数据库技术与应用

第二章 MYSQL数据库与表的创建

本章主要内容

- 2.1 MYSQL数据库
- 2.2 MYSQL数据类型
- 2.3 表的创建
- 2.4 表结构的修改
- 2.5 表的删除
- 2.6 向表中添加数据
- 2.7 表的索引
- 2.8 简单数据库备份与数据导入导出

2.1 MYSQL数据库

任何一个数据库，都是由各种数据库对象组织而成的，比如表、索引、视图等等。而操作的第一步，就是要创建数据库。本节将对下面两个问题进行讲解。

- 2.1.1 MYSQL模式简介
- 2.1.2 MYSQL数据库的创建与删除

2.1.1 MYSQL模式简介

与其他数据库不同，MYSQL可以运行在不同的SQL MODE模式下。SQL Mode定义了MYSQL应支持的SQL语法、数据校验等，便于在不同的环境中使用MYSQL。

- SQL Mode解决以下问题
  - 完成不同严格程度的数据校验，有效保障数据准确性。
  - 将SQL Mode设置为ANSI模式，保证大多数SQL符合标准的SQL语法，应用在不同的数据库之间迁移时，不需要对业务SQL进行较大的修改。
  - 设置SQL Mode可以使MYSQL上的数据更方便的迁移到目标数据库中。

2.1.1 MYSQL模式简介

v MYSQL中常用的SQL Mode

Sql_mode值	描述
ANSI	等同于REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE和ANSI组合，这种模式使语法 和行为更符合标准的sql。
STRICT_TRANS_TABLES	适用于事务表和非事务表，属严格模式，不允许非法日期、或超过字段长度插入数据，插入不正确的值给出错误而不是警告。
TRADITIONAL	等同于STRICT_TRANS_TABLES、STRICT_ALL_TABLES、NO_ZERO_IN_DATE、NO_ZERO_DATE、ERROR_FOR_DIVISION_BY_ZERO、TRADITIONAL、NO_AUTO_CREATE_USER的组合，属于严格模式，适用于事务表和非事务表，在事务表时，只要出现错误就立即回滚

实际上，第一列模式都是一些原子模式的组合，类似于角色和权限的关系。

MYSQL模式操作实例

```
mysql> select @@sql_mode;
+-----------------------------------------------------------+
| @@sql_mode                                                |
+-----------------------------------------------------------+
| REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI |
+-----------------------------------------------------------+
```

```
mysql> desc t;
+--------+------------------+------+-----+---------+----------------+
| Field  | Type             | Null | Key | Default | Extra          |
+--------+------------------+------+-----+---------+----------------+
| userno | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name   | varchar(20)      | YES  |     | NULL    |                |
| email  | varchar(40)      | YES  |     | NULL    |                |
+--------+------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

```
mysql> insert into t(name,email) values('1000000000200000000030000000000','zzhstudio@126.com');
Query OK, 1 row affected, 1 warning (0.03 sec)
```

```
mysql> select * from t;
+--------+----------------------+-------------------+
| userno | name                 | email             |
+--------+----------------------+-------------------+
|      1 | 1000000000200000000 | zzhstudio@126.com |
+--------+----------------------+-------------------+
1 row in set (0.00 sec)
```

MYSQL模式操作实例

```
mysql> set session sql_mode='STRICT_TRANS_TABLES';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select @@sql_mode;
+---------------------+
| @@sql_mode          |
+---------------------+
| STRICT_TRANS_TABLES |
+---------------------+
1 row in set (0.01 sec)
```

```
mysql> insert into t(name,email) values('1000000000200000000030000000000','zzhstudio@126.com');
ERROR 1406 (22001): Data too long for column 'name' at row 1
```

```
mysql> select * from t;
+--------+----------------------+--------------------+
| userno | name                 | email              |
+--------+----------------------+--------------------+
|      1 | 10000000002000000000 | zzhstudio@126.com  |
+--------+----------------------+--------------------+
1 row in set (0.00 sec)
```


2.1.2 MYSQL数据库的创建与删除

- MYSQL系统数据库说明：
  - Information_schema：存储系统的数据库对象信息，如用户表信息、列信息、权限信息、字符集信息、分区信息等。
  - cluster：存储系统的集群信息
  - mysql：存储系统的用户权限信息
  - Test:系统自动创建的测试数据库，任何用户都可以使用
- MYSQL数据库操作命令：
  - 创建数据库：create database dbname
  - 删除数据库：drop database dbname
  - 显示系统中所有数据库：show databases
  - 选择数据库：use dbname
  - 查看数据库中所有数据表：show tables

2.2 MYSQL数据类型

在开始创建表之前，必须了解表中字段可以选择的数据类型（即域定义），使得字段使用合理的数据类型，从而有效提高数据库的性能：存储空间小、查询速度快。

- 2.2.1 数值类型
- 2.2.2 日期时间类型
- 2.2.3 字符串类型

2.2.1 MYSQL中数值类型

类型	字节	最小值(带符号的/无符号的)	最大值(带符号的/无符号的)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615
FLOAT	4	+(-)1.175494351E-38	+(-)3.402823466E+38
DOUBLE	8	+(-)2.2250738585072014E-308	+(-)1.7976931348623157E-308
DECIMAL(M,D)	M+2	最大取值范围与DOUBLE相同，给定DECIMAL的有效值范围由M和D决定	
BIT(M)	1~8	BIT(1)	BIT(8)

2.2.1 MYSQL中数值类型-说明

- 整数类型支持INT(n)形式，用于指定整数值的显示宽度，如显示宽度小于n时从左侧填满宽度。显示宽度并不限制可以在列内保存的值的范围，也不限制超过列的指定宽度的值的显示。当结合可选扩展属性ZEROFILL使用时， 默认补充的空格用零代替。例如，对于声明为INT(5) ZEROFILL的列，值4检索为00004。
- 所有整数类型可以有一个可选(非标准)属性UNSIGNED。在列内只允许非负数和该列需要较大的上限数值范围时可以使用无符号值。如果为一个数值列指定ZEROFILL，MySQL自动为该列添加UNSIGNED属性。
- MYSQL小数有两种方式：浮点数和定点数。浮点数包括float、double，为近似数据类型。定点数只有decimal，用于保存必须为确切精度的值，例如货币数据。
- 浮点和定点类型也可以为UNSIGNED。同数类型，该属性防止负值保存到列中。然而，与整数类型不同的是，列值的上范围保持不变。
- 浮点数和定点数都可以使用数据名后加“(M,D)”形式。例如：salary DECIMAL(5,2)在该例子中，5是精度，2是标度。精度表示保存值的主要位数，标度表示小数点后面可以保存的位数。
- BIT数据类型可用来保存位(二进制数)字段值。如果为BIT(M)列分配的值的长度小于M位，在值的左边用0填充。例如，为BIT(6)列分配一个值b‘101’，其效果与分配b‘000101’相同。
- 整数类型有一个属性：AUTO_INCREMENT，用于定义自动增长字段。每增加一行增加1。一个表最多只能定义一个AUTO_INCREMENT列，应定义为NOT NULL，并定义为PRIMARY KEY或UNIQUE。

2.2.2 时间日期类型

列类型	字节	“零”值	最小值	最大值
DATETIME	4	'0000-00-00 00:00:00'	1000-01-01 00:00:00	9999-12-31 23:59:59
DATE	8	'0000-00-00'	1000-01-01	9999-12-31
TIMESTAMP	4	00000000000000	19700101080001	2038年的某个时刻
TIME	3	'00:00:00'	-838:59:59	838:59:59
YEAR	1	0000	1901	2155

- 每个时间类型有一个有效值范围和一个“零”值，当指定不合法的MySQL不能表示的值时使用“零”值。
- DATE用来表示“年月日”、DATETIME用来表示“年月日时分秒”、TIME用来表示“时分秒”。
- TIMESTAMP值显示格式为“YYYY-MM-DD HH:MM:SS”，用于需要经常插入或更新日期为当前系统时间的列。比如注册时间。
- YEAR只表示年份。
- DATETIME与TIMESTAMP的区别
  - DATETIME范围大、TIMESTAMP范围小
  - 表中第一个TIMESTAMP列自动设置为系统时间CURRENT_TIMESTAMP。其后则不管。
  - TIMESTAMP插入和查询都受当地时区的影响；DATETIME只反映当地时区，其他时区人查询时数据会有误差。NOW()。

2.2.3 字符串类型

列类型	存储需求
CHAR(*M*)	*M*个字节，0 <= *M* <= 255
VARCHAR(*M*)	*L*+1个字节，其中*L* <= *M* 且0 <= *M* <= 65535(参见下面的注释)
BINARY(*M*)	*M*个字节，0 <= *M* <= 255
VARBINARY(*M*)	*L*+1个字节，其中*L* <= *M* 且0 <= *M* <= 255
TINYBLOB，TINYTEXT	*L*+1个字节，其中*L* < 2^8
BLOB，TEXT	*L*+2个字节，其中*L* < 2^{16}
MEDIUMBLOB，MEDIUMTEXT	*L*+3个字节，其中*L* < 2^{24}
LONGBLOB，LONGTEXT	*L*+4个字节，其中*L* < 2^{32}
ENUM('*value1*','*value2*',...)	1或2个字节，取决于枚举值的个数(最多65,535个值)
SET('*value1*','*value2*',...)	1、2、3、4或者8个字节，取决于set成员的数目(最多64个成员

2.2.3 字符串类型

v CHAR和VARCHAR比较

* CHAR为定长、VARCHAR为变长。
* 保存CHAR值时，在右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格被删除掉。在存储或检索过程中不进行大小写转换。
* VARCHAR只保存需要的字符数，另加一个字节来记录长度。VARCHAR不进行填充。当值保存和检索时尾部的空格仍保留，符合标准SQL。

值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
''	' '	4个字节	''	1个字节
'ab'	'ab '	4个字节	'ab '	3个字节
'abcd'	'abcd'	4个字节	'abcd'	5个字节
'abcdefgh'	'abcd'	4个字节	'abcd'	5个字节

上表中最后一行的值只适用非严格模式时；如果MySQL运行在严格模式，超过列长度不的值不保存，并且会出现错误。

2.2.3 字符串类型

vBLOB和TEXT比较

* BLOB 为二进制字符串(字节字符串),用来保存二进制数据，如图片。TEXT字符字符串，保存较大文本，如文章。
* BLOB列没有字符集，并且排序和比较基于列值字节的数值值。TEXT列有一个字符集，并且根据字符集的 校对规则对值进行排序和比较。
* 在TEXT或BLOB列的存储或检索过程中，不存在大小写转换。
* 当保存或检索BLOB和TEXT列的值时不删除尾部空格。
* 对于BLOB和TEXT列的索引，必须指定索引前缀的长度。对于CHAR和VARCHAR，前缀长度是可选的。
* BLOB和TEXT列不能有默认值。
* 在执行大量的删除操作时，使用BLOB和TEXT会留下很大的“空洞”，从而影响插入数据的性能。可使用OPTIMIZE TABLE功能对表进行碎片整理。

2.2.3 字符串类型

ENUM枚举类型说明

* ENUM是一个字符串对象，其值来自表创建时在列规定中显式枚举的一列值。枚举最多可以有65,535个元素。
* 创建表时，ENUM成员值的尾部空格将自动被删除。
* 检索时，保存在ENUM列的值使用列定义中所使用的大小写来显示。
* 确定一个ENUM列的所有可能的值：SHOW COLUMNS FROM tbl_name
* 定义了一组ENUM枚举类型值后，在严格模式下，向该列中插入不在该枚举中的值时，系统禁止插入。在ANSI模式下，向该列插入空值。

```
mysql> create table enum_t (city enum('beijing','shanghai','shenzhen','chongqing'));
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> SHOW COLUMNS FROM enum_t;
+-------+-----------------------------------------------+------+-----+---------+-------+
| Field | Type                                          | Null | Key | Default | Extra |
+-------+-----------------------------------------------+------+-----+---------+-------+
| city  | enum('beijing','shanghai','shenzhen','chongqing') | YES  |     | NULL    |       |
+-------+-----------------------------------------------+------+-----+---------+-------+
1 row in set (0.01 sec)
```

2.3 字符串类型

v ENUM枚举类型说明

```
mysql> select @@sql_mode;
+-----------------------------------------------------------+
| @@sql_mode                                                |
+-----------------------------------------------------------+
| STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----------------------------------------------------------+
```

```
mysql> insert into enum_t values('beijing'),('chongqing'),('guiyang');
ERROR 1265 (01000): Data truncated for column 'city' at row 3
mysql> select * from enum_t;
Empty set (0.00 sec)
```

```
mysql> insert into enum_t values('beijing'),('chongqing');
Query OK, 2 rows affected (0.03 sec)
```

```
mysql> select * from enum_t;
+-----------+
| city      |
+-----------+
| beijing   |
| chongqing |
+-----------+
```

```
mysql> set session sql_mode="ANSI";
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> insert into enum_t values ('guiyang');
Query OK, 1 row affected, 1 warning (0.02 sec)

mysql> select * from enum_t;
+-----------+
| city      |
+-----------+
| beijing   |
| chongqing |
|           |
+-----------+
3 rows in set (0.00 sec)
```

2.2.3 字符串类型

v SET类型说明

* SET是一个字符串对象，可以有零或多个值，其值来自表创建时规定的允许的一列值。
* SET可以向列中插入多个定义的值，而ENUM只能插入一个定义的值。
* 指定包括多个SET成员的SET列值时各成员之间用逗号(‘,’)间隔开。这样SET成员值本身不能包含逗号。
* 创建表时，SET成员值的尾部空格将自动被删除。
* 对于包含多个SET元素的值，当插入值时元素所列的顺序并不重要。在值中一个给定的元素列了多少次也不重要。当以后检索该值时，值中的每个元素出现一次，根据表创建时指定的顺序列出元素。
* 在严格模式下，向该列中插入不在该集合中的值时，系统禁止插入。在ANSI模式下，向该列采用截断方式插入。

2.2.3 字符串类型

v SET类型说明

```
mysql> select @@sql_mode;
+-------------------------------------------------------------+
| @@sql_mode                                                  |
+-------------------------------------------------------------+
| REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI |
+-------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE set_t (col SET('a', 'b', 'c', 'd'));
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO set_t (col) VALUES ('a,d'), ('d,a'), ('a,d,a'), ('a,d,e');
Query OK, 4 rows affected, 1 warning (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 1
```

```
mysql> select * from set_t;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
4 rows in set (0.00 sec)
```

```
mysql> set session sql_mode='STRICT_TRANS_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> select @@sql_mode;
+--------------------+
| @@sql_mode         |
+--------------------+
| STRICT_TRANS_TABLES |
+--------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO set_t (col) VALUES ('a,d,e');
ERROR 1265 (01000): Data truncated for column 'col' at row 1
```

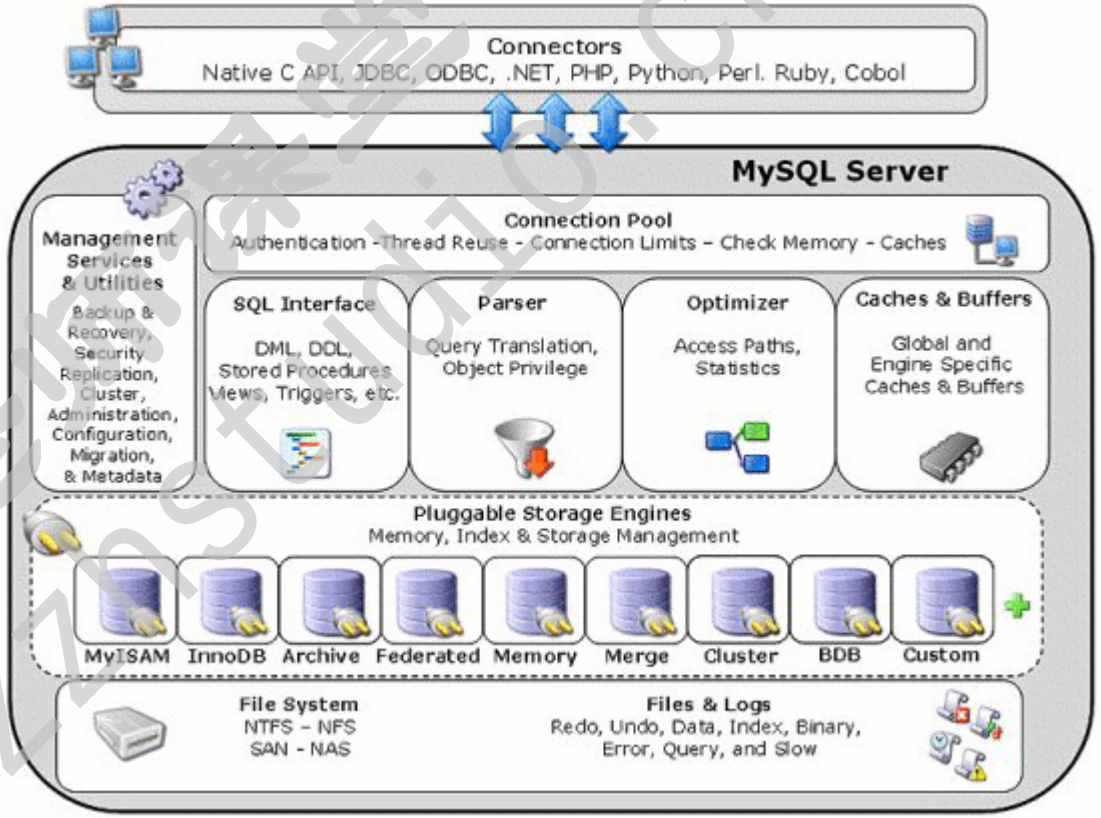
```
mysql> select * from set_t;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
4 rows in set (0.00 sec)
```

2.3 表的创建

- 2.3.1 MYSQL表的存储引擎
- 2.3.2 创建表
  - 2.3.2.1 SQL基本语法
  - 2.3.2.2 主键设置
  - 2.3.2.3 空值约束设置
  - 2.3.2.4 唯一性约束设置
  - 2.3.2.5 自动增长字段设置
  - 2.3.2.6 默认值设置
  - 2.3.2.7 自定义约束设置
  - 2.3.2.8 外键设置
- 2.3.3 查看表的结构

2.3.1 MYSQL表的存储引擎

MySQL插件式存储引擎是MySQL数据库服务器中的组件，负责为数据库执行实际的数据I/O操作，并能允许和强制执行面向特殊应用需求的特定特性集合。使用特殊存储引擎的主要优点在于，仅需提供特殊应用所需的特性，数据库中的系统开销较小，最终结果具有更有效和更高的数据库性能。这也是MySQL被始终视为具有高性能的原因之一，在行业标准基准方面，它能匹敌或击败专有的整体式数据库。



2.3.1 MYSQL表的存储引擎

v 各种存储引擎的特性

Feature	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	No	No	Yes	64TB	No	Yes
Transactions (commit, rollback, etc.)		✔		✔		
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				✔	✔	✔
Geospatial support	✔					
B-Tree indexes	✔	✔	✔	✔		✔
Hash indexes			✔	✔		✔
Full text search index	✔					
Clustered index				✔		
Data Caches			✔	✔		✔
Index Caches	✔		✔	✔		✔
Compressed data	✔				✔	
Encrypted data (via function)	✔	✔	✔	✔	✔	✔
Storage cost (space used)	Low	Low	N/A	High	Very Low	Low
Memory cost	Low	Low	Medium	High	Low	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database support						✔
Replication support	✔	✔	✔	✔	✔	✔
Foreign key support				✔		
Backup/Point-in-time recovery	✔	✔	✔	✔	✔	✔
Query cache support	✔	✔	✔	✔	✔	✔
Update Statistics for Data Dictionary	✔	✔	✔	✔	✔	✔

下面重点介绍最常用的4种存储引擎：MyISAM、InnoDB、Memory、Merge。

2.3.1 MYSQL表的存储引擎

MyISAM：MySQL默认存储引擎。MyISAM不支持事务、不支持外键，特点是访问速度快。 MyISAM是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。

- 每个MyISAM表在磁盘上存储为3个文件，文件名与表名相同，扩展名分别是：
  - .frm（存储表定义）
  - MYD（MYData,存储数据）
  - MYI(MYIndex，存储索引)。

数据文件和索引文件可以房子不同路径，平衡IO，获得更快访问速度。

- MyISAM类型表可能会损坏。应注意检查：
  - CHECK TABLE语句检查健康
  - REPAIR TABLE语句修复损坏

2.3.1 MYSQL表的存储引擎

InnoDB：用于事务处理应用程序，具有众多特性，包括ACID(Atomicity原子性、Consistency一致性、Isolation隔离性、Durability持续性)事务支持。相对MyISAM引擎，写处理效率低一些，且占用较多磁盘空间用于存储数据和索引。

- InnoDB是MYSQL中唯一支持外键的引擎。
- InnoDB存储表和索引有两种方式：
  - 共享表存储：表的结构存放在.frm文件中，数据和索引保存在innodb_data_home_dir和innodb_data_file_path定义的表空间中。
  - 多表空间存储：表的结构存放在.frm文件中，数据和索引单独保存在.idb中。如果是分区表，则每个分区对应单独的.ibd，文件名是“表名+分区名”。

2.3.1 MYSQL表的存储引擎

- Memory：所有数据保存在RAM（内存）中，一旦服务器关闭，表中的数据就会丢失。在需要快速查找引用和其他类似数据的环境下，可提供极快的访问。
- Merge：允许MySQL DBA或开发人员将一系列等同的MyISAM表以逻辑方式组合在一起，并作为一个对象引用它们。优点是可以突破单个MyISAM表大小的限制，并将不同的表分布在多个磁盘上，从而有效改善访问效率。对于诸如数据仓储等VLDB环境十分适合。

2.3.1 MYSQL表的存储引擎

- 查看当前默认存储引擎
  - Show variables like ‘table_type’;
- 查看当前数据库支持的存储引擎
  - Show engines;
  - Show variables like ‘have%’;
- 查看指定表的存储引擎
  - Show create table 表名;

```
mysql> Show variables like 'have%';
+----------------------+----------+
| Variable_name        | Value    |
+----------------------+----------+
| have_compress        | YES      |
| have_crypt           | NO       |
| have_csv             | YES      |
| have_dynamic_loading | YES      |
| have_geometry        | YES      |
| have_innodb          | YES      |
| have_ndbcluster      | NO       |
| have_openssl         | DISABLED |
| have_partitioning    | YES      |
| have_profiling       | YES      |
| have_query_cache     | YES      |
| have_rtree_keys      | YES      |
| have_ssl             | DISABLED |
| have_symlink         | YES      |
+----------------------+----------+
14 rows in set (0.01 sec)
```

```
mysql> show create table set_t;
+-------+-----------------------------------------
| Table | Create Table
+-------+-----------------------------------------
| set_t | CREATE TABLE `set_t` (
  `col` set('a','b','c','d') DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------
1 row in set (0.00 sec)
```

2.3 创建表

- 2.3.2.1 SQL基本语法
- 2.3.2.2 主键设置
- 2.3.2.3 空值约束设置
- 2.3.2.4 唯一性约束设置
- 2.3.2.5 自动增长字段设置
- 2.3.2.6 默认值设置
- 2.3.2.7 自定义约束设置
- 2.3.2.8 外键设置
- 2.3.2.9 查看表的结构

2.3.2.1 SQL基本语法

CREATE TABLE <表名>

（

<列名> <数据类型>[<列级完整性约束条件>]

[，<列名> <数据类型>[<列级完整性约束条件>]] …

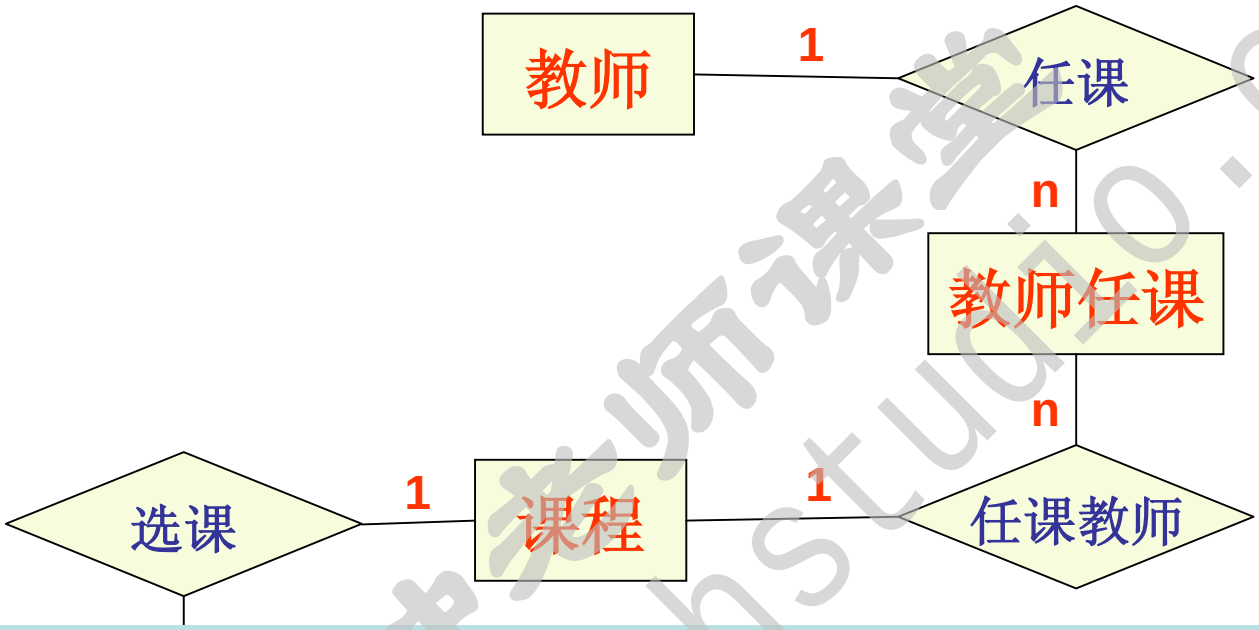
[，<表级完整性约束条件>]

）**ENGINE=存储类型 DEFAULT CHARSET=字符集；**

v完整性约束说明：

* 表级完整性约束：完整性约束条件涉及到该表的多个属性列
* 列级完整性约束：仅对定义的列进行约束

教学管理实体关系图



学生-课程模式 S-T：

学生表：Student(Sno,Sname,Ssex,Sage,Sdept)

课程表：Course(Cno,Cname,Cpno,Ccredit)

学生选课表：SC(Sno,Cno,Grade)

学生表Student

```
CREATE TABLE Student
   ( Sno   CHAR(10)       PRIMARY KEY,
     Sname  VARCHAR(10)  UNIQUE,
     Ssex    CHAR(2)  NOT NULL  check( Ssex IN ('男', '女') ),
     Sage   SMALLINT default 20,
     Sdept  CHAR(20)
     CONSTRAINT Sagelimit check(Sage between 16 and 40)
   ) ENGINE=MyISAM  DEFAULT CHARSET=utf-8;
```

```
mysql> describe student;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| sno   | char(10)   | NO   | PRI | NULL    |       |
| sname | varchar(10)| YES  | UNI | NULL    |       |
| ssex  | char(2)    | NO   |     | NULL    |       |
| sage  | smallint(6)| YES  |     | 20      |       |
| sdep  | char(20)   | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

注意，虽然上面SQL语句定义了CHECK子句，并且通过MYSQL正确执行了，但时间上CHECK子句在MYSQL中是不支持的。思考，如何实现用户自定义约束？

课程表Course

```
CREATE TABLE  Course
       ( Cno      CHAR(4) ,
         Cname  VARCHAR(40),
         Ccredit  SMALLINT,
         PRIMARY KEY(Cno)
       )ENGINE=MyISAM  DEFAULT CHARSET=utf-8;

CREATE TABLE  Course
      ( Cid INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
        Cno      CHAR(4) ,
        Cname  VARCHAR(40),
        Ccredit  SMALLINT,
       PRIMARY KEY (Cid)
      )ENGINE=MyISAM  DEFAULT CHARSET=utf-8;
```

MYSQL中一个表只能定义个自动增长字段，并且必须是主键。

选课成绩表SC

```
CREATE TABLE  SC
    ( Sno  CHAR(10)，
      Cno  CHAR(4)，
      Grade   SMALLINT，
      PRIMARY KEY (Sno，Cno)，
      FOREIGN KEY (Sno) REFERENCES Student(Sno)
       ON Delete Cascade On Update Cascade，
      FOREIGN KEY (Cno) REFERENCES Course(Cno)
       ON Delete Cascade On Update Cascade
    )ENGINE=InnoDB DEFAULT CHARSET=utf-8;
```

注意

1、创建参照完整性要求主、外键的字段数据类型、字符集必须一致。

2、创建主、外键的数据表，都要将其存储引擎设置为 InnoDB。

外键约束说明

- 外键约束有四种形式：
  - RESTRICT:在子表有相关记录时，父表不允许更新或删除。
  - CASCADE:父表在更新或删除时，同时更新或删除子表中对应的记录
  - SET NULL:父表在更新或删除时，子表中对应记录的对应字段设置为NULL
  - NOT ACTION:与RESTRICT相同

2.3.2.9 查看表的结构

- 查看表的结构有三种MYSQL指令
  - **Desc tablename;**
  - **Describe tablename;**
  - **Show create table tablename;**

```
mysql> desc student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| sno   | char(10)    | NO   | PRI | NULL    |       |
| sname | varchar(10) | YES  | UNI | NULL    |       |
| ssex  | char(2)     | NO   |     | NULL    |       |
| sage  | smallint(6) | YES  |     | 20      |       |
| sdep  | char(20)    | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> describe student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| sno   | char(10)    | NO   | PRI | NULL    |       |
| sname | varchar(10) | YES  | UNI | NULL    |       |
| ssex  | char(2)     | NO   |     | NULL    |       |
| sage  | smallint(6) | YES  |     | 20      |       |
| sdep  | char(20)    | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> show create table student;
+---------+-----------------------------------------
-----------------------------------------------------
| Table   | Create Table
+---------+-----------------------------------------
-----------------------------------------------------
| student | CREATE TABLE `student` (
  `sno` char(10) NOT NULL,
  `sname` varchar(10) DEFAULT NULL,
  `ssex` char(2) NOT NULL,
  `sage` smallint(6) DEFAULT '20',
  `sdep` char(20) DEFAULT NULL,
  PRIMARY KEY (`sno`),
  UNIQUE KEY `sname` (`sname`)
) ENGINE=MyISAM DEFAULT CHARSET=gbk |
+---------+-----------------------------------------
-----------------------------------------------------
1 row in set (0.00 sec)
```

2.4 表结构的修改

- 2.4.1 修改表名、字段名
- 2.4.2 修改字段数据类型
- 2.4.3 修改字段排列位置
- 2.4.4 增加字段、删除字段
- 2.4.5 更改表的存储引擎
- 2.4.6 删除表的外键约束

修改表名、字段名

- 修改表名

```
mysql> desc qq;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| tid   | int(11)  | NO   | PRI | NULL    |       |
| qq    | char(15) | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

ALTER TABLE qq RENAME qq_t;

- 修改字段名

ALTER TABLE qq_t CHANGE qq qqno varchar(15);

```
mysql> ALTER TABLE qq RENAME qq_t;
Query OK, 0 rows affected (0.01 sec)

mysql> desc qq;
ERROR 1146 (42S02): Table 'st.qq' doesn't exist
mysql> desc qq_t;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| tid   | int(11)  | NO   | PRI | NULL    |       |
| qq    | char(15) | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE qq_t CHANGE qq qqno varchar(15);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| tid   | int(11)     | NO   | PRI | NULL    |       |
| qqno  | varchar(15) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

2.4.2 修改字段数据类型

ALTER TABLE qq_t MODIFY qqno varchar(20);

```
mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| tid   | int(11)     | NO   | PRI | NULL    |       |
| qqno  | varchar(15) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> alter table qq_t modify qqno varchar(20);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| tid   | int(11)     | NO   | PRI | NULL    |       |
| qqno  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

2.4.3 修改字段排列位置

ALTER TABLE qq_t MODIFY qqno varchar(15) first;

```
mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| tid   | int(11)     | NO   | PRI | NULL    |       |
| qqno  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE qq_t MODIFY qqno varchar(15) first;
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| qqno  | varchar(15) | YES  |     | NULL    |       |
| tid   | int(11)     | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

注意：根据关系数据库的规范化理论：字段的位置无关紧要，实际上调整字段位置是没必要的。

2.4.4 增加字段、删除字段

- 增加字段

ALTER TABLE qq_t ADD email varchar(20);

- 删除字段

ALTER TABLE qq_t DROP email;

```
mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| qqno  | varchar(15) | YES  |     | NULL    |       |
| tid   | int(11)     | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE qq_t ADD email varchar(20);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| qqno  | varchar(15) | YES  |     | NULL    |       |
| tid   | int(11)     | NO   | PRI | NULL    |       |
| email | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

```
mysql> ALTER TABLE qq_t  DROP email;
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc qq_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| qqno  | varchar(15) | YES  |     | NULL    |       |
| tid   | int(11)     | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

2.4.5 更改表的存储引擎

ALTER TABLE qq_t engine=myisam;

```
mysql> show create table qq_t;
+-------+-------------------------------
| Table | Create Table
+-------+-------------------------------
| qq_t  | CREATE TABLE `qq_t` (
  `qqno` varchar(15) DEFAULT NULL,
  `tid` int(11) NOT NULL,
  PRIMARY KEY (`tid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-------------------------------
1 row in set (0.00 sec)
```

```
mysql> ALTER TABLE qq_t  engine=myisam;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table qq_t;
+-------+-------------------------------
| Table | Create Table
+-------+-------------------------------
| qq_t  | CREATE TABLE `qq_t` (
  `qqno` varchar(15) DEFAULT NULL,
  `tid` int(11) NOT NULL,
  PRIMARY KEY (`tid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 |
+-------+-------------------------------
1 row in set (0.00 sec)
```

2.4.6 删除表的外键约束

ALTER TABLE SC DROP FOREIGN KEY FK_sc_1;

```
mysql> show create table sc\G
*************************** 1. row ***************************
       Table: sc
Create Table: CREATE TABLE `sc` (
  `sno` char(10) NOT NULL,
  `cno` char(4) NOT NULL,
  `grade` smallint(6) DEFAULT NULL,
  PRIMARY KEY (`sno`,`cno`),
  KEY `FK_sc_1` (`cno`),
  CONSTRAINT `FK_sc_1` FOREIGN KEY (`cno`) REFERENCES `course` (`cno`) ON DELETE
 CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_sc_2` FOREIGN KEY (`sno`) REFERENCES `student` (`sno`) ON DELET
E CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC
1 row in set (0.00 sec)
```


2.5 表的删除

- 2.5.1 删除无相关的表

```
DROP TABLE qq_t;
```

- 2.5.2 删除被其他表关联的父表
  - 首先要删除外键
  - 然后删除表

2.6 向表中添加数据

- 2.6.1 Insert插入记录
- 2.6.2 Load data 装载记录
- 2.6.3 复制表中的记录

2.6.1 Insert插入记录

- 语句格式

```
INSERT
INTO <表名> [(<属性列1>[，<属性列2 >…)]
VALUES (<常量1> [，<常量2>]  …  )
```

- INTO子句
  - 属性列的顺序可与表定义中的顺序不一致
  - 没有指定属性列
  - 指定部分属性列
- VALUES子句
  - 提供的值必须与INTO子句匹配（值的个数、值的类型）

2.6.1 Insert插入记录

学生表关系模式：Student(Sno,Sname,Ssex,Sage,Sdept)

```
INSERT
INTO Student (Sno，Sname，Ssex，Sdept，Sage)
VALUES ('200215128'，'陈冬'，'男'，'IS'，18)
```

如果插入值的顺序与关系模式中定义一致，可以不输入属性列表。

```
INSERT
INTO Student
VALUES ('200215128'，'陈冬'，'男'，18 ，'IS')
```

可仅填写部分字段

```
INSERT
INTO Student (Sno，Sname，Ssex)
VALUES ('200215128'，'陈冬'，'男')
```

```
mysql> desc student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| sno   | char(10)    | NO   | PRI | NULL    |       |
| sname | varchar(10) | YES  | UNI | NULL    |       |
| ssex  | char(2)     | NO   |     | NULL    |       |
| sage  | smallint(6) | YES  |     | 20      |       |
| sdep  | char(20)    | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

2.6.1 Insert插入记录

MYSQL中，Insert语句一次可以插入多条记录

```
INSERT  INTO  qq_t
VALUES
('57879798',5),
('56498779',6);
```

```
mysql> select * from qq_t;
+-----------+-----+
| qqno      | tid |
+-----------+-----+
| 657897897 |   1 |
| 7979879   |   2 |
| 657897897 |   3 |
| 7979879   |   4 |
+-----------+-----+
4 rows in set (0.00 sec)
```

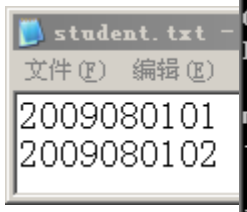
```
mysql> insert into qq_t values
    -> ('56498779',5),
    -> ('55877',6);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from qq_t;
+-----------+-----+
| qqno      | tid |
+-----------+-----+
| 657897897 |   1 |
| 7979879   |   2 |
| 657897897 |   3 |
| 7979879   |   4 |
| 56498779  |   5 |
| 55877     |   6 |
+-----------+-----+
6 rows in set (0.00 sec)
```

2.6.2 Load data 装载记录

MYSQL中，当创建了空数据表后，可以利用Load data方法将数据文件中的记录导入到数据表中。操作步骤：

* 创建文本文件“*.txt”，每行包含一个记录，用定位符(tab)把值分开，并且以CREATE TABLE语句中列出的列次序给出。对于空值NULL，使用\N（反斜线，字母N）表示。
* 将文本文件“*.txt”装载到空数据表中
* **mysql> LOAD DATA LOCAL INFILE ‘*.txt’ INTO TABLE 表名**

  mysql> LINES TERMINATED BY ‘\r\n’ ； 说明行结束符




```
mysql> LOAD DATA LOCAL INFILE 'D:/student.txt' INTO TABLE student
    -> LINES TERMINATED BY '\r\n';
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from student;
+------------+-------+------+------+---------+
| sno        | sname | ssex | sage | sdep    |
+------------+-------+------+------+---------+
| 2009080101 | mark  | m    |   20 | null    |
| 2009080102 | mary  | f    |   20 | compute |
+------------+-------+------+------+---------+
2 rows in set (0.00 sec)
```

2.6.3 复制表中的记录

利用MySQL命令，可以将以原有数据表为基础，创建相同结构和数据的新数据表。帮助在开发过程中快速的复制表格作为测试数据，而不必冒险直接操作正在运行的数据表。

* 创建新表 newtable,并复制mytbl的数据表结构：

```
CREATE TABLE newtable LIKE mytbl;
```

* 将数据表mytbl中的数据复制到新表newtable：

```
INSERT newtable SELECT * FROM mytbl;
```

* 一次复制整个表的结构和数据：

```
CREATE TABLE newtable SELECT * FROM old_table;
```

2.7 表的索引

索引是MYSQL数据库中用来提高性能最常用的工具。MySQL表中所有列类型都可以被索引。

- 2.7.1 MYSQL索引的分类
- 2.7.2 索引设计基本原则
- 2.7.3 索引的创建
- 2.7.4 MYSQL索引的类型与要点

2.7.1 MYSQL索引的分类

- **普通索引**：可在任何数据类型的字段上创建。
- **唯一索引**：UNIQUE索引，该索引对应字段的值不能重复。
- **全文索引**：FULLTEXT索引，只能在CHAR、VARCHAR、TEXT类型字段上创建，仅MyISAM表支持全文索引。
- **单列索引**：索引建立在单个字段上。
- **多列索引**：索引建立在多个字段上。查询条件只有使用了该索引的第一个字段时，索引才会被使用。
- **空间索引**：SPATIAL索引。详情参看MYSQL的空间扩展。

2.7.2 索引设计基本原则

- 索引应创建在用于作为条件中的列、或连接子句中的列。
- 使用唯一索引，索引的列的基数（不同值个数）越大，索引效果越好。
- 使用短索引，如对长字符串可以考虑前缀索引的长度。索引越小，涉及的磁盘IO越少，比较越快。
- 利用最左前缀。对应组合索引（多列索引），可利用最左边的列集进行匹配，这样的列集称为最左前缀。
- 不要过度索引。索引需要占用磁盘空间，并降低写操作性能。应在保证有利于查询优化的前提下尽量少的使用索引。
- 对于InnoDB存储引擎的表，记录默认会按照一定的顺序保存。如按照主键、或唯一索引。

2.7.3 索引的创建

- 1、创建表时创建索引

```
CREATE TABLE 表名（属性名 数据类型 [完整性约束],
                  属性名 数据类型 [完整性约束]，…
                  [UNIQUE | FULLTEXT | SPATIAL] INDEX | KEY
                  [别名]  (属性名 [(长度)] [ ASC | DESC], …)
```

- 2、在现有表上创建索引

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX 索引名
        ON 表名 (属性名 [(长度)] [ ASC | DESC], …)
```

- 3、利用ALTER TABLE语句创建索引

```
ALTER TABLE 表名 ADD [UNIQUE | FULLTEXT | SPATIAL]
INDEX 索引名 (属性名 [(长度)] [ ASC | DESC], …)
```

2.7.3 索引的创建

CREATE TBALE qq (tid int pramarykey AUTO_INCREMENT

qqno char(15),

qqname varchar(10),

email varchar(25),

),

nail desc)

```
mysql> show create table qq \G
*************************** 1. row ***************************
       Table: qq
Create Table: CREATE TABLE `qq` (
  `qqno` varchar(15) DEFAULT NULL,
  `tid` int(11) NOT NULL,
  `qqname` varchar(10) DEFAULT NULL,
  `email` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`tid`),
  UNIQUE KEY `qq_qqno_unique` (`qqno`(10)),
  KEY `qq_qqname_email` (`qqname`,`email`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

CREATE UNIQU

CREATE INDEX qq_qqname_email ON qq(qqname asc,email desc)

ALTER TABLE qq ADD UNIQUE INDEX qq_qqno_unique(qqno(10))

ALTER TABLE qq ADD INDEX qq_qqname_email(qqname asc,email desc)

EXPLAIN 语句查看索引是否被使用

```
mysql> select * from qq;
+----------+-----+--------+-------------------+
| qqno     | tid | qqname | email             |
+----------+-----+--------+-------------------+
| 23432435 |   1 | 天使   | zzhstudio@126.com |
| 9898080  |   2 | 钟老师 | zzh@126.com       |
+----------+-----+--------+-------------------+
```

```
mysql> explain select * from qq \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: qq
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
        Extra:
1 row in set (0.06 sec)
```

```
mysql> explain select * from qq order by qqno asc\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: qq
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
        Extra: Using filesort
1 row in set (0.00 sec)
```

```
mysql> show create table qq \G
*************************** 1. row ***************************
       Table: qq
Create Table: CREATE TABLE `qq` (
  `qqno` varchar(15) DEFAULT NULL,
  `tid` int(11) NOT NULL,
  `qqname` varchar(10) DEFAULT NULL,
  `email` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`tid`),
  UNIQUE KEY `qq_qqno_unique` (`qqno`(10)),
  KEY `qq_qqname_email` (`qqname`,`email`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

不带条件的查询未使用索引

仅按索引字段排序的查询未使用索引

EXPLAIN 语句查看索引是否被使用

```
mysql> explain select * from qq where qqno='9898080'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: qq
         type: const
possible_keys: qq_qqno_unique
          key: qq_qqno_unique
      key_len: 33
          ref: const
         rows: 1
        Extra:
1 row in set (0.06 sec)
```

带有按索引字段查询条件的查询使用了索引

```
mysql> explain select * from qq where email='zzh*'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: qq
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2
        Extra: Using where
1 row in set (0.00 sec)
```

组合索引未使用第一个字段时，查询未使用索引

```
mysql> explain select * from qq where qqname='天*'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: qq
         type: ref
possible_keys: qq_qqname_email
          key: qq_qqname_email
      key_len: 33
          ref: const
         rows: 1
        Extra: Using where
1 row in set (0.06 sec)
```

组合索引使用第一个字段时，查询使用索引

索引的删除

DROP INDEX 索引名 ON 表名;

例如删除**qq**表上的多列索引：

```
DROP INDEX qq_qqname_email ON qq;
```


2.7.4 MYSQL索引的类型与要点

- MYSQL中完整的CREATE INDEX语法

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX 索引名
[索引类型] ON 表名（属性名,..）

存储引擎	允许的索引类型
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

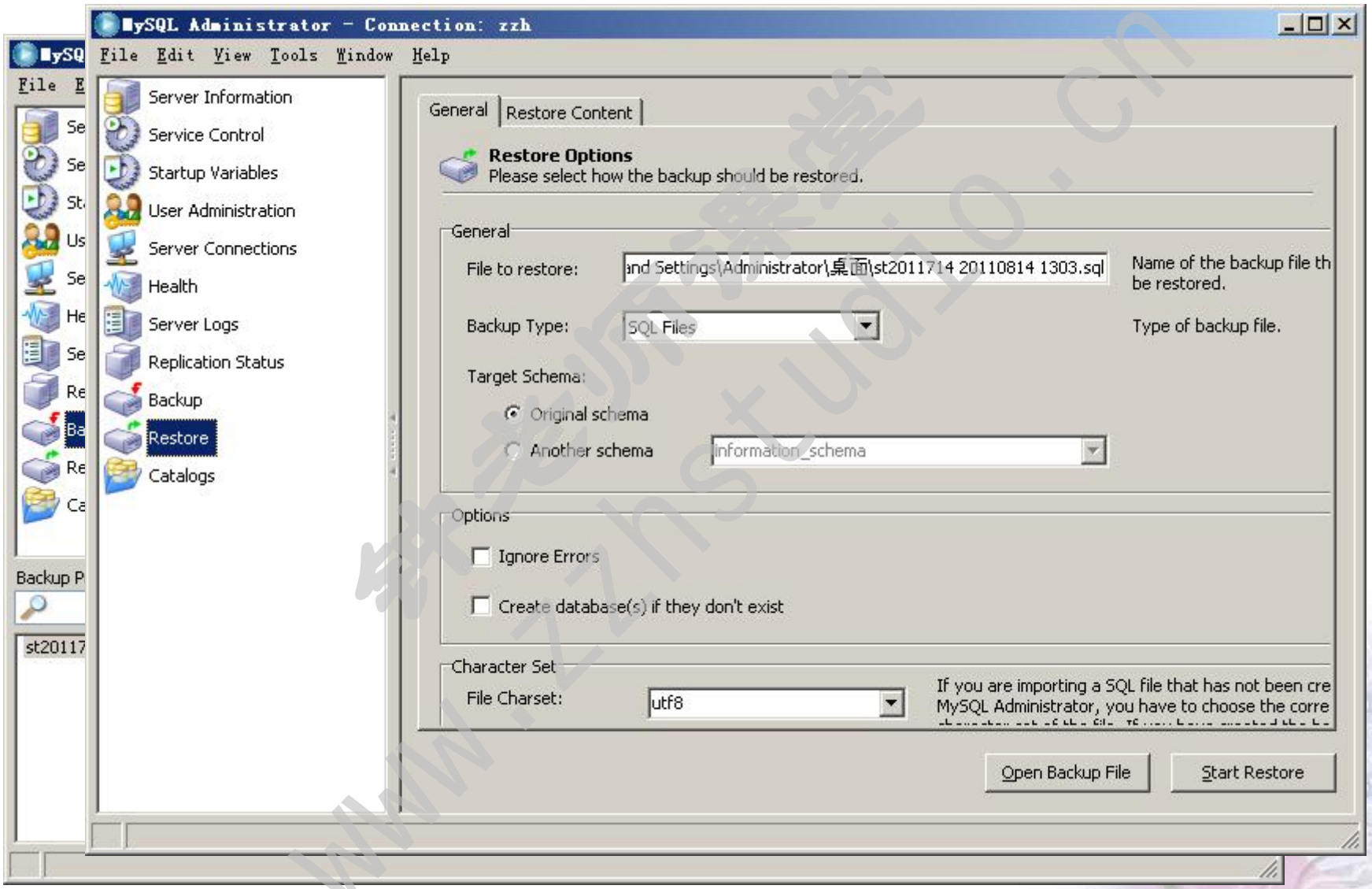
2.7.4 MYSQL索引的类型与要点

- Hash索引特征：
  - 只用于使用=或<=>操作符的等式比较(但很快)。
  - 优化器不能使用hash索引来加速ORDER BY操作。
  - MySQL不能确定在两个值之间大约有多少行。如果你将一个MyISAM表改为hash-索引的MEMORY表，会影响一些查询。
  - 只能使用整个关键字来搜索一行。(用B-树索引，任何关键字的最左面的前缀可用来找到行）。
- BTREE索引，使用>、<、>=、<=、BETWEEN、!=、<>或like’pattern’('pattern'不以通配符开始)的操作时，都可以使用相关列上的索引。

2.8 简单数据库备份与数据导入导出

- 2.8.1 数据库备份、数据库恢复
- 2.8.2 表数据导入、表输入导出

2.8.1 数据库备份、数据库恢复



2.8.2 表数据导入、表输入导出

- 数据导出：

```
Select * from course into outfile 'course.txt'
```

- 数据导入：

```
load data infile 'course.txt' into table course
```


注意：MYSQL数据的导入与导出存在很多选项与设置，需要认真研究，进行数据的转换。

本章内容小结

- 2.1 MYSQL数据库
- 2.2 MYSQL数据类型
- 2.3 表的创建
- 2.4 表结构的修改
- 2.5 表的删除
- 2.6 向表中添加数据
- 2.7 表的索引
- 2.8 简单数据库备份与数据导入导出