发中，如果过度的使用存储过程，将大量复杂的运算放到存储过程中，也会导致占用数据库服务器的CPU资源，造成数据库服务器承受巨大的压力。为此，我们一般会将复杂的运算和处理交给应用服务器，因为很容易部署多台应用服务器来分摊这些压力。

几个重要的概念

范式理论 - 设计二维表的指导思想

1. 第一范式：数据表的每个列的值域都是由原子值组成的，不能够再分割。
2. 第二范式：数据表里的所有数据都要和该数据表的键（主键与候选键）有完全依赖关系。
3. 第三范式：所有非键属性都只和候选键有相关性，也就是说非键属性之间应该是独立无关的。

数据完整性

1. 实体完整性 - 每个实体都是独一无二的
    - 主键（primary key） / 唯一约束 / 唯一索引（unique）
2. 引用完整性（参照完整性）- 关系中不允许引用不存在的实体
    - 外键（foreign key）
3. 域完整性 - 数据是有效的
    - 数据类型及长度
    - 非空约束（not null）
    - 默认值约束（default）
    - 检查约束（check）

        > 说明：在MySQL数据库中，检查约束并不起作用。

数据一致性

1. 事务：一系列对数据库进行读/写的操作，这些操作要么全都成功，要么全都失败。
2. 事务的ACID特性
    - 原子性：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行
    - 一致性：事务应确保数据库的状态从一个一致状态转变为另一个一致状态
    - 隔离性：多个事务并发执行时，一个事务的执行不应影响其他事务的执行
    - 持久性：已被提交的事务对数据库的修改应该永久保存在数据库中
3. MySQL中的事务操作
    - 开启事务环境

      ```
      start transaction
      ```

      或

      ```
      begin
      ```

    - 提交事务

      ```
      commit
      ```

    - 回滚事务

      ```
      rollback
      ```

其他内容

大家应该能够想到，关于MySQL的知识肯定远远不止上面列出的这些，比如MySQL的性能优化、管理和维护MySQL的相关工具、MySQL数据的备份和恢复、监控MySQL、部署高可用架构等问题我们在这里都没有进行讨论。当然，这些内容也都是跟项目开发密切相关的，我们就留到后续的章节中再续点进行讲解。

Python数据库编程

我们用如下所示的数据库来演示在Python中如何访问MySQL数据库。

```
drop database if exists hrs;
create database hrs default charset utf8;

use hrs;

drop table if exists tb_emp;
drop table if exists tb_dept;

create table tb_dept
(
dno   int not null comment '编号',
dname varchar(10) not null comment '名称',
dloc  varchar(20) not null comment '所在地',
primary key (dno)
);

insert into tb_dept values
```

```
    (10, '会计部', '北京'),
    (20, '研发部', '成都'),
    (30, '销售部', '重庆'),
    (40, '运维部', '深圳');

create table tb_emp
(
eno   int not null comment '员工编号',
ename varchar(20) not null comment '员工姓名',
job   varchar(20) not null comment '员工职位',
mgr   int comment '主管编号',
sal   int not null comment '员工月薪',
comm  int comment '每月补贴',
dno   int comment '所在部门编号',
primary key (eno)
);

alter table tb_emp add constraint fk_emp_dno foreign key (dno) references
tb_dept (dno);

insert into tb_emp values
    (7800, '张三丰', '总裁', null, 9000, 1200, 20),
    (2056, '乔峰', '分析师', 7800, 5000, 1500, 20),
    (3088, '李莫愁', '设计师', 2056, 3500, 800, 20),
    (3211, '张无忌', '程序员', 2056, 3200, null, 20),
    (3233, '丘处机', '程序员', 2056, 3400, null, 20),
    (3251, '张翠山', '程序员', 2056, 4000, null, 20),
    (5566, '宋远桥', '会计师', 7800, 4000, 1000, 10),
    (5234, '郭靖', '出纳', 5566, 2000, null, 10),
    (3344, '黄蓉', '销售主管', 7800, 3000, 800, 30),
    (1359, '胡一刀', '销售员', 3344, 1800, 200, 30),
    (4466, '苗人凤', '销售员', 3344, 2500, null, 30),
    (3244, '欧阳锋', '程序员', 3088, 3200, null, 20),
    (3577, '杨过', '会计', 5566, 2200, null, 10),
    (3588, '朱九真', '会计', 5566, 2500, null, 10);
```

在Python 3中，我们通常使用纯Python的三方库PyMySQL来访问MySQL数据库，它应该是目前Python操作MySQL数据库最好的选择。

1. 安装PyMySQL。

```
pip install pymysql
```

2. 添加一个部门。

```
import pymysql


def main():
    no = int(input('编号: '))
    name = input('名字: ')
    loc = input('所在地: ')
    # 1. 创建数据库连接对象
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='yourname', password='yourpass')
```

```
    try:
        # 2. 通过连接对象获取游标
        with con.cursor() as cursor:
            # 3. 通过游标执行SQL并获得执行结果
            result = cursor.execute(
                'insert into tb_dept values (%s, %s, %s)',
                (no, name, loc)
            )
        if result == 1:
            print('添加成功!')
        # 4. 操作成功提交事务
        con.commit()
    finally:
        # 5. 关闭连接释放资源
        con.close()


if __name__ == '__main__':
    main()
```

3. 删除一个部门。

```
import pymysql


def main():
    no = int(input('编号: '))
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='yourname', password='yourpass',
                          autocommit=True)
    try:
        with con.cursor() as cursor:
            result = cursor.execute(
                'delete from tb_dept where dno=%s',
                (no, )
            )
        if result == 1:
            print('删除成功!')
    finally:
        con.close()


if __name__ == '__main__':
    main()
```

> 说明：如果不希望每次SQL操作之后手动提交或回滚事务，可以像上面的代码那样，在创建连接的时候多加一个名为`autocommit`的参数并将它的值设置为`True`，表示每次执行SQL之后自动提交。如果程序中不需要使用事务环境也不希望手动的提交或回滚就可以这么做。

4. 更新一个部门。

```
import pymysql


def main():
```

```
    no = int(input('编号: '))
    name = input('名字: ')
    loc = input('所在地: ')
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='yourname', password='yourpass',
                          autocommit=True)
    try:
        with con.cursor() as cursor:
            result = cursor.execute(
                'update tb_dept set dname=%s, dloc=%s where dno=%s',
                (name, loc, no)
            )
        if result == 1:
            print('更新成功!')
    finally:
        con.close()


if __name__ == '__main__':
    main()
```

5. 查询所有部门。

```
import pymysql
from pymysql.cursors import DictCursor


def main():
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='yourname', password='yourpass')
    try:
        with con.cursor(cursor=DictCursor) as cursor:
            cursor.execute('select dno as no, dname as name, dloc as loc
from tb_dept')
            results = cursor.fetchall()
            print(results)
            print('编号\t名称\t\t所在地')
            for dept in results:
                print(dept['no'], end='\t')
                print(dept['name'], end='\t')
                print(dept['loc'])
    finally:
        con.close()


if __name__ == '__main__':
    main()
```

6. 分页查询员工信息。

```
import pymysql
from pymysql.cursors import DictCursor
```

```python
class Emp(object):

    def __init__(self, no, name, job, sal):
        self.no = no
        self.name = name
        self.job = job
        self.sal = sal

    def __str__(self):
        return f'\n编号：{self.no}\n姓名：{self.name}\n职位：{self.job}\n月薪：
{self.sal}\n'


def main():
    page = int(input('页码: '))
    size = int(input('大小: '))
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='yourname', password='yourpass')
    try:
        with con.cursor() as cursor:
            cursor.execute(
                'select eno as no, ename as name, job, sal from tb_emp limit
%s,%s',
                ((page - 1) * size, size)
            )
            for emp_tuple in cursor.fetchall():
                emp = Emp(*emp_tuple)
                print(emp)
    finally:
        con.close()


if __name__ == '__main__':
    main()
```