数据库基础操作报告-罗宏

1.创建库和表

1.1创库

1.1.1 创库

```
CREATE DATABASE database_name;
```

```
mysql> CREATE DATABASE mylife;
Query OK, 1 row affected (0.00 sec)
```

1.1.2查询创库信息

```
SELECT CREATE DATABASE database_name;
```

```
mysql> SHOW CREATE DATABASE books;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| books    | CREATE DATABASE `books` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
```

1.2 创表

1.2.1创表

```
USE database_name;
```

```
CREATE TABLE table_name(
    -> **** ****,
    -> **** ****,
    -> );
```

```
mysql> CREATE TABLE term( number TINYINT(2), mo_course CHAR(10), tu_course CHAR(10), we_course CH
AR(10), th_course CHAR(10), fr_course CHAR(10), present ENUM('yes','no'));
Query OK, 0 rows affected (0.02 sec)
```

2.查询

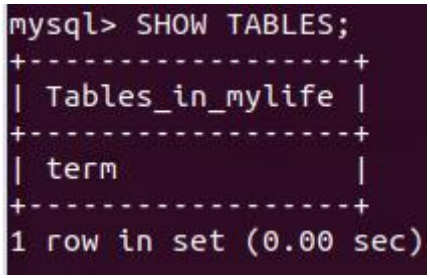
2.1查询库

```
SHOW DATABASES;
```

2.2查询表

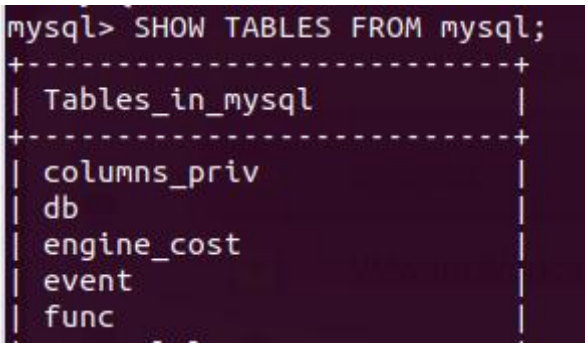
2.2.1 当前库查询

```
SHOW TABLES;
```



2.2.2 跨库查询

```
SHOW TABLES FROM database_name;
```



2.3 查询字段

```
SHOW COLUMNS FROM table_name;
```

或者

```
DESC table_name;
```

2.4查询内容

2.4.1查询所有字段内容

```
SELECT * FROM table_name;
```

2.4.2 查询指定字段内容

```
SELECT column1,column2,... FROM table_name;
```

2.4.3 补充

2.4.3.1 查询内容并新命名，但并不更改原表的字段名

```
SELECT col [AS] new_name FROM tab;
```

2.4.3.2 星号（*）表示所有列

2.4.3.3 WHERE子句能够指定查询数据

2.4.3.4 GROUP BY语句能对查询进行分组

2.4.3.5 HAVING语句用在查询分组更详细地进行（也可单独使用，功能与WHERE语句差不多，但效率不如WHERE）

2.4.3.6 ORDER BY排序语句，字段正逆序排列

```
SELECT × FROM tab ORDER BY column {ASC | DESC},...;
```

2.4.3.7 LIMIT语句限制查询

```
SELECT ... FROM tab LIMIT m,n;
```

p s : 这里m,n的意思可以理解成跳过前m条数据，查询之后n条

2.4.3.× 注意：当使用SELECT查询时，后面的扩展语句要按顺序使用，即WHERE GROUP_BY HAVING ORDER_BY LIMIT顺序进行输入

2.5 子查询(subquery | sub)

2.5.1 比较运算符修饰符

运算符	ANY	SOME	ALL
> >=	最大值	最小值	最大值
< <=	最大值	最大值	最小值
=	任意值	任意值	
<> !=			任意值

2.5.2 [NOT] IN /EXISTS引发的子查询

```
[NOT] IN (sub)
```

=ANY 与IN等效

!= ALL 或者 <>ALL 运算与NOT IN 等效

```
[NOT] EXISTS (sub)
```

如果sub返回任何行，EXISTS返回TRUE；否则为FALSE

3.字段约束

3.1主键约束

```
CREATE TABLE table_name(
    -> column1 AUTO_INCREMENT PRIMARY KEY,
    -> column2 ...
    -> );
```

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| user  | varchar(10) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

p s : AUTO_INCREMENT必须和PRIMARY KEY混合使用，但PRIMARY KEY可单独使用

（其中AUTO_INCREMENT功能为递增主键）

3.2 唯一键约束

```
CERATE TABLE table_name(
    -> column1 INT AUTO_INCREMENT PRIMARY KEY,
    -> column2 UNIQUE KEY,
    -> column3 UNIQUE KEY
    -> ...);
```

```
mysql> CREATE TABLE t2(
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> user CHAR(10) UNIQUE KEY,
    -> password CHAR(10) UNIQUE KEY
    -> );
```

```
+----------+----------+------+-----+---------+----------------+
| Field    | Type     | Null | Key | Default | Extra          |
+----------+----------+------+-----+---------+----------------+
| id       | int(11)  | NO   | PRI | NULL    | auto_increment |
| user     | char(10) | YES  | UNI | NULL    |                |
| password | char(10) | YES  | UNI | NULL    |                |
+----------+----------+------+-----+---------+----------------+
```

```
+----+-------+----------+
| id | user  | password |
+----+-------+----------+
|  1 | xihua | 15648465 |
+----+-------+----------+
1 row in set (0.00 sec)

mysql> INSERT INTO t2 VALUES(NULL,'xihuo','15648465');
ERROR 1062 (23000): Duplicate entry '15648465' for key 'password'
mysql> INSERT INTO t2 VALUES(NULL,'xihua','15645865');
ERROR 1062 (23000): Duplicate entry 'xihua' for key 'user'
mysql> INSERT INTO t2 VALUES(NULL,'xihuo','15645865');
Query OK, 1 row affected (0.00 sec)
```

3.3 外键约束

3.3.1

```
CERATE TABLE table_name(
    -> column1 AUTO_INCREMENT PRIMARY KEY,
    -> column2 UNIQUE KEY,
    -> column3 ,...
    -> FOREIGN KEY (column) REFERENCES table_name(column)
    -> );
```

```
mysql> CREATE TABLE t3(
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> userS CHAR(10),
    -> FOREIGN KEY (userS) REFERENCES t2 (user)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

```
SHOW CREATE TABLE table_name\G;
```

```
       Table: t3
Create Table: CREATE TABLE `t3` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `userS` char(10) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `userS` (`userS`),
  CONSTRAINT `t3_ibfk_1` FOREIGN KEY (`userS`) REFERENCES `t2` (`user`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

p s : 上面红标所指为外键索引

** 查看索引命令

```
SHOW INDEXES FROM table_name;
```

3.3.2 其他扩展项

3.3.2.1 CASCADE

CASCADE：父表作删除或更新，子表中相对应的行也作同样操作

```
CREATE TABLE tab_name(
    -> column _KEY ON DELETE | UPDATE CASCADE
    -> );
```

3.3.2.2 SET NULL

SET NULL：父表作删除或更新，子表对应外键设置为NULL（）前提是子表列没有指定NOT NULL

3.3.2.3 RESTRICT和NO ACTION

上述两者都为拒绝对父表作删除或更新

4.修改表

4.1 修改约束

4.1.1 添加/删除默认约

```
mysql> ALTER TABLE tab_name ALTER [COLUMN] col_name {SET DEFAULT **| DROP
DEFAULT};
```

4.1.2 删除主键/唯一/外键约束

```
ALTER TABLE tab_name DROP PRIMARY KEY
      /{INDEX | KEY} index_name
      /FOREGIN KEY fk_name;
```

4.2 修改列

4.2.1修改定义

```
ALTER TABLE tab_name MODIFY col_name col_defination [FIRST | AFTER col_name];
```

p s : col_defination 为想修改成的新定义，后面[]里的是移动字段在表中的位置

4.2.2 修改名称

```
ALTER TABLE tab_name CHANGE old_name new_name col_defination [FIRST |AFTER
col_name];
```

4.3 修改表

```
ALTER TABLE tab_name RENAME [TOIAS] new_name;
```

或者

```
RENAME TABLE tab_name TO new_name [,tab_name2 TO new_name2...];
```

5.插入数据INSERT

```
INSERT [INTO] TABLE tab_name(column1...) {VALUE | VALUES} (.....),(....,),...;
```

或者

```
INSERT [INTO] TABLE tab_name SET column1=... ,column2=... ,.....;
```

#通过下面语句能将查询结果插入到表中

```
INSERT tab SELECT...;
```

6.更新记录UPDATE

6.1 单表更新

```
UPDATE tab_name SET ....WHERE....;
```

6.2 多表更新

```
UPDATE tab_ref SET col1={expr1 | DEFAULT}
[,col2={expr2 | DEFAULT}]...[WHERE where];
```

6.3 多表更新一步到位

```
CREATE TABLE tab [(create_defination,....)]
select_statement;
```

7.删除记录DELETE

7.1 单表删除

```
DELETE FROM tab_name [WHERE...]
```

7.2 多表删除

```
DELETE tab1[.*][tab2[.*]...]...FROM tab_refe
[WHERE where];
```

8.连接

8.1 SELECT语句，多表更新/删除语句支持JOIN

语法结构

```
tab_ref1 [[AS] alias1] {[INNER | CROSS] JOIN | {LEFT | RIGHT} [OUTER] JOIN}
tab_ref2 [[AS] alias2] ON condition;
```

- alias意为：别名
- JOIN / CROSS JOIN / INNER JOIN 等价
- LEFT [OUTER] JOIN 左外连接（显示左表所有数据）
- RIGHT [OUTER] JOIN 右外连接（显示右表所有数据）

8.2 连接条件

ON / WHERE 可用来设定连接条件

通常使用ON设定连接条件

使用WHERE进行结果过滤

8.3 自身连接查询

参考8.1