基本操作

```
/* Windows服务 */
-- 启动MySQL
    net start mysql
-- 创建Windows服务
    sc create mysql binPath= mysqld_bin_path(注意：等号与值之间有空格)
/* 连接与断开服务器 */
mysql -h 地址 -P 端口 -u 用户名 -p 密码
SHOW PROCESSLIST -- 显示哪些线程正在运行
SHOW VARIABLES -- 显示系统变量信息
```

数据库操作

```
-- 查看当前数据库
    SELECT DATABASE();
-- 显示当前时间、用户名、数据库版本
    SELECT now(), user(), version();
-- 创建库
    CREATE DATABASE[ IF NOT EXISTS] 数据库名 数据库选项
    数据库选项：
        CHARACTER SET charset_name
        COLLATE collation_name
-- 查看已有库
    SHOW DATABASES[ LIKE 'PATTERN']
-- 查看当前库信息
    SHOW CREATE DATABASE 数据库名
-- 修改库的选项信息
    ALTER DATABASE 库名 选项信息
-- 删除库
    DROP DATABASE[ IF EXISTS] 数据库名
        同时删除该数据库相关的目录及其目录内容
```

表的操作

```
-- 创建表
    CREATE [TEMPORARY] TABLE[ IF NOT EXISTS] [库名.]表名 ( 表的结构字段定义 )[ 表选项]
```

每个字段必须有数据类型　最后一个字段后不能有逗号　字段不能加引号

TEMPORARY为临时表，会话结束时表自动消失

IF NOT EXISTS是语句的可选子句。可防止创建数据库服务器中已存在的新数据库的错误。

字段定义：

```
        字段名 数据类型 [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'string']
eg.CREATE TABLE IF NOT EXISTS runoob_tbl(
   runoob_id INT UNSIGNED AUTO_INCREMENT,
   runoob_title VARCHAR(100) NOT NULL,
   runoob_author VARCHAR(40) NOT NULL,
   submission_date DATE,
   PRIMARY KEY ( runoob_id )
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

//PRIMARY KEY关键字用于定义列为主键。 您可以使用多列来定义主键，列间以逗号分隔。

```
-- 表选项
    -- 字符集
        CHARSET = charset_name
        如果表没有设定，则使用数据库字符集
    -- 存储引擎
        ENGINE = engine_name
        表在管理数据时采用的不同的数据结构，结构不同会导致处理方式、提供的特性操作等不同
        常见的引擎：InnoDB MyISAM Memory/Heap BDB Merge Example CSV MaxDB Archive
        不同的引擎在保存表的结构和数据时采用不同的方式
        MyISAM表文件含义：.frm表定义，.MYD表数据，.MYI表索引
        InnoDB表文件含义：.frm表定义，表空间数据和日志文件
        SHOW ENGINES -- 显示存储引擎的状态信息
        SHOW ENGINE 引擎名 {LOGS|STATUS} -- 显示存储引擎的日志或状态信息
    -- 自增起始数
        AUTO_INCREMENT = 行数     //定义自增属性，一般用于主键，自动加一
    -- 数据文件目录
        DATA DIRECTORY = '目录'
    -- 索引文件目录
```

```
        INDEX DIRECTORY = '目录'
    -- 表注释
        COMMENT = 'string'
    -- 分区选项
        PARTITION BY ... (详细见手册)

-- 查看所有表
    SHOW TABLES[ LIKE 'pattern']
    SHOW TABLES FROM  库名
-- 查看表机构
    SHOW CREATE TABLE 表名  (信息更详细)
    DESC 表名 / DESCRIBE 表名 / EXPLAIN 表名 / SHOW COLUMNS FROM 表名 [LIKE 'PATTERN']
    SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
-- 查看表字段
/*SHOW COLUMNS FROM testalter_tbl;*/
-- 修改表
    -- 修改表本身的选项
       ALTER TABLE 表名 表的选项
        eg: ALTER TABLE 表名 ENGINE=MYISAM;
    -- 对表进行重命名
        RENAME TABLE 原表名 TO 新表名
        RENAME TABLE 原表名 TO 库名.表名  (可将表移动到另一个数据库)
        -- RENAME可以交换两个表名
    -- 修改表的字段机构(13.1.2. ALTER TABLE语法)
       ALTER TABLE 表名 操作名


        -- 操作名
            ADD[ COLUMN] 字段定义        -- 增加字段
                AFTER 字段名            -- 表示增加在该字段名后面
                FIRST                    -- 表示增加在第一个
             ADD PRIMARY KEY(字段名)    -- 创建主键
             ADD UNIQUE [索引名] (字段名)-- 创建唯一索引
             ADD INDEX [索引名] (字段名) -- 创建普通索引
```

```
            DROP[ COLUMN] 字段名         -- 删除字段
            MODIFY[ COLUMN] 字段名 字段属性      -- 支持对字段属性进行修改，不能修改字段名(所有原有属性也需写上)
            CHANGE[ COLUMN] 原字段名 新字段名 字段属性       -- 支持对字段名修改
            DROP PRIMARY KEY     -- 删除主键(删除主键前需删除其AUTO_INCREMENT属性)
            DROP INDEX 索引名 -- 删除索引
            DROP FOREIGN KEY 外键     -- 删除外键
-- 删除表
    DROP TABLE[ IF EXISTS] 表名 ...
-- 清空表数据
    TRUNCATE [TABLE] 表名
-- 复制表结构
    CREATE TABLE 表名 LIKE 要复制的表名
-- 复制表结构和数据
    CREATE TABLE 表名 [AS] SELECT * FROM 要复制的表名
-- 检查表是否有错误
    CHECK TABLE tbl_name [, tbl_name] ... [option] ...
-- 优化表
    OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
-- 修复表
    REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
-- 分析表
    ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

数据操作

```
/* 数据操作 */ ----------------
-- 增
```

```
    INSERT [INTO] 表名 [(字段列表)] VALUES (值列表)[, (值列表), ...]
        -- 如果要插入的值列表包含所有字段并且顺序一致，则可以省略字段列表。
        -- 可同时插入多条数据记录!
        REPLACE 与 INSERT 完全一样，可互换。
    INSERT [INTO] 表名 SET 字段名=值[, 字段名=值, ...]
-- 查
    SELECT 字段列表 FROM 表名[ 其他子句]
        -- 可来自多个表的多个字段
        -- 其他子句可以不使用
eg.
SELECT
    column_1, column_2, ...
FROM
    table_1
[INNER | LEFT |RIGHT] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1
LIMIT offset, length;
```

- JOIN根据某些连接条件从其他表中获取数据。

BETWEEN选择在给定范围值内的值。

LIKE匹配基于模式匹配的值。

IN指定值是否匹配列表中的任何值。

```
-- 删
    DELETE FROM 表名[ 删除条件子句]
        没有条件子句，则会删除全部
```

表中的行顺序未指定，当使用LIMIT子句时，应始终使用ORDER BY子句：

```
DELETE FROM table_name
ORDER BY c1, c2, ...
LIMIT row_count;
```

```
-- 改
```

UPDATE 表名 SET 字段名=新值[, 字段名=新值] [更新条件]

```
eg.UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
field1=new-value1, field2=new-value2
[WHERE Clause]
```

*GNORE*修饰符忽视导致错误(如重复键冲突)的行，不会更新。

列属性(列约束)

```
/* 列属性（列约束） */ ------------------
1. PRIMARY 主键
    - 能唯一标识记录的字段，可以作为主键,不能为null。
    - 一个表只能有一个主键。主键具有唯一性。
    - 声明字段时，用 primary key 标识。 也可以在字段列表之后声明
            例：create table tab ( id int, stu varchar(10), primary key (id));
    - 主键可以由多个字段共同组成。此时需要在字段列表后声明的方法。
        例：create table tab ( id int, stu varchar(10), age int, primary key
(stu, age));
2. UNIQUE 唯一索引（唯一约束）
    使得某字段的值也不能重复。
3. NULL 约束
    null不是数据类型，是列的一个属性。
    表示当前列是否可以为null，表示什么都没有。
    null, 允许为空。默认。
    not null, 不允许为空。
    insert into tab values (null, 'val');
        -- 此时表示将第一个字段的值设为null, 取决于该字段是否允许为null
4. DEFAULT 默认值属性
    当前字段的默认值。
    insert into tab values (default, 'val');    -- 此时表示强制使用默认值。
    create table tab ( add_time timestamp default current_timestamp );
        -- 表示将当前时间的时间戳设为默认值。
        current_date, current_time
5. AUTO_INCREMENT 自动增长约束
    自动增长必须为索引（主键或unique）
```

只能存在一个字段为自动增长。

默认为1开始自动增长。可以通过表属性 auto_increment = x进行设置，或 alter table tbl auto_increment = x;

6. COMMENT 注释

例：create table tab (id int) comment '注释内容';

7. FOREIGN KEY 外键约束

用于限制主表与从表数据完整性。

```
alter table t1 add constraint `t1_t2_fk` foreign key (t1_id) references t2(id);
    -- 将表t1的t1_id外键关联到表t2的id字段。
    -- 每个外键都有一个名字，可以通过 constraint 指定
```

存在外键的表，称之为从表（子表），外键指向的表，称之为主表（父表）。

作用：保持数据一致性，完整性，主要目的是控制存储在外键表（从表）中的数据。

MySQL中，可以对InnoDB引擎使用外键约束：

语法：

```
foreign key (外键字段) references 主表名 (关联字段) [主表记录删除时的动作] [主表记录更新时的动作]
```

此时需要检测一个从表的外键需要约束为主表的已存在的值。外键在没有关联的情况下，可以设置为null.前提是该外键列，没有not null。

可以不指定主表记录更改或更新时的动作，那么此时主表的操作被拒绝。

如果指定了 on update 或 on delete：在删除或更新时，有如下几个操作可以选择：

1. cascade，级联操作。主表数据被更新（主键值更新），从表也被更新（外键值更新）。主表记录被删除，从表相关记录也被删除。

2. set null，设置为null。主表数据被更新（主键值更新），从表的外键被设置为null。主表记录被删除，从表相关记录外键被设置成null。但注意，要求该外键列，没有not null属性约束。

3. restrict，拒绝父表删除和更新。

注意，外键只被InnoDB存储引擎所支持。其他引擎是不支持的。

SELECT

```
/* SELECT */ ------------------
SELECT [ALL|DISTINCT] select_expr FROM -> WHERE -> GROUP BY [合计函数] -> HAVING -> ORDER BY -> LIMIT
a. select_expr
    -- 可以用 * 表示所有字段。
        select * from tb;
    -- 可以使用表达式（计算公式、函数调用、字段也是个表达式）
        select stu, 29+25, now() from tb;
    -- 可以为每个列使用别名。适用于简化列标识，避免多个列标识符重复。
        - 使用 as 关键字，也可省略 as.
        select stu+10 as add10 from tb;
b. FROM 子句
```

用于标识查询来源。

```
-- 可以为表起别名。使用as关键字。
    SELECT * FROM tb1 AS tt, tb2 AS bb;
-- from子句后，可以同时出现多个表。
    -- 多个表会横向叠加到一起，而数据会形成一个笛卡尔积。
    SELECT * FROM tb1, tb2;
-- 向优化符提示如何选择索引
    USE INDEX、IGNORE INDEX、FORCE INDEX
    SELECT * FROM table1 USE INDEX (key1,key2) WHERE key1=1 AND key2=2 AND key3=3;
    SELECT * FROM table1 IGNORE INDEX (key3) WHERE key1=1 AND key2=2 AND key3=3;
```

c. WHERE 子句

```
-- 从from获得的数据源中进行筛选。
-- 整型1表示真，0表示假。
-- 表达式由运算符和运算数组成。
    -- 运算数：变量（字段）、值、函数返回值
    -- 运算符：
        =, <=>, <>, !=, <=, <, >=, >, !, &&, ||,
        in (not) null, (not) like, (not) in, (not) between and, is (not), and, or, not, xor
        is/is not 加上ture/false/unknown，检验某个值的真假
        <=>与<>功能相同，<=>可用于null比较
```

d. GROUP BY 子句，分组子句

```
GROUP BY 字段/别名 [排序方式]
分组后会进行排序。升序：ASC，降序：DESC
以下[合计函数]需配合 GROUP BY 使用：
count 返回不同的非NULL值数目  count(*)、count(字段)
sum 求和
max 求最大值
min 求最小值
avg 求平均值
group_concat 返回带有来自一个组的连接的非NULL值的字符串结果。组内字符串连接。
```

e. HAVING 子句，条件子句

```
与 where 功能、用法相同，执行时机不同。
where 在开始时执行检测数据，对原数据进行过滤。having 对筛选出的结果再次进行过滤。
having 字段必须是查询出来的，where 字段必须是数据表存在的。
where 不可以使用字段的别名，having 可以。因为执行WHERE代码时，可能尚未确定列值。
where 不可以使用合计函数。一般需用合计函数才会用 having
SQL标准要求HAVING必须引用GROUP BY子句中的列或用于合计函数中的列。
```

f. ORDER BY 子句，排序子句

```
order by 排序字段/别名 排序方式 [,排序字段/别名 排序方式]...
升序：ASC，降序：DESC
支持多个字段的排序。
```

g. LIMIT 子句，限制结果数量子句

```
仅对处理好的结果进行数量限制。将处理好的结果的看作是一个集合，按照记录出现的顺序，索引从0开始。
limit 起始位置，获取条数
省略第一个参数，表示从索引0开始。limit 获取条数
```

```
h. DISTINCT, ALL 选项
    distinct 去除重复记录
    默认为 all, 全部记录
```

子查询

```
/* 子查询 */ ----------------
    - 子查询需用括号包裹。
-- from型
    from后要求是一个表，必须给子查询结果取个别名。
    - 简化每个查询内的条件。
    - from型需将结果生成一个临时表格，可用以原表的锁定的释放。
    - 子查询返回一个表，表型子查询。
    select * from (select * from tb where id>0) as subfrom where id>1;
-- where型
    - 子查询返回一个值，标量子查询。
    - 不需要给子查询取别名。
    - where子查询内的表，不能直接用以更新。
    select * from tb where money = (select max(money) from tb);
    -- 列子查询
        如果子查询结果返回的是一列。
        使用 in 或 not in 完成查询
        exists 和 not exists 条件
            如果子查询返回数据，则返回1或0。常用于判断条件。
            select column1 from t1 where exists (select * from t2);
    -- 行子查询
        查询条件是一个行。
        select * from t1 where (id, gender) in (select id, gender from t2);
        行构造符：(col1, col2, ...) 或 ROW(col1, col2, ...)
        行构造符通常用于与对能返回两个或两个以上列的子查询进行比较。
    -- 特殊运算符
    != all()    相当于 not in
    = some()    相当于 in。any 是 some 的别名
    != some()   不等同于 not in，不等于其中某一个。
    all, some 可以配合其他运算符一起使用。
```

LIKE

```
SELECT field1, field2,...fieldN
FROM table_name
WHERE field1 LIKE condition1 [AND [OR]] filed2 = 'somevalue'
'%a'     //以a结尾的数据
'a%'     //以a开头的数据
'%a%'    //含有a的数据
'_a_'    //三位且中间字母是a的
```

```
'_a'      //两位且结尾字母是a的
'a_'      //两位且开头字母是a的
```

UNION

将多个select查询的结果组合成一个结果集合。 建议对每个SELECT查询加上小括号包裹。
SELECT ... UNION [ALL|DISTINCT] SELECT ...
默认 DISTINCT 方式，即删除重复数据
排序按第一个搜索的目标，ORDER BY 排序时，需加上 LIMIT 进行结合。
需要各select查询的字段数量一样。
每个select查询的字段列表(数量、类型)应一致，因为结果中的字段名以第一条select语句为准。

```
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
UNION [ALL | DISTINCT]
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
ORDER BY 列名称;
```

连接查询(join)

```
/* 连接查询(join) */ ------------------
    将多个表的字段进行连接，可以指定连接条件。
-- 内连接(inner join)
    - 默认就是内连接，可省略inner。
    - 只有数据存在时才能发送连接。即连接结果不能出现空行。
    on 表示连接条件。其条件表达式与where类似。也可以省略条件（表示条件永远为真）
    也可用where表示连接条件。
    还有 using，但需字段名相同。 using(字段名)
    -- 交叉连接 cross join
        即，没有条件的内连接。
        select * from tb1 cross join tb2;
-- 外连接(outer join)
    - 如果数据不存在，也会出现在连接结果中。
    -- 左外连接 left join
        如果数据不存在，左表记录会出现，而右表为null填充
    -- 右外连接 right join
        如果数据不存在，右表记录会出现，而左表为null填充
-- 自然连接(natural join)
    自动判断连接条件完成连接。
    相当于省略了using，会自动查找相同字段名。
    natural join
    natural left join
    natural right join
```

```
select info.id, info.name, info.stu_num, extra_info.hobby, extra_info.sex from info,
extra_info where info.stu_num = extra_info.stu_id;
```

- **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。**
- **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。**

```
mysql> SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM
runoob_tbl a INNER JOIN tcount_tbl b ON a.runoob_author =
b.runoob_author;
```

等同

```
mysql> SELECT a.runoob_id, a.runoob_author, b.runoob_count FROM
runoob_tbl a, tcount_tbl b WHERE a.runoob_author = b.runoob_author;
```

正则表达式

REGEXP 操作符来进行正则表达式匹配。

模式	描述
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。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，请使用象 '[.\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' 则匹配 "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 是一个非负整数。匹配确定的 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 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。

外键约束

外键表示一个表中的一个字段被另一个表中的一个字段引用。表可以有多个外键，子表中的每个外键可能引用不同的父表。

主　　键：

数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。**一个数据列只能有一个主键**，且主键的取值不能缺失，即不能为空值（Null）。

超　　键：

在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。**超键包含候选键和主键。**

候选键：

是**最小超键**，即没有冗余元素的超键。

外　　键：

在一个表中存在的**另一个表的主键**称此表的外键。