1 初涉 MySQL

1.1 MySQL 概述

MySQL 是开源的关系型数据库管理系统。

1.2 MySQL 安装与配置

1） msi 安装、zip 安装

2） my.ini 配置修改

```
[mysql]

default_character_set = utf8

[mysqld]

character_set_server = utf8
```

3） 中文乱码

```
ALTER DATABASE d_name CHARACTER SET = utf8;

ALTER TABLE t_name CONVERT TO CHARACTER SET utf8;

set names gbk;#将客户端编码修改为 GBK;
```

1.2.1 MySQL 的 zip 安装步骤

1、解压 zip

2、创建设置文件

创建配置文件 my.ini，设置[mysqld]分类下的 basedir 和 datadir，一般 basedir 为安装目录，datadir 可放在 basedir 中，例如：

```
basedir = D:/Apache/mysql-5.7.16-winx64

datadir = D:/Apache/mysql-5.7.16-winx64/data
```

3、选择服务类型

跳过

4、初始化数据文件夹

```
mysqld –initialize –console
```

```
mysqld --initialize --user=root --console
```

初始化数据文件夹，并显示自动生成的 root 的密码。

5、首次启动服务，修改密码

```
mysql -u root -p
ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
```

6、从命令行启动

```
mysqld
mysqladmin -u root shutdown
```

7、定制 MySQL 工具的路径

跳过

8、将 MySQL 作为系统服务启动（管理员权限）

```
mysqld -install
mysqld -remove
```

1.2.2 MySQL 在 Linux 下的安装步骤

```
shell> sudo dpkg -i mysql-apt-config_w.x.y-z_all.deb
shell> sudo apt-get update
shell> sudo apt-get install mysql-server
```

状态查看，启动与停止

```
shell> sudo service mysql status
shell> sudo service mysql stop
shell> sudo service mysql start
```

参考：https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/

1.3 启动和停止 MySQL 服务

```
net start mysql
net stop mysql
```

1.4 登录和退出

1） mysql 命令

参数	描述
-D, --database=name	打开指定数据库
--delimiter = name	指定分隔符
-h, --host=name	服务器名称
-p, --password[=name]	密码
-P, --port=#	端口号
--prompt=name	设置提示符
-u, --user=name	用户名
-V, --version	版本信息

2） 登录

```
mysql -uroot -p -P3306 -h127.0.0.1
```

3） 退出

```
exit;quit;\q;
```

1.5 修改 MySQL 提示符

```
mysql>prompt
```

\D 日期 \d 数据库 \h 主机名 \u 用户

1.6 常用命令及语法规范

1） 常用命令

命令	功能
select version()	显示当前服务器版本
select now()	显示当前日期时间
select user()	显示当前用户
SET NAMES gbk	修改客户端字符编码
DELIMITER exp	修改结尾标识符

2） 必须分号结尾

1.7 操作数据库

1）创建数据库

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[DEFAULT] CHARACTER SET [=] charset_name
```

2）显示数据库

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

3）查看创建信息

```
show create database raintest;
```

4）修改数据库

```
ALTER {DATABASE | SCHEMA} [db_name]
[DEFAULT] CHARACTER SET charset_name
```

5）删除数据库

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

2 数据类型与操作数据表

2.1 数据类型

1）整型

类型	字节	有符号	无符号
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

2） 浮点型

类型	字节
FLOAT[(M,D)]	M 总位数，D 小数点后面的位数
DOUBLE[(M,D)]	

3） 日期时间型

列类型	字节	“零”值
DATETIME	8	'0000-00-00 00:00:00'
DATE	3	'0000-00-00'
TIMESTAMP	4	00000000000000
TIME	3	'00:00:00'
YEAR	1	0000

4） 字符型

列类型	存储要求
CHAR(M)	M 个字节，0<=M<=255
VARCHAR(M)	L+1 个字节，$L<2^8$
TINYTEXT	L+1 个字节，$L<2^{16}$
TEXT	L+2 个字节，$L<2^{24}$
MEDIUMTEXT	L+3 个字节，$L<2^8$
LONGTEXT	L+4 个字节，$L<2^{32}$
ENUM(’value1’,’value2’,...)	1 或 2 个字节，取决于枚举值个数（最多 65535）
SET(’value1’,’value2’,...)	1、2、3、4、8 个字节，取决于个数（最多 64）

2.2 数据表操作

1） 创建数据表

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name(
   column_name data_type [NULL | NOT NULL] [ AUTO_INCREMENT]
[Primary Key] [UNIQUE KEY] [DEFAULT value],
);
```

2) 查看所有数据表

```
show tables [from database_name];
```

3) 查看表结构

```
show columns  from table_name;
DESC table_name;
```

2.3 记录操作

1) 插入记录

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
   [INTO] tbl_name [(col_name,...)]
   VALUES ({expr | DEFAULT},...),(...),...
   [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

2) 查找记录

```
SELECT
   select_expr, ...
   [FROM table_references
   [WHERE where_definition]
```

3) 约束

```
[NULL | NOT NULL] [Primary Key] [UNIQUE KEY] [DEFAULT value]
```

3 约束以及修改数据表

3.1 外键约束

1) 要求

关联表必须均使用 InnoDB；//show create table tablename

必须具有相似的数据类型；

必须创建索引，参照列会自动创建；//show indexes from tablename [\G]

禁止使用临时表。

2） 引擎配置

```
default-storage-engine = INNODB
```

3） 语句

```
foreign key(pid) references MainTableName(id) on delete
cascade
```

3.2 外键约束的参照操作

1） cascade：

从父表自动删除或更新子表中匹配的行。

2） set null：

从父表删除或更新行，并设置子表中的外链列为 NULL。子表列必能为 not null。

3） restrict：

拒绝对父表的删除或更新操作。

4） no action：

同 restrict。

3.3 表级约束与列级约束

列级约束使用多，表级极少使用。

3.4 修改数据表

```
ALTER [IGNORE] TABLE tbl_name

    alter_specification [, alter_specification] ...


alter_specification:

#添加一列

    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
```

```
#添加多列
  | ADD [COLUMN] (column_definition,...)
#删除一列
  | DROP [COLUMN] col_name
#添加主键约束
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
#添加唯一约束
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [index_name] [index_type] (index_col_name,...)
#添加外键约束
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
#添加/删除默认值约束
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP
DEFAULT}
#删除主键约束
  | DROP PRIMARY KEY
#删除唯一约束
  | DROP {INDEX|KEY} col_name
#删除外键约束
  | DROP FOREIGN KEY fk_symbol
#修改列定义
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
#修改列定义和名称
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
```

```
#修改数据表表名
  | RENAME [TO|AS] new_tbl_name
```

4 操作数据表中的记录

4.1 插入数据 INSERT

```
1) #方法一 可以插入多条数据
INSERT [INTO] tbl_name [(col_name,...)]
    {VALUES|VALUES} ({expr | DEFAULT},...),(...),...
2) #方法二 一次一条
INSERT [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
3) #方法三 将查询结果插入
INSERT [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

4.2 更新数据 UPDATE

```
1) #Single-table 语法：
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
2) #Multiple-table 语法：
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

4.3 删除数据 DELETE

1） #单表语法：

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
```

2） 多表语法：

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*] ...]
    FROM table_references
    [WHERE where_definition]
```

或：

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*] ...]
    USING table_references
    [WHERE where_definition]
```

4.4 检索数据 SELECT

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [FROM table_references
    [WHERE where_definition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_definition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC] , ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

```
//限制数据量 起始位置，长度
    [PROCEDURE procedure_name(argument_list)]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

5 子查询与连接

5.1 子查询

嵌套在查询中的查询，要放在小括号中。

1） 使用比较运算符

```
= < > <= >= <> != <=>
```

2） 比较运算符修饰

```
ANY、SOME、ALL、[NOT] IN、[NOT] EXISTS
```

5.2 连接

```
table_reference
{[INNER | CROSS] JOIN | {LEFT | RIGHT} [OUTER] JOIN}
table_reference
ON conditional_expr
```

5.3 实例

1） 删除重复记录

```
  DELETE t1 FROM tdb_goods AS t1 LEFT JOIN (SELECT
goods_id,goods_name FROM tdb_goods GROUP BY goods_name
HAVING count(goods_name) >= 2 ) AS t2  ON t1.goods_name =
t2.goods_name  WHERE t1.goods_id > t2.goods_id;
```

6 运算符和函数

6.1 字符函数

CONCAT()　　字符连接

CONCAT_WS() 使用指定的分隔符进行字符连接

FORMAT() 数字转字符格式化

LOWER() 转换成小写字母

UPPER() 转换成大写字母

LEFT() 获取左侧字符

RIGHT() 获取右侧字符

LENGTH() 获取字符串长度

LTRIM() 删除前导空格

RTRIM() 删除后续空格

TRIM() 删除前导和后续空格

SUBSTRING() 字符串截取

[NOT] LIKE 模式匹配

REPLACE() 字符串替换

6.2 数值运算符与函数

CEIL() 进一取整

FLOOR() 舍一取整

DIV 整数除法

MOD 取模

POWER() 幂运算

ROUND() 四舍五入

TRUNCATE()数字截取

6.3 比较运算符与函数

[NOT] BETWEEN...AND...[不]在范围内

[NOT] IN() [不]在列出的范围内

IS [NOT] NULL [不]为空

6.4 日期时间函数

NOW() 当前日期和时间

CURDATE() 当前日期

CURTIME() 当前时间

DATE_ADD() 日期变化

DATEDIFF() 日期差值

DATE_FORMAT() 日期格式化

6.5 信息函数

CONNECTION_ID() 连接 ID

DATEBASE() 当前数据库

LAST_INSERT_ID() 最后插入记录的 ID

USER() 当前用户

VERSION() 版本信息

6.6 聚合函数

AVG() 平均值

COUNT()计数

MAX() 最大值

MIN() 最小值

SUM() 求和

6.7 加密函数

MD5() 信息摘要算法

PASSWORD()密码算法

7 自定义函数

7.1 创建语法

1）简要

```
CREATE FUNCTION function_name
RETURNS
{STRING | INTEGER | REAL | DECIMAL}
routine_body
```

2）详细

```
CREATE FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

    proc_parameter:
    [ IN | OUT | INOUT ] param_name type

    func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

```
  | COMMENT 'string'


routine_body:

   Valid SQL procedure statement or statements
```

7.2 创建无参函数

```
   CREATE FUNCTION NOW_CHINA() RETURNS VARCHAR(30)

   RETURN DATE_FORMAT(NOW(),'%Y年%m月%d日 %H时%i分%s秒');
```

7.3 创建有参函数

```
CREATE FUNCTION MyAvgFun(num1 SMALLINT UNSIGNED,num2 SMALLINT
UNSIGNED) RETURNS FLOAT(10,2) UNSIGNED

RETURN (num1+num2)/2;
```

7.4 创建具有复合结构函数体的函数

```
CREATE FUNCTION ADDUSER(username VARCHAR(20))

RETURNS INT UNSIGNED

BEGIN

INSERT users(username) VALUES(username);

RETURN LAST_INSERT_ID();

END
```

8 存储过程

8.1 简介

1） 定义

存储过程是 SQL 语句和控制语句的预编译集合，以一个名称存储并作为一个单元处理。

2） 优点

增强 SQL 语句的功能和灵活性；

实现较快的执行速度；

减少网络流量。

8.2 创建存储过程

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body


proc_parameter:
[ IN | OUT | INOUT ] param_name type
characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'


routine_body:
   Valid SQL procedure statement or statements
```

8.3 过程体

1） 由合法的 SQl 语句构成；

2） 可以是任意 SQL 语句；

3） 为复合结构则使用 BEGIN...END 语句；

4） 复合结构可以包含声明，循环，控制结构；

8.4 创建不带参数的存储过程

1） 创建

```
CREATE PROCEDURE sp1() SELECT VERSION();
```

2） 调用

```
CALL sp1();
```

8.5 创建带有 IN 类型参数的存储过程

1） 创建

```
mysql> delimiter //
mysql> CREATE PROCEDURE removeUserById (IN p_id INT UNSIGNED)
    -> BEGIN
    ->   DELETE FROM users WHERE id= p_id;//参数名与字段名不能一样
    -> END
   -> //
```

2） 调用

```
mysql> delimiter ;
mysql> CALL removeUserById(3);
```

3） 修改

```
//不能修改过程体
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic:
   { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
  | SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

4） 删除

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

5） 查看创建语句

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

6） 查看状态信息

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

8.6 创建带有 IN 和 OUT 类型参数的存储过程

1） 创建

```
mysql> delimiter //

mysql> CREATE PROCEDURE removeUserAndReturnUserNums (IN p_id
INT UNSIGNED, OUT userNums INT UNSIGNED)

    -> BEGIN

    ->   DELETE FROM users WHERE id= p_id;//参数名与字段名不能一样

    ->   SELECT count(id) FROM users INTO userNums;

    -> END

    -> //
```

2） 调用

```
mysql> delimiter ;

mysql> CALL removeUserAndReturnUserNums(13, @nums);//用户（客户
端）变量

mysql> SELECT @nums;
```

8.7 创建带有多个 OUT 类型参数的存储过程

1） 创建

```
mysql> delimiter //

mysql> CREATE PROCEDURE removeUserByAgeAndReturnInfos(IN p_age
SMALLINT UNSIGNED, OUT deleteUsers INT UNSIGNED, OUT
userCounts INT UNSIGNED)

    -> BEGIN

    ->   DELETE FROM users WHERE age= p_age;

    ->   SELECT ROW_COUNT() INTO deleteUsers;//受影响条数

    ->   SELECT COUNT(id) FROM users INTO userCounts;

    -> END

    -> //
```

2） 调用

```
mysql> delimiter ;

mysql> CALL removeUserByAgeAndReturnInfos(20, @a, @b);

mysql> SELECT @a,@b
```

8.8 存储过程与自定义函数的区别

	存储过程	函数
特性	功能复杂	针对性强
返回值数量	任意个	一个
独立性	独立执行	作为其他 SQL 的组成部分
使用	较多	很少

9 存储引擎

9.1 简介

MySQL 可以将数据以不同的技术存储在文件（内存）中，这种技术就称为存储引擎。

每一种存储引擎使用不同的存储机制、索引技巧、锁定水平，最终提供广泛且不同的功能。

MySQL 主要引擎包括：MyISAM、InnoDB、Memory、CSV、Archive

9.2 并发处理

1） 并发控制

当多个连接对记录进行修改时保证数据的一致性和完整性。

2） 锁

共享锁（读锁）：同一时间段内，多个用户可以读取同一个资源，读取过程中数据不会发生任何改变。

排他锁（写锁）：在任何时候只能有一个用户写入资源，当进行写锁时会阻塞其他的读锁或者写锁操作。

3） 锁颗粒

表锁，开销最小。

行锁，开销最大。

9.3 事务处理

1） 事务

用于保证数据库的完整性。

2） 特性

原子性、一致性、隔离性、持久性

Atomicity、Consistency、Isolation、Durability

9.4 外键和索引

1） 外键

保证数据一致性的一种策略。

2） 索引

对数据表中一列或多列的值进行排序的一种结构。

9.5 各引擎比较

特点	MyISAM	InnoDB	Memory	Archive
存储限制	256TB	64TB	有	无
事务安全	-	支持	-	-
支持索引	支持	支持	支持	-
锁颗粒	表锁	行锁	表锁	行锁
数据压缩	支持	-	-	支持
支持外键	-	支持	-	-

9.6 设置引擎

1） 修改 MySQL 配置文件实现

```
default-storage-engine = INNODB
```

2） 创建数据表命令实现

```
CREATE TABLE table_name(
   exp...
)ENGINE = INNODB;
```

3) 修改数据表引擎

```
CREATE TABLE table_name ENGINE = INNODB;
```

10 mysql8

```
ALTER USER 'root'@'localhost'
  IDENTIFIED WITH mysql_native_password
  BY 'password';
```