MySQL笔记

一、开始使用

创建、删除、使用数据库、退出Mysql

1.退出数据库

```
exit
```

2.创建数据库

```
CREATE DATABASE student;
```

3.删除数据库

```
drop database student;
```

4.选择数据库

```
use student;
```

5.显示数据库

```
show databases;
```

二、MySQL数据类型

MySQL数据类型：数值、日期/时间和字符串(字符)

1.数值类型

- INT:4Bytes
- TINYINT:1Bytes
- FLOAT:4Bytes
- DOUBLE:8Bytes

2.日期/时间类型

- DATE:1000-01-01/9999-12-31
- TIME:'-838:59:59'/'838:59:59'
- YEAR:1901/2155

3.字符串类型

- CHAR:定长字符串(0-255 bytes)
- VARCHAR:变长字符串(0-65535 bytes)
- TEXT:长文本数据(0-65535 bytes)

三、数据表操作

1.创建数据表

通用语法

```
CREATE TABLE sheet0 (id INT);
```

示例（为了区分MySQL的关键字与普通字符，MySQL引入了一个反引号。）

```
CREATE TABLE IF NOT EXISTS `sheet1`(
   `stu_id` INT NOT NULL AUTO_INCREMENT,
   `stu_name` VARCHAR(100) NOT NULL,
   `stu_age` INT NOT NULL,
   `stu_bir` DATE,
   PRIMARY KEY (`stu_id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

参数

- NOT NULL：不想字段为NULL
- AUTO_INCREMENT：自增属性
- PRIMARY KEY：设置主键
- ENGINE：存储引擎
- CHARSET：设置编码

2.删除数据表

```
DROP TABLE sheet0;
```

3.显示数据表列表

```
SHOW TABLES;
SHOW TABLES FROM student;
```

4.显示数据表结构

```
SHOW COLUMNS FROM sheet1;
```

5.插入数据

```
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1001,"zcz",20,"2002-07-29");
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1002,"Tom",26,"1998-03-04");
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1003,"lisa",25,"1999-09-29");
```

6.读取数据表

```
select * from sheet1;
```

7.表中数据清空

```
delete from sheet1;
```

示例1

```
CREATE DATABASE student;
use student;
show databases;
CREATE TABLE IF NOT EXISTS `sheet1`(
   `stu_id` INT NOT NULL AUTO_INCREMENT,
   `stu_name` VARCHAR(100) NOT NULL,
   `stu_age` INT NOT NULL,
   `stu_bir` DATE,
   PRIMARY KEY (`stu_id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
SHOW TABLES FROM student;
SHOW COLUMNS FROM sheet1;
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1001,"zcz",20,"2002-07-29");
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1002,"Tom",26,"1998-03-04");
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1003,"lisa",25,"1999-09-29");
select * from sheet1;
```

```
delete from sheet1;
DROP TABLE sheet1;
drop database student;
show databases;

CREATE DATABASE student;
use student;
show databases;
CREATE TABLE IF NOT EXISTS `sheet1`(
   `stu_id` INT NOT NULL AUTO_INCREMENT,
   `stu_name` VARCHAR(100) NOT NULL,
   `stu_age` INT NOT NULL,
   `stu_bir` DATE,
   PRIMARY KEY (`stu_id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
SHOW TABLES FROM student;
SHOW COLUMNS FROM sheet1;
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1001,"zcz",20,"2002-07-29");
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1002,"Tom",26,"1998-03-04");
INSERT INTO sheet1(stu_id,stu_name,stu_age,stu_bir) VALUES(1003,"lisa",25,"1999-09-29");
select * from sheet1;
```

四、查询数据

1.基本查询语句

SELECT column_name,column_name
FROM table_name
[WHERE Clause]
[LIMIT N][OFFSET M]

- WHERE语句来包含任何条件

- LIMIT属性来设定返回的记录数
- OFFSET指定SELECT语句开始查询的数据偏移量

2.WHERE子句

WHERE [BINARY] condition1 [AND|OR] condition2.....

- WHERE 子句也可以运用于 SQL 的 DELETE 或者 UPDATE 命令
- WHERE 子句类似于程序语言中的 if 条件
- BINARY 关键字来设定 WHERE 子句的字符串比较是区分大小写的

```
-- 使用示例1
SELECT * from sheet1 WHERE stu_name='zcz';
SELECT * from sheet1 WHERE BINARY stu_name='ZCZ';
SELECT * from sheet1 WHERE BINARY stu_name='zcz';
```

3.UPDATE更新

UPDATE table_name SET field1=new-value1, field2=new-value2 [WHERE Clause]

```
-- 使用示例1
UPDATE sheet1 SET stu_name='ewk' WHERE stu_id=1003;
select * from sheet1;
```

4.DELETE语句(删除数据)

DELETE FROM table_name [WHERE Clause]

```
-- 使用示例1
DELETE FROM sheet1 WHERE stu_id=1003;
select * from sheet1;
```

5.LIKE子句

LIKE 子句中使用百分号 %字符来表示任意字符。

语法格式

SELECT field1, field2,...fieldN FROM table_name
WHERE field1 LIKE condition1 [AND|OR] filed2 = 'somevalue'

要点

- 可以在 WHERE 子句中使用LIKE子句
- 可以在 DELETE 或 UPDATE 命令中使用 WHERE...LIKE 子句来指定条件
- LIKE 通常与 % 一同使用，类似于一个元字符的搜索
- 如果没有使用百分号 %, LIKE 子句与等号 = 的效果是一样的

示例

```
-- 使用示例1
SELECT * from sheet1  WHERE stu_id LIKE '%%1';
SELECT * from sheet1  WHERE stu_name LIKE 'z%%';
```

6.UNION语句

UNION 操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。

语法格式

SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
UNION [ALL | DISTINCT]
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];

- **DISTINCT:** 可选，删除结果集中重复的数据,默认
- **ALL:**可选，返回所有结果集，包含重复数据

```
-- 使用示例2
select * from sheet1;
select * from sheet2;
-- 1 *
SELECT * FROM sheet1
UNION
SELECT * FROM sheet2;
-- 2 ORDER BY ID
SELECT * FROM sheet1
UNION
SELECT * FROM sheet2
ORDER BY id;
-- 3 id/ALL
SELECT id FROM sheet1
UNION ALL
SELECT id FROM sheet2
-- 4 id/ALL/ORDER BY ID
SELECT id FROM sheet1
UNION ALL
SELECT id FROM sheet2
ORDER BY id;
```

7.ORDER BY子句排序

SELECT field1, field2,...fieldN FROM table_name1, table_name2...
ORDER BY field1 [ASC [DESC][默认 ASC]], [field2...] [ASC [DESC][默认 ASC]]

- ASC 或 DESC 关键字来设置查询结果是按升序或降序排列。 默认情况下，它是按升序排列。
- 可以设定多个字段来排序

```
-- 使用示例2
SELECT * from sheet1 ORDER BY id ASC;
SELECT * from sheet1 ORDER BY id DESC;
```

8.GROUP BY语句

GROUP BY 语句根据一个或多个列对结果集进行分组。在分组的列上我们可以使用 COUNT, SUM, AVG,等函数。

SELECT column_name, function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;

```
-- 使用示例2
-- 统计每组有多少条记录
SELECT name, COUNT(*) as nameCount FROM sheet1 GROUP BY name;
-- 分组求和：NULL表示所有组的总和
SELECT name, SUM(money) as nameCount FROM sheet1 GROUP BY name WITH ROLLUP;
-- 使用coalesce来设置一个可以取代NUll的名称
SELECT coalesce(name,'总数'), SUM(money) as nameCount FROM sheet1 GROUP BY name WITH ROLLUP;
```

示例2

```
CREATE DATABASE class;
use class;
-- 表1
CREATE TABLE IF NOT EXISTS `sheet1`(
   `id` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(100) NOT NULL,
   `age` INT NOT NULL,
   `money` INT,
   PRIMARY KEY (`id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
INSERT INTO sheet1(id,name,age,money) VALUES(1001,"张三",20,100);
INSERT INTO sheet1(id,name,age,money) VALUES(1002,"李四",19,200);
INSERT INTO sheet1(id,name,age,money) VALUES(1003,"王五",18,80);
INSERT INTO sheet1(id,name,age,money) VALUES(1004,"王五",27,300);
-- 表2
CREATE TABLE IF NOT EXISTS `sheet2`(
   `id` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(100) NOT NULL,
   `age` INT NOT NULL,
   `grade` VARCHAR(100) NOT NULL,
   PRIMARY KEY (`id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
INSERT INTO sheet2(id,name,age,grade) VALUES(1001,"Sira",20,"A+");
INSERT INTO sheet2(id,name,age,grade) VALUES(1002,"Tom",26,"B-");
INSERT INTO sheet2(id,name,age,grade) VALUES(1003,"lisa",25,"C+");
INSERT INTO sheet2(id,name,age,grade) VALUES(1005,"Ann",24,"A-");
-- 展示数据表
show databases;
SHOW TABLES FROM class;
SHOW COLUMNS FROM sheet1;
SHOW COLUMNS FROM sheet2;
select * from sheet1;
select * from sheet2;
```

```
delete from sheet1;
delete from sheet2;
DROP TABLE sheet1;
DROP TABLE sheet2;
drop database class;
show databases;
```

五、连接

JOIN 在两个或多个表中查询数据

JOIN 按照功能大致分为如下三类：

- **INNER JOIN（内连接,或等值连接）**：获取两个表中字段匹配关系的记录，默认。
- **LEFT JOIN（左连接）：**获取左表所有记录，即使右表没有对应匹配的记录。
- **RIGHT JOIN（右连接）：** 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。

```
-- 使用示例2
-- 写法1
SELECT * FROM sheet1 a INNER JOIN sheet2 b ON a.id = b.id;
SELECT * FROM sheet1 a LEFT JOIN sheet2 b ON a.id = b.id;
SELECT * FROM sheet1 a RIGHT JOIN sheet2 b ON a.id = b.id;
-- 写法2(内连接)
SELECT * FROM sheet1 a, sheet2 b WHERE a.id = b.id;
```

六、NULL值的处理

MySQL 使用 SQL SELECT 命令及 WHERE 子句来读取数据表中的数据,但是当提供的查询条件字段为 NULL 时，该命令可能就无法正常工作。

为了处理这种情况，MySQL提供了三大运算符:

- **IS NULL:** 当列的值是 NULL,此运算符返回 true。
- **IS NOT NULL:** 当列的值不为 NULL, 运算符返回 true。
- **<=>:** 比较操作符（不同于 = 运算符），当比较的的两个值相等或者都为 NULL 时返回 true。

NULL的特殊性

- 不能使用 = NULL 或 != NULL 在列中查找 NULL 值 。
- 在 MySQL 中，NULL值与其它值的比较（即使是 NULL）永远返回 NULL，即 NULL = NULL 返回 NULL 。

```
-- 使用示例3
-- 不起作用
SELECT * FROM sheet1 WHERE stu_age = NULL;
SELECT * FROM sheet1 WHERE stu_age != NULL;
-- 正确写法
SELECT * FROM sheet1 WHERE stu_age IS NULL;
SELECT * FROM sheet1 WHERE stu_age IS NOT NULL;
```

示例3

```
CREATE DATABASE student;
use student;
show databases;
CREATE TABLE IF NOT EXISTS `sheet1`(
```

```
    `stu_id` INT NOT NULL AUTO_INCREMENT,
    `stu_name` VARCHAR(100) NOT NULL,
    `stu_age` INT,
    PRIMARY KEY (`stu_id`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
SHOW TABLES FROM student;
SHOW COLUMNS FROM sheet1;
INSERT INTO sheet1(stu_id,stu_name,stu_age) VALUES(1001,"zcz",20);
INSERT INTO sheet1(stu_id,stu_name,stu_age) VALUES(1002,"Tom",26);
INSERT INTO sheet1(stu_id,stu_name,stu_age) VALUES(1003,"lisa",25);
INSERT INTO sheet1(stu_name) VALUES("张三");
INSERT INTO sheet1(stu_name) VALUES("李四");
select * from sheet1;
```

```
drop database student;
show databases;
```

七、正则表达式

MySQL中使用 REGEXP 操作符来进行正则表达式匹配。

- ^匹配开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
- $匹配结束位置。如果设置了RegExp 对象的 Multiline 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
- .匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用像 '[.\n]' 的模式。
- [...]：字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
- [^...]：负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
- p1|p2|p3：匹配 p1 或 p2 或 p3。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 匹配 "zood" 或 "food"。
- *匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
- +匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
- {n}：n为非负整数。匹配n次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
- {n,m}：m和n为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。

```
-- 使用示例3
-- 1.以'z'开头
SELECT stu_name FROM sheet1 WHERE stu_name REGEXP '^z';
-- 2.以'z'结尾
SELECT stu_name FROM sheet1 WHERE stu_name REGEXP 'z$';
-- 3.包含'zc'
SELECT stu_name FROM sheet1 WHERE stu_name REGEXP 'zc';
-- 4.以'z'开头或以'z'结尾的所有数据
SELECT stu_name FROM sheet1 WHERE stu_name REGEXP '^zc|cz$';
```

八、事务

1.事务的运用场景

- MySQL 事务主要用于处理操作量大，复杂度高的数据。
- 在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务；
- 事务处理可以用来维护数据库的完整性，保证成批的 SQL 语句要么全部执行，要么全部不执行；
- 事务用来管理 insert,update,delete 语句。

2.必须满足的4个条件

一般来说，事务是必须满足4个条件（ACID）：原子性（Atomicity，或称不可分割性）、一致性（Consistency）、隔离性（Isolation，又称独立性）、持久性（Durability）。

3.注意事项

在 MySQL 命令行的默认设置下，事务都是自动提交的，即执行 SQL 语句后就会马上执行 COMMIT 操作。因此要显式地开启一个事务务须使用命令 BEGIN 或 START TRANSACTION，或者执行命令 SET AUTOCOMMIT=0，用来禁止使用当前会话的自动提交。

4.事务控制语句

- 显式开启一个事务：BEGIN 或 START TRANSACTION；
- 提交事务：COMMIT或COMMIT WORK。
- 事务回滚：ROLLBACK或ROLLBACK WORK。结束用户的事务，撤销正在进行的所有未提交的修改。
- 创建保存点：SAVEPOINT identifier。一个事务中可以有多个 SAVEPOINT。
- 删除事务保存点：RELEASE SAVEPOINT identifier。当没有指定保存点时，执行该语句会抛出一个异常；
- 把事务回滚到标记点：ROLLBACK TO identifier。
- 设置事务的隔离级别：SET TRANSACTION 。InnoDB 存储引擎提供事务的隔离级别有 READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ 和 SERIALIZABLE。

5.事务处理的两种方法

用 BEGIN, ROLLBACK, COMMIT来实现

- BEGIN 开始一个事务
- ROLLBACK 事务回滚
- COMMIT 事务确认

直接用 SET 来改变 MySQL 的自动提交模式:

- SET AUTOCOMMIT=0 禁止自动提交
- SET AUTOCOMMIT=1 开启自动提交

6.事务代码测试

```
-- 使用示例3
begin;    # 开始事务
select * from sheet1;
INSERT INTO sheet1(stu_id,stu_name,stu_age) VALUES(1006,"Hello",18);
select * from sheet1;
rollback;   # 回滚
select * from sheet1;
commit; # 提交事务
-- 提交事务后，再回滚事务，不再生效。
INSERT INTO sheet1(stu_id,stu_name,stu_age) VALUES(1007,"nihao",19);
rollback;   # 回滚
select * from sheet1;
```

```
-- 检查autocommit模式是否有开启
select @@autocommit;
-- 如果需要关闭，则输入以下代码
set autocommit = 0;

-- 以下代码测试失败
-- 插入保存点
SAVEPOINT a;
-- 用保存点进行回滚
rollback to SAVEPOINT a;
-- 删除保存点
RELEASE SAVEPOINT a;
```

示例4

```
CREATE DATABASE alterTest;
use alterTest;
show databases;
CREATE TABLE IF NOT EXISTS `sheet1`(
    `math` INT,
    `english` INT,
    `chinese` INT,
    `game` INT
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
SHOW TABLES FROM alterTest;
SHOW COLUMNS FROM sheet1;
INSERT INTO sheet1(math,english,chinese,game) VALUES(99,45,54,87);
INSERT INTO sheet1(math,english,chinese,game) VALUES(45,83,54,97);
INSERT INTO sheet1(math,english,chinese,game) VALUES(65,73,54,56);
INSERT INTO sheet1(math,english,chinese,game) VALUES(76,29,54,44);
INSERT INTO sheet1(math,english,chinese,game) VALUES(95,73,67,56);
select * from sheet1;
```

```
drop database alterTest;
show databases;
```

九、ALTER命令

修改数据表名或者修改数据表字段时，就需要使用到MySQL ALTER命令。

1.删除表字段

注意事项：如果数据表中只剩余一个字段则无法使用DROP来删除字段。

补充：truncate、drop和delete的区别

- truncate与drop是DDL语句，执行后无法回滚；delete是DML语句，可回滚。
- truncate只能作用于表；delete，drop可作用于表、视图等。
- truncate会清空表中的所有行，但表结构及其约束、索引等保持不变；drop会删除表的结构及其所依赖的约束、索引等。
- truncate会重置表的自增值；delete不会。
- truncate不会激活与表有关的删除触发器；delete可以。
- truncate后会使表和索引所占用的空间会恢复到初始大小；delete操作不会减少表或索引所占用的空间，drop语句将表所占用的空间全释放掉。

```
-- 使用示例4
-- ALTER 命令及 DROP 子句删除字段
SHOW COLUMNS FROM sheet1;
ALTER TABLE sheet1 DROP math;  # 删除字段（不可回滚）
SHOW COLUMNS FROM sheet1;
```

2.增加表字段

指定新增字段的位置：FIRST (设定位第一列)， AFTER 字段名（设定位于某个字段之后）。

```
-- 使用示例4
-- ALTER 命令及 ADD 子句增加字段
select * from sheet1;
ALTER TABLE sheet1 ADD Japanese INT;
select * from sheet1;
ALTER TABLE sheet1 ADD math INT FIRST;
ALTER TABLE sheet1 ADD math2 INT AFTER math;
select * from sheet1;
```

3.修改字段类型及名称

修改字段类型及名称, 可以在ALTER命令中使用 MODIFY 或 CHANGE 子句 。

```
-- 使用示例4
-- ALTER 命令及 MODIFY 子句修改字段类型
SHOW COLUMNS FROM sheet1;
ALTER TABLE sheet1 MODIFY math CHAR(10);
SHOW COLUMNS FROM sheet1;
-- ALTER 命令及 CHANGE 子句修改字段类型和名称
ALTER TABLE sheet1 CHANGE math math VARCHAR(100);
SHOW COLUMNS FROM sheet1;
```

4.对NULL、默认值影响

如果不设置默认值，MySQL会自动设置该字段默认为 NULL。

- 当要修改的字段math中有NOT NULL值时，不能MODIFY为NULL值。
- 字段全为NULL值时，不能MODIFY为NOT NULL属性

```
-- 使用示例4
-- 修改设置默认值为100
INSERT INTO sheet1(math,english,chinese,game) VALUES(95,73,67,56);
select * from sheet1;
-- 当要修改的字段math中有NOT NULL值时，不能MODIFY为NULL值。
ALTER TABLE sheet1 MODIFY math INT DEFAULT 100;  # 此项全为NULL值，不能MODIFY为NOT NULL属性
SHOW COLUMNS FROM sheet1;
-- 删除字段默认值
ALTER TABLE sheet1 ALTER math DROP DEFAULT;
SHOW COLUMNS FROM sheet1;
```

5.修改表名

```
-- 使用示例4
SHOW COLUMNS FROM sheet1;

-- 此处运行失败，显示不存在math
ALTER TABLE math RENAME TO math1;
SHOW COLUMNS FROM sheet1;
```

ALTER 命令还可以用来创建及删除MySQL数据表的索引。

十、索引

索引的优缺点：

- 优点：索引可以大大提高MySQL的检索速度。
- 缺点：建立索引占用磁盘空间索引文件，降低更新表的速度。（更新表时，不仅要保存数据，还要保存索引文件。）

索引分类：

- 单列索引，即一个索引只包含单个列，一个表可以有多个单列索引，但这不是组合索引。
- 组合索引，即一个索引包含多个列。

使用场景：

- 创建索引时，需要确保该索引是应用在 SQL 查询语句的条件(一般作为 WHERE 子句的条件)。
- 索引用于快速找出在某个列中有一特定值的行。不使用索引，MySQL必须从第一条记录开始读完整个表，直到找出相关的行，表越大查询数据所花费的时间就越多。如果表中查询的列有索引，MySQL能够快速到达一个位置去搜索数据文件，而不必查看所有数据，那么将会节省很大一部分时间。

本质：

- 索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录。
- 一个索引就是创建一个B+树。

普通索引：

1.显示索引信息

```
-- 使用示例4
-- 添加\G来格式化输出信息。\G后不用加分号，否则就会出现错误。
SHOW INDEX FROM sheet1\G
```

2.创建索引

```
CREATE INDEX gameindex ON sheet1 (game);
SHOW INDEX FROM sheet1\G
```

3.删除索引

```
DROP INDEX gameindex ON sheet1;
SHOW INDEX FROM sheet1\G
```

4.修改表结构(添加索引)

```
ALTER table sheet1 ADD INDEX gameindex(game);
SHOW INDEX FROM sheet1\G
```

5.创建表时直接指定索引

索引长度的指定

- 如果是CHAR，VARCHAR类型，length可以小于字段实际长度；
- 如果是BLOB和TEXT类型，必须指定 length；
- 通过减小索引长度，能够减小索引文件的大小，加快数据的insert。

```
CREATE TABLE sheet2(
ID INT NOT NULL,
username VARCHAR(16) NOT NULL,
INDEX IDindex (username(8))
);
SHOW INDEX FROM sheet2\G
```

6.唯一索引

普通索引与唯一索引的异同：

- 普通索引允许被索引的数据列包含重复的值，唯一索引可以保证数据记录的唯一性。
- 两者都允许有空值。

7.ALTER添加和删除索引

添加索引

- ALTER TABLE tbl_name ADD PRIMARY KEY (column_list)：添加主键。索引值唯一，且不能为NULL
- ALTER TABLE tbl_name ADD UNIQUE index_name (column_list)：索引值唯一（除NULL外）
- ALTER TABLE tbl_name ADD INDEX index_name (column_list)：添加普通索引，索引值可出现多次。
- ALTER TABLE tbl_name ADD FULLTEXT index_name (column_list)：指定索引为FULLTEXT，全文索引。

在 ALTER 命令中使用 DROP 子句来删除索引

```
-- 使用示例4
-- sheet2在10.5时创建
ALTER TABLE sheet2 DROP INDEX IDindex;
SHOW INDEX FROM sheet2\G
```

8.ALTER添加和删除主键

主键本质、用法

- 主键是一种特殊的唯一索引；
- 主键作用于列上（可以一个列或多个列联合主键）
- 添加主键索引时，需要确保该主键默认不为空（NOT NULL）。

添加主键

```
-- 使用示例4
ALTER TABLE sheet2 ADD PRIMARY KEY (ID);
SHOW INDEX FROM sheet2\G  # 主键必须为NOT NULL
```

删除主键

```
-- 使用示例4
ALTER TABLE sheet2 DROP PRIMARY KEY;
SHOW INDEX FROM sheet2\G
```

十一、复制表

如果需要完全的复制MySQL的数据表，包括表的结构，索引，默认值等。 如果仅仅使用**CREATE TABLE ... SELECT** 命令，是无法实现的。

完整复制MySQL数据表，步骤如下：

- 获取创建数据表的语句：**SHOW CREATE TABLE** 包含原数据表的结构、索引等。
- 创建新表：复制显示的SQL语句，修改数据表名，并执行SQL语句创建新表。
- 复制表的内容： **INSERT INTO ... SELECT** 语句来实现。

```
-- 使用示例4
SHOW CREATE TABLE sheet2 \G
```

```
-- 复制到此处
CREATE TABLE `sheet2` (
  `ID` int(11) NOT NULL,
  `username` varchar(16) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
-- 修改表名再执行
CREATE TABLE `sheet2copy` (
  `ID` int(11) NOT NULL,
  `username` varchar(16) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
-- 显示新表结构
SHOW COLUMNS FROM sheet2copy;
```

十二、元数据

MySQL的三种信息：

- **查询结果信息：** SELECT, UPDATE 或 DELETE语句影响的记录数。
- **数据库和数据表的信息：** 包含了数据库及数据表的结构信息。
- **MySQL服务器信息：** 包含了数据库服务器的当前状态，版本号等。

命令	描述
SELECT VERSION( )	服务器版本信息
SELECT DATABASE( )	当前数据库名 (或者返回空)
SELECT USER( )	当前用户名
SHOW STATUS	服务器状态
SHOW VARIABLES	服务器配置变量

```
SELECT VERSION();
SELECT DATABASE();
SELECT USER();

-- 这两项信息非常长
-- SHOW STATUS;
-- SHOW VARIABLES;
```

十三、序列使用

- mysql的AUTO_INCREMENT可以设置起始值，但是不能设置步长，其步长默认就是1；
- mysql一个表只能有一个自增长字段。

1.AUTO_INCREMENT

使用 MySQL AUTO_INCREMENT 来定义序列，自增字段必须定义为主键属性。

```
-- 使用示例4
CREATE TABLE sheet3(
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
name VARCHAR(30) NOT NULL,
PRIMARY KEY (id)
);
SHOW COLUMNS FROM sheet3;
```

2.获取最后插入自增列值

```
-- 使用示例4
INSERT INTO sheet3(id,name) VALUES(1001,"zcz");
INSERT INTO sheet3(id,name) VALUES(1002,"Tom");
INSERT INTO sheet3(name) VALUES("lisa");
SELECT * from sheet3;
select LAST_INSERT_ID();
```

3.重置序列

删除了数据表中的多条记录，并希望对剩下数据的AUTO_INCREMENT列进行重新排列，可以通过删除自增的列，然后重新添加来实现。

默认自增序列的开始值为1。

```
-- 使用示例4
DELETE FROM sheet3 WHERE id=1002;
SELECT * from sheet3;
```

```
ALTER TABLE sheet3 DROP id;
ALTER TABLE sheet3 ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,ADD PRIMARY KEY(id);
SELECT * from sheet3;
```

4.设置序列的开始值

使用ALTER修改序列开始值，已经有的数据不受影响

```
-- 使用示例4
ALTER TABLE sheet3 AUTO_INCREMENT = 1001;
INSERT INTO sheet3(name) VALUES("lisa1");
INSERT INTO sheet3(name) VALUES("lisa2");
INSERT INTO sheet3(name) VALUES("lisa3");
SELECT * from sheet3;
```

在创建表时设置序列的开始值

```
-- 使用示例4
CREATE TABLE sheet4(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    PRIMARY KEY (id)
)engine=InnoDB AUTO_INCREMENT=1001 CHARSET=utf8;
INSERT INTO sheet4(name) VALUES("lisa1");
INSERT INTO sheet4(name) VALUES("lisa2");
INSERT INTO sheet4(name) VALUES("lisa3");
SELECT * from sheet4;
```

十四、处理重复数据

1.防止表中出现重复数据

设置指定的字段为 PRIMARY KEY（主键） 或者 UNIQUE（唯一） 索引来保证数据的唯一性。

INSERT IGNORE INTO、INSERT INTO 和REPLACE INTO

- INSERT IGNORE INTO 会忽略数据库中已经存在的数据，如果数据库没有数据，就插入新的数据，如果有数据的话就跳过这条数据。这样就可以保留数据库中已经存在数据，达到在间隙中插入数据的目的；
- INSERT IGNORE INTO 当插入数据时，在设置了记录的唯一性后，如果插入重复数据，将不返回错误，只以警告形式返回；
- REPLACE INTO 如果存在 primary 或 unique 相同的记录，则先删除掉。再插入新记录。

2.统计重复数据

```
-- 使用示例4
INSERT INTO sheet4(name) VALUES("lisa3");
INSERT INTO sheet4(name) VALUES("lisa3");
INSERT INTO sheet4(name) VALUES("lisa3");
SELECT * from sheet4;
SELECT *,COUNT(*) as namecount FROM sheet4 GROUP BY name HAVING namecount > 1;
```

3.过滤重复数据

读取不重复的数据可以在 SELECT 语句中使用 DISTINCT 关键字来过滤重复数据。

```
-- 使用示例4
SELECT DISTINCT name FROM sheet4;
SELECT DISTINCT * FROM sheet4;
```

也可以使用 GROUP BY 来读取数据表中不重复的数据

```
-- 使用示例4
SELECT * FROM sheet4 GROUP BY id;
SELECT * FROM sheet4 GROUP BY name;
```

4.删除重复数据

新建表，不重复地复制原表内容，删除原表，将新表重命名为原表

注意事项：这种方法创建地新表没有保留原表的主键约束、默认值、索引。

```
-- 使用示例4
SHOW COLUMNS FROM sheet4;
CREATE TABLE tmp SELECT * FROM sheet4  GROUP BY name;
DROP TABLE sheet4;
ALTER TABLE tmp RENAME TO sheet4;
SELECT * FROM sheet4;
SHOW COLUMNS FROM sheet4;
```

在数据表中添加 INDEX（索引） 和 PRIMAY KEY（主键）来删除表中的重复记录。

```
-- 使用示例4
INSERT INTO sheet4(id,name) VALUES(1003,"lisa3");
INSERT INTO sheet4(id,name) VALUES(1003,"lisa3");
INSERT INTO sheet4(id,name) VALUES(1003,"lisa3");
SELECT * FROM sheet4;

-- ALTER INGORE TABLE sheet4 ADD PRIMARY KEY(id,name);
-- ALTER TABLE sheet4 ADD PRIMARY KEY(id,name);
-- 此处写ingore，会产生语法错误；不写，当存在重复值时，添加主键无法成功。个人猜测，由于版本更新，此方法已经失效。(ingore忽略重复值)
```

十五、MySQL 及 SQL 注入

1.SQL注入

- 即通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令
- 如果通过网页获取用户输入的数据并将其插入一个MySQL数据库，那么就有可能发生SQL注入安全的问题。

2.防止SQL注入要点

- 1.永远不要信任用户的输入。对用户的输入进行校验，可以通过正则表达式，或限制长度；对单引号和 双"-"进行转换等。
- 2.永远不要使用动态拼装sql，可以使用参数化的sql或者直接使用存储过程进行数据查询存取。
- 3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。
- 4.不要把机密信息直接存放，加密或者hash掉密码和敏感的信息。
- 5.应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装
- 6.sql注入的检测方法一般采取辅助软件或网站平台来检测，软件一般采用sql注入检测工具jsky，网站平台就有亿思网站安全平台检测工具。MDCSOFT SCAN等。采用MDCSOFT-IPS可以有效的防御SQL注入，XSS攻击等。

十六、MySQL 导出数据

安装MySQL的时候限制了导入与导出的目录权限只能在规定的目录下才能导入。

1.secure_file_priv变量

secure_file_priv变量：导出文件地址

- secure_file_priv为NULL：禁止导出文件secure_file_priv=""
- secure_file_priv指定地址：限制导出地址只能在此secure_file_priv="D:/"
- secure_file_priv为空：导出到任意文件secure_file_priv=

查看secure-file-priv的当前值：

```
show variables like '%secure_file_priv%';
```

如何修改secure_file_priv变量：修改my.ini 配置文件中的secure_file_priv变量值

2.修改my.ini 配置文件

- my.ini 的配置文件，该文件记录了 MySQL 的所有默认配置，如端口号、默认字符集、默认存储引擎等
- 位于隐藏文件夹ProgramData中

3.配置文件、数据位置

- 显示mysql在本机中的安装位置：select @@basedir;
- 显示数据库数据存储的位置：select @@datadir;
- 同上写法2：show global variables like "%datadir%";

4.安全权限问题的解决

- 在配置文件my.ini中，设置secure-file-priv为某个路径
- net stop mysql80：关闭服务(右键以管理员身份进入cmd)
- net start mysql80：启动服务(右键以管理员身份进入cmd)
- 输入密码重写进入MySQL窗口：show variables like '%secure%';
- 成功！

5.导入数据到txt文件

使用 SELECT ... INTO OUTFILE 语句导出数据

- 输出不能是一个已存在的文件，防止文件数据被篡改。
- 在UNIX中，该文件被创建后是可读的，权限由MySQL服务器所拥有。这意味着，虽然你就可以读取该文件，但可能无法将其删除。

```
-- 使用示例3
SELECT * FROM sheet1;
SELECT * FROM sheet1 INTO OUTFILE 'E:/WorkSpace/NotebookWorkspace/sheet1.txt';
```

6.导入数据到csv文件

CSV格式，其要点包括：
(1)字段之间以逗号分隔，数据行之间以\r\n分隔；
(2)字符串以半角双引号包围，字符串本身的双引号用两个双引号表示。

参数：

- fields terminated by描述字段的分隔符，默认情况下是tab字符（\t）

- optionally enclosed by描述的是字段的括起字符。
- escaped by描述的转义字符。默认的是反斜杠（backslash：\）
- lines terminated by : 行与行之间的分隔

```
-- 使用示例3
SELECT * FROM sheet1 INTO OUTFILE 'E:/WorkSpace/NotebookWorkspace/sheet1.csv'
FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\r\n';
```

7.mysqldump的用法

注意事项：

- **mysqldump是在控制控制mysql的语句，不是mysql内部操作数据的语句!**
- 不写完整路径会拒绝访问

使用步骤：

- select @@basedir;
- cd C:\Program Files\MySQL\MySQL Server 8.0\bin
- mysqldump -u root -p student >E:/WorkSpace/NotebookWorkspace/student.sql # 此处不能有分号

几种常用语法：

- 导出整个数据库(包括数据库中的数据)
  mysqldump -u root -p student >E:/WorkSpace/NotebookWorkspace/student.sql
- 导出数据库结构（不含数据）
  mysqldump -u root -p -d student >E:/WorkSpace/NotebookWorkspace/student.sql
- 导出数据库中的某张数据表（包含数据）
  mysqldump -u root -p student sheet1 >E:/WorkSpace/NotebookWorkspace/sheet1.sql
- 导出数据库中的某张数据表的表结构（不含数据）
  mysqldump -u root -p -d student sheet1 >E:/WorkSpace/NotebookWorkspace/sheet1.sql

8.数据库备份到指定文件

备份指定数据库(以命名形式)

mysqldump -u root -p student >E:/WorkSpace/NotebookWorkspace/student.txt

备份所有数据库(以命名形式)

mysqldump -u root -p --all-databases >E:/WorkSpace/NotebookWorkspace/alldatabases.txt

9.备份数据库、表的导入

①将备份的数据库导入到MySQL服务器中

使用以下命令你需要确认数据库已经创建：

```
CREATE DATABASE studentbenfen;
```

mysql -u root -p studentbenfen <E:/WorkSpace/NotebookWorkspace/student.txt

```
use studentbenfen;
SHOW TABLES;
SHOW COLUMNS FROM sheet1;
```

②将导出的数据直接导入到远程的服务器上，但请确保两台服务器是相通的，是可以相互访问的。

使用管道来将导出的数据导入到指定的远程主机上。

```
-- 无法测试，略
```

十七、MySQL导入数据

1.mysql 命令导入

mysql -u root -p student <E:/WorkSpace/NotebookWorkspace/student.sql

2.source 命令导入

```
create database studentbenfen2;  # 创建数据库
use studentbenfen2;              # 使用已创建的数据库
set names utf8;                  # 设置编码
source E:/WorkSpace/NotebookWorkspace/student.sql;  # 导入备份数据库
-- 显示数据库studentbenfen2
use studentbenfen2;
SHOW TABLES;
SHOW COLUMNS FROM sheet1;
```

3.LOAD DATA导入数据

读取文件，将该文件中的数据插入到当前数据库的数据表中。

- 如果指定LOCAL关键词，则表明从客户主机上按路径读取文件；
- 如果没有指定，则文件在服务器上按路径读取文件。

local-infile模块：用于控制MySQL Client是否允许使用LOAD DATA LOCAL INFILE命令导入存放于客户端的数据文件。

```
-- 开启local_infile模块
show global variables like 'local_infile';
set global local_infile='ON';
```

管理员模式命名行下重启MySQL

- net stop mysql80
- net start mysql80
- LOAD DATA有时似乎必须在命名行窗口下以管理员权限进入mysql再运行：mysql -u root -p --local-infile=1

```
-- 导入数据
use studentbenfen2;
CREATE TABLE sheet2(id INT);  # sheet2表只创建了id字段，就只能复制到txt中的一个字段。
SHOW COLUMNS FROM sheet2;
LOAD DATA LOCAL INFILE 'E:WorkSpace/NotebookWorkspace/sheet1.txt' INTO TABLE sheet2;
SHOW COLUMNS FROM sheet2;
SELECT * from sheet2;
```

4.mysqlimport导入数据

mysqlimport 客户端提供了 LOAD DATA INFILEQL 语句的一个命令行接口。mysqlimport 的大多数选项直接对应 LOAD DATA INFILE 子句。

①从txt文件将数据导入到数据表中(数据表必须存在，只能备份数据)

```
-- 先清空数据表数据
delete from sheet1;
SELECT * from sheet1;
```

mysqlimport -u root -p --local studentbenfen2 E:WorkSpace/NotebookWorkspace/sheet1.txt

```
set names gbk;  # 设置编码
SELECT * from sheet1;
```

十八、运算符

MySQL 主要有以下几种运算符：

- 算术运算符：+、-、*、/或DIV(除数为0，返回结果为NULL)、%或MOD
- 比较运算符：=、<>或!=、>、<、<=、>=、BETWEEN、NOT BETWEEN、IN、NOT IN、<=>(严格判断两个NULL值是否相等)、LIKE(模糊匹配)、REGEXP或RLIKE(正则式匹配)、IS NULL、IS NOT NULL
- 逻辑运算符：NOT或!、AND、OR、XOR
- 位运算符：&、|、^(按位异或)、!(取反)、<<、>>

十九、函数

1.字符串函数

- 系列1
  - ASCII(s)：返回字符串 s 的第一个字符的 ASCII 码
  - CHAR_LENGTH(s)：返回字符串 s 的字符数(==CHARACTER_LENGTH)
  - CONCAT(s1,s2...sn)：字符串合并
  - CONCAT_WS(x, s1,s2...sn)：合并，加分隔符
  - FIELD(s,s1,s2...)：返回第一个字符串 s 在字符串列表(s1,s2...)中的位置
  - FIND_IN_SET(s1,s2)：返回在字符串s2中与s1匹配的字符串的位置
  - FORMAT(x,n)：格式化数字 "#,###.##" 形式
- 系列2
  - LCASE(s)：将字符串 s 的所有字母变成小写字母
  - LOWER(s)： 将字符串 s 的所有字母变成小写字母
  - UCASE(s)：将字符串转换为大写
  - UPPER(s)：将字符串转换为小写
  - LEFT(s,n)： 返回字符串 s 的前 n 个字符
  - RIGHT(s,n)：返回字符串 s 的后 n 个字符
  - LTRIM(s)： 去掉字符串 s 开始处的空格
  - RTRIM(s)：去掉字符串 s 结尾处的空格
  - TRIM(s)：去掉字符串 s 开始和结尾处的空格
- 系列3
  - INSERT(s1,x,len,s2)：字符串 s2 替换 s1 的 x 位置开始长度为 len 的字符串
  - LOCATE(s1,s)：从字符串 s 中获取 s1 的开始位置
  - LPAD(s1,len,s2)：在字符串 s1 的开始处填充字符串 s2，使字符串长度达到 len
  - MID(s,n,len)：从字符串 s 的 n 位置截取长度为 len 的子字符串
  - POSITION(s1 IN s)：从字符串 s 中获取 s1 的开始位置
  - REPEAT(s,n)：将字符串 s 重复 n 次
  - REPLACE(s,s1,s2)：将字符串 s2 替代字符串 s 中的字符串 s1
  - REVERSE(s)：将字符串s的顺序反过来
  - RPAD(s1,len,s2)： 在字符串 s1 的结尾处添加字符串 s2，使字符串的长度达到 len
  - SPACE(n)： 返回 n 个空格

- STRCMP(s1,s2)：比较字符串 s1 和 s2，如果 s1 与 s2 相等返回0。s1>s2 返回 1，如果 s1<s2 返回 -1
- SUBSTR(s, start, length)：从s的start位置截取长度为length的子字符串
- SUBSTRING(s, start, length)：同上
- SUBSTRING_INDEX(s, delimiter, number)：返回s的第number个分隔符delimiter之后的子串。(负为左)

2.数学函数

3.日期函数

4.高级函数

5.参考

MySQL 函数 | 菜鸟教程 (runoob.com)