MYSQL基础

- 数据库概述
  - 好处：实现数据持久化，数据库存储量大，可存数据类型多
  - 相关概念
    - MYSQL属于DBMS

  - 关系型数据库设计规则
    - 数据库中的表名具有唯一性
    - 表、记录、字段：表与表之间的关系用ER模型表示

    - 表的关联关系
      - 一对一（字段过多可拆成常用、不常用两张表）
      - 一对多
      - 多对多：必须创建中间表来连接
      - 自我引用
  - SQL书写规范
    - 字符串型和日期时间类型的数据可以使用单引号（''）表示
    - 列的别名，尽量使用双引号（""），不建议省略as
    - 关键字大写，表明字段名小写

- windows大小写不敏感，linux大小写敏感
- 注释结构

```
单行注释：#注释文字(MySQL特有的方式)
单行注释：-- 注释文字(--后面必须包含一个空格。)
多行注释：/* 注释文字  */
```

- 分类
  - **DDL（Data Definition Languages、数据定义语言）**，这些语句定义了不同的数据库、表、视图、索引等数据库对象，还可以用来创建、删除、修改数据库和数据表的结构。
    - 主要的语句关键字包括 `CREATE` 、 `DROP` 、 `ALTER` 等。
  - **DML（Data Manipulation Language、数据操作语言）**，用于添加、删除、更新和查询数据库记录，并检查数据完整性。
    - 主要的语句关键字包括 `INSERT` 、 `DELETE` 、 `UPDATE` 、 `SELECT` 等。
    - **SELECT是SQL语言的基础，最为重要。**
  - **DCL（Data Control Language、数据控制语言）**，用于定义数据库、表、字段、用户的访问权限和安全级别。
    - 主要的语句关键字包括 `GRANT` 、 `REVOKE` 、 `COMMIT` 、 `ROLLBACK` 、 `SAVEPOINT` 等。
- 登陆mysql：mysql -uroot -p，exit退出

————————————————————使用————————————————————

- DML：增删改查，排序分组
  - SELECT
    - 全选：SELECT*最好不用
    - 去重：SELECT DISTINCT sno
    - 限制：LIMIT 返回前 5 行
      - `SELECT * FROM mytable LIMIT 5;`
      - `SELECT * FROM mytable LIMIT 0, 5;`
    - 子查询：在WHERE 子句中添加，可多层嵌套，在父查询之前执行
    - **where**
      - 用于过滤记录；在SELECT/UPDATE/DELETE中使用；后跟一个返回 true 或 false 的条件。
      - 可以在 WHERE 子句中使用的操作符

运算符	描述
=	等于
<>	不等于。注释：在 SQL 的一些版本中，该操作符可被写成 !=
>	大于
<	小于
>=	大于等于
<=	小于等于
BETWEEN	在某个范围内
LIKE	搜索某种模式
IN	指定针对某个列的多个可能值

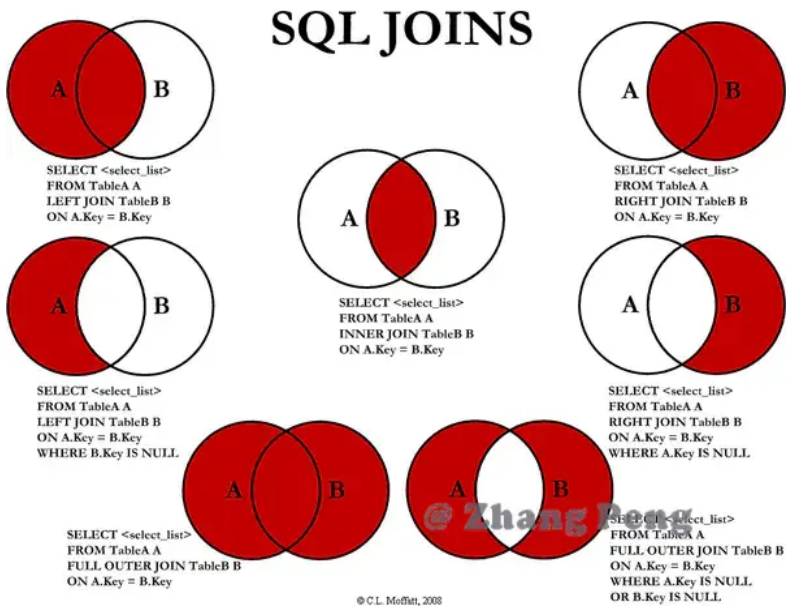
- **IN**
  - 在指定的几个特定值中任选一个值
  - `SELECT *FROM products`
  - `WHERE vend_id IN ('DLL01', 'BRS01');`
- **BETWEEN...AND**
  - 选取介于某个范围内的值
  - `SELECT *FROM products`
  - `WHERE prod_price BETWEEN 3 AND 5;`
- **AND、OR、NOT**
  - `SELECT *FROM products`
  - `WHERE prod_price NOT BETWEEN 3 AND 5;`
- **LIKE**
  - **不要滥用通配符，通配符位于开头处匹配会非常慢。**
  - **% 表示任何字符出现任意次数。**
  - **_ 表示任何字符出现一次。**
  - **% 示例**
    - `SELECT prod_id, prod_name, prod_price`
    - `FROM products`
    - `WHERE prod_name LIKE '%bean bag%';`

- **_示例**
  - `SELECT prod_id, prod_name, prod_price`
  - `FROM products`
  - `WHERE prod_name LIKE '__ inch teddy bear';`
- INSERT INTO
  - 插入完整的行
    - `INSERT INTO user`
    - `VALUES (10, 'root', 'root', 'xxxx@163.com');`
  - 插入行的一部分
    - `INSERT INTO user(username, password, email)`
    - `VALUES ('admin', 'admin', 'xxxx@163.com');`
  - 插入查询出来的数据
    - `INSERT INTO user(username)`
    - `SELECT name`
    - `FROM account;`
- UPDATE
  - `UPDATE user`
  - `SET username='robot', password='robot'`
  - `WHERE username = 'root';`
- DELETE
  - DELETE删除表内容，DROP删除整个表结构，属于DDL
  - 删除表中的指定数据
    - `DELETE FROM user`
    - `WHERE username = 'robot';`

- 清空表中的数据
  - `DROP TABLE user;`
- ORDER BY
  - ASC：升序（默认），DESC：降序
  - 可以按多个列进行排序，并且为每个列指定不同的排序方式
  - 指定多个列的排序方向
    - `SELECT * FROM productsORDER BY prod_price DESC, prod_name ASC;`
- GROUP BY...HAVING
  - HAVING过滤针对行，WHERE过滤针对分组
  - 使用 WHERE 和 HAVING 过滤数据
    - `SELECT cust_name, COUNT(*) AS num`
    - `FROM Customers`
    - `WHERE cust_email IS NOT NULL`
    - `GROUP BY cust_name`
    - `HAVING COUNT(*) >= 1;`
  - 分组后排序
    - `SELECT cust_name, COUNT(cust_address) AS addr_num`
    - `FROM Customers GROUP BY cust_name`
    - `ORDER BY cust_name DESC;`
- 多表查询：连接和组合
  - 连接（JOIN...ON）
    - 概述
      - X*Y**笛卡尔积**：X和Y的所有可能组合，组合个数为乘积数

- 使用**别名**可以简化查询，提高查询效率，在FROM中规定表的别名，因为MYSQL从这里开始执行
- 连接N个表需要N-1个条件

- **内连接（INNER JOIN）、左连接（LEFT JOIN）、右连接（RIGHT JOIN）、全外连接（OUTER JOIN）**
  - MYSQL不支持全外连接



  - 自连接：是内连接的一种
    - `SELECT c1.cust_id, c1.cust_name, c1.cust_contact`
    - `FROM customers c1, customers c2`
    - `WHERE c1.cust_name = c2.cust_name`
    - `AND c2.cust_contact = 'Jim Jones';`
- **自然连接（NATURAL JOIN）**相当于JOIN加条件，自动查询同名列，并等值连接

- 组合（UNION）
  - 概述
    - UNION 运算符**将两个或更多查询的结果组合起来**，并生成一个结果集
    - UNION 基本规则
      - 所有查询的**列数和列顺序必须相同。**

        - 每个查询中涉及表的列的**数据类型必须相同或兼容。**
        - 通常**返回的列名取自第一个查询。**
    - **默认会去除相同行**，如果需要保留相同行，使用 UNION ALL。
    - **只能包含一个 ORDER BY 子句**，并且必须位于语句的最后。
    - 应用场景
        - 在一个查询中从不同的表返回结构数据。
        - 对一个表执行多个查询，按一个查询返回数据。
- 组合查询
    - `SELECT cust_name, cust_contact, cust_email`
    - `FROM customers`
    - `WHERE cust_state IN ('IL', 'IN', 'MI')`
    - `UNION`
    - `SELECT cust_name, cust_contact, cust_email`
    - `FROM customers`
    - `WHERE cust_name = 'Fun4All';`
- JOIN vs UNION
    - **JOIN 中连接表的列可能不同**，但在 UNION 中，所有查询的列数和列顺序必须相同。
    - UNION 将查询之后的行放在一起（垂直放置），但 JOIN 将查询之后的列放在一起（水平放置），即它构成一个笛卡尔积。
- 函数
    - 文本处理

- 图示

函数	说明
LEFT()、RIGHT()	左边或者右边的字符
LOWER()、UPPER()	转换为小写或者大写
LTRIM()、RTIM()	去除左边或者右边的空格
LENGTH()	长度
SOUNDEX()	转换为语音值

- 其中， **SOUNDEX()** 可以将一个字符串转换为描述其语音表示的字母数字模式。
  - `SELECT *`
  - `FROM mytable`
  - `WHERE SOUNDEX(col1) = SOUNDEX('apple')`

- 日期和时间处理
  - 日期格式：YYYY-MM-DD
  - 时间格式：HH:MM:SS
  - 图示

函 数	说 明
AddDate()	增加一个日期（天、周等）
AddTime()	增加一个时间（时、分等）
CurDate()	返回当前日期
CurTime()	返回当前时间
Date()	返回日期时间的日期部分
DateDiff()	计算两个日期之差
Date_Add()	高度灵活的日期运算函数
Date_Format()	返回一个格式化的日期或时间串
Day()	返回一个日期的天数部分
DayOfWeek()	对于一个日期，返回对应的星期几
Hour()	返回一个时间的小时部分
Minute()	返回一个时间的分钟部分
Month()	返回一个日期的月份部分
Now()	返回当前日期和时间
Second()	返回一个时间的秒部分
Time()	返回一个日期时间的时间部分
Year()	返回一个日期的年份部分

  - 代码示例：
    - `mysql> SELECT NOW();`

- 2018-4-14 20:25:11
- 数值处理
  - 图示

函数	说明
SIN()	正弦
COS()	余弦
TAN()	正切
ABS()	绝对值
SQRT()	平方根
MOD()	余数
EXP()	指数
PI()	圆周率
RAND()	随机数

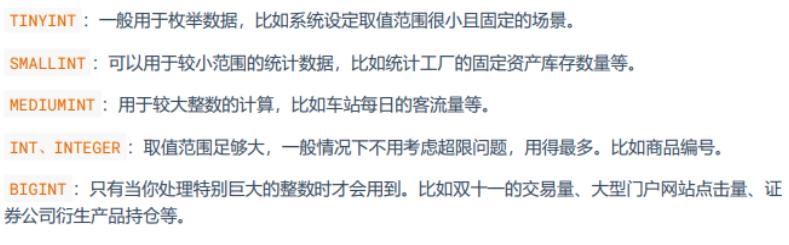
- 汇总
  - 图示

函 数	说 明
AVG ()	返回某列的平均值
COUNT ()	返回某列的行数
MAX ()	返回某列的最大值
MIN ()	返回某列的最小值
SUM ()	返回某列值之和

  - 使用 DISTINCT 可以让汇总函数值汇总不同的值。
    - `SELECT AVG(DISTINCT col1) AS avg_col`
    - `FROM mytable`
- 数据类型：创建表时数据类型不合理会影响精度和性能
  - 整型
    - 一个字节8bit，INT一般都是4字节，声明时数值范围不能越界

整数类型	字节	有符号数取值范围	无符号数取值范围
TINYINT	1	-128~127	0~255
SMALLINT	2	-32768~32767	0~65535
MEDIUMINT	3	-8388608~8388607	0~16777215
INT、INTEGER	4	-2147483648~2147483647	0~4294967295
BIGINT	8	-9223372036854775808~9223372036854775807	0~18446744073709551615

- 5.7版本中，会显示数据范围宽度，TINYINT(4)，加上符号宽度为4
- ————————属性————————————————————————————
- UNSIGNED 无符号正数
- INT(5) ZEROFILL # 宽度不足5位用0填充
- ————————使用————————————————————————————
- 首先确保数据不会超过范围，再考虑存储空间带来的成本增加



- 小数 FLOAT/DOUBLE/DECIMAL
  - 浮点型：FLOAT四字节，DOUBLE八字节
  - 定点型：DECIMAL(5,2) 小数点后两位，整个数必须是五位，默认(10,0)，以字符串存储，必然精确
  - 浮点类型范围大，但会有精度损失，数据很长会四舍五入，避免用=判断两个浮点数是否相等，追求精度推荐DECIMAL，但超出范围也会四舍五入
- 位类型BIT
  - BIT默认一位，存一个二进制值
  - BIT(5)存五位

二进制字符串类型	长度	长度范围	占用空间
BIT(M)	M	1 <= M <= 64	约为(M + 7)/8个字节

- 日期
  - **实际项目中，DATETIME使用最多，取值范围也最大，但方便计算最好使用TIMESTAMP**
  - 类型 | 名称 | 字节 | 日期格式 | 最小值 | 最大值

YEAR	年	1	YYYY或YY	1901	2155
TIME	时间	3	HH:MM:SS	-838:59:59	838:59:59
DATE	日期	3	YYYY-MM-DD	1000-01-01	9999-12-03
DATETIME	日期 时间	8	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00	9999-12-31 23:59:59
TIMESTAMP	日期 时间	4	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00 UTC	2038-01-19 03:14:07UTC

- TIME可以表示时间间隔，所以范围超过24
- DATE中，年份取00到69，对应2000到2069，年份取70到99，对应1970到1999
- TIME前面可以写天数，会换算成小时

```
INSERT INTO test_time1
VALUES('2 12:30:29'), ('12:35:29'), ('12:40'), ('2 12:40'),('1 05'), ('45');
```

- 不同版本会不一样，这里是MYSQL80
- TIMESTAMP在不同时区不同，DATETIME固定不变

- 文本字符串
  - 概览

文本字符串类型	值 的 长 度	长 度 范 围	占用的存储空间
CHAR(M)	M	0 <= M <= 255	M个字节
VARCHAR(M)	M	0 <= M <= 65535	M+1个字节
TINYTEXT	L	0 <= L <= 255	L+2个字节
TEXT	L	0 <= L <= 65535	L+2个字节
MEDIUMTEXT	L	0 <= L <= 16777215	L+3个字节
LONGTEXT	L	0 <= L <= 4294967295	L+4个字节
ENUM	L	1 <= L <= 65535	1或2个字节
SET	L	0 <= L <= 64	1,2,3,4或8个字节

  - CHAR和VARCHAR
    - CHAR可以不指定长度，默认一个字符，若存的值小，会在右侧插入空格（一个英文字符占用一个字节）
    - VARCHAR必须指定长度，会多占一个存储空间存尾部空格
    - CHAR效率高，浪费空间，VARCHAR节约空间，但不预先分配存储空间，效率低，长度最好小于5000，大于用TEXT
    - 若字符串长度几乎相等，用CHAR
  - TEXT存较大的文本段，速度慢，不用设默认值

- 定义时可以使用的属性

NULL	数据列可包含NULL值
NOT NULL	数据列不允许包含NULL值
DEFAULT	默认值
PRIMARY KEY	主键
AUTO_INCREMENT	自动递增，适用于整数类型
UNSIGNED	无符号
CHARACTER SET name	指定一个字符集

- DDL：数据定义
  - DDL 的主要功能是定义数据库对象（如：数据库、数据表、视图、索引等）。
  - 数据库（DATABASE）
    - 创建数据库
      - `CREATE DATABASE test;`
    - 删除数据库
      - `DROP DATABASE test;`
    - 选择数据库
      - `USE test;`
  - 数据表（TABLE）
    - 创建数据表
      - **普通创建**
        - `CREATE TABLE user (`
        - `id int(10) unsigned NOT NULL COMMENT 'Id',`
        - `username varchar(64) NOT NULL DEFAULT 'default' COMMENT '用户名',`
        - `password varchar(64) NOT NULL DEFAULT 'default' COMMENT '密码',`
        - `email varchar(64) NOT NULL DEFAULT 'default' COMMENT '邮箱'`
        - `) COMMENT='用户表';`

- **根据已有的表创建新表**
  - `CREATE TABLE vip_user AS`
  - `SELECT * FROM user;`
- 删除数据表
  - `DROP TABLE user;`
- 修改数据表
  - **添加列**
    - `ALTER TABLE user`
    - `ADD age int(3);`
  - **删除列**
    - `ALTER TABLE user`
    - `DROP COLUMN age;`
  - **修改列**
    - `ALTER TABLE `user``
    - `MODIFY COLUMN age tinyint;`
  - **添加主键**
    - `ALTER TABLE user`
    - `ADD PRIMARY KEY (id);`
  - **删除主键**
    - `ALTER TABLE user`
    - `DROP PRIMARY KEY;`
- 索引（INDEX）
  - 作用
    - 通过索引可以更加快速高效地查询数据。
    - 用户无法看到索引，它们只能被用来加速查询。
  - 注意

        - 更新一个包含索引的表需要比更新一个没有索引的表花费更多的时间，这是由于索引本身也需要更新。因此，理想的做法是仅仅在常常被搜索的列（以及表）上面创建索引。
        - 唯一索引表明此索引的每一个索引值只对应唯一的数据记录。
    - 创建索引`CREATE INDEX user_indexON user (id);`
    - 创建唯一索引`CREATE UNIQUE INDEX user_indexON user (id);`
    - 删除索引`ALTER TABLE userDROP INDEX user_index;`
- 约束
    - 概述
        - 约束可以在创建表时规定（通过 CREATE TABLE 语句），或者在表创建之后规定（通过 ALTER TABLE 语句）。
        - 约束类型
            - NOT NULL - 指示某列不能存储 NULL 值。
            - UNIQUE - 保证某列的每行必须有唯一的值。
            - PRIMARY KEY - NOT NULL 和 UNIQUE 的结合。确保某列（或两个列多个列的结合）有唯一标识，有助于更容易更快速地找到表中的一个特定的记录。
            - FOREIGN KEY - 保证一个表中的数据匹配另一个表中的值的参照完整性。
            - CHECK - 保证列中的值符合指定的条件。
            - DEFAULT - 规定没有给列赋值时的默认值。
    - 创建表时使用约束条件：
        - `CREATE TABLE Users (`

- `Id INT(10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '自增Id',`
- `Username VARCHAR(64) NOT NULL UNIQUE DEFAULT 'default' COMMENT '用户名',`
- `Password VARCHAR(64) NOT NULL DEFAULT 'default' COMMENT '密码',`
- `Email VARCHAR(64) NOT NULL DEFAULT 'default' COMMENT '邮箱地址',`
- `Enabled TINYINT(4) DEFAULT NULL COMMENT '是否有效',`
- `PRIMARY KEY (Id)`
- `) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COMMENT='用户表';`

- ————————————————其他结构————————————————
- 视图（VIEW）
  - 虚拟的表，不存数据，更改查询视图其实是对原表进行操作（在原表上开小窗，可用做权限管理），原表称为基表，若表中数据变了，视图中的数据也会变
  - 视图相当于存储起来的SELECT语句，删去视图不会影响原表，经常查询的语句以视图方式存储，可以简化查询
  - 每次使用都需要检索，不适合小型项目，可能会影响性能，适合大型项目
  - 作用是方便查询，最好不要对其进行增删改操作
  - **创建**：CREATE VIEW 视图名 AS... SELECT查询（查询语句中可以FROM视图）
  - **查看**：数据库中有多少视图
    - SHOW TABLES；//查看当前数据库有哪些表和视图
    - DESCRIBE 视图名；//查看视图结构

    - SHOW TABLE STATUS LIKE ‘视图名’；//查看视图属性信息
    - SHOW CREATE VIEW 视图名；//查看视图详细定义信息
  - **更新：**原表中没有对应字段无法更新
    - UPDATE VIEW sno
    - SET salary =20000
    - WHERE employee_id=101;
  - **修改：**
    - CREATE或REPLACE VIEW emp1 AS SELECT....
    - ALTER VIEW...AS SELECT...
  - **删除：**基于a,b视图创建c视图，如果a,b删除，需要手动修改，否则影响使用
    - DROP VIEW empl
    - DROP VIEW IF EXISTS empl 如果存在就删掉，这样写不会报错
- 存储过程与函数（PROCEDURE/FUNCTION）
  - 下面四种结构是在存储过程和函数中使用
  - **存储函数：**用户自定义的函数
  - **存储过程：**封装了预先编译的一系列SQL语句，存储在MYSQL服务器上，客户端请求，服务器便全部执行（省去编译的时间，写查询语句的时间），可移植性差
  - 区别：存储函数一定有返回值，存储过程没有返回值
  - **创建存储过程**
    - CREATE PROCEDURE 存储过程名(IN|OUT|INOUT+参数列表
    - IN是传进来的参数，为默认，OUT把查询结果写入，INOUT都有
  - **创建存储函数**：只有入参，调用时直接写在SELECT后面即可

- CREATE FUNCTION 函数名（参数列表）
- RETURNS +返回值类型
- BEGIN
- 函数体（中间有RETURN语句）
- END

```
CREATE FUNCTION email_by_name()
RETURNS VARCHAR(25)
        DETERMINISTIC
        CONTAINS SQL
        READS SQL DATA
BEGIN
        RETURN (SELECT email FROM employees WHERE last_name = 'Abel');
END //
```

- **查看**
  - 查看创建信息：SHOW CREATE FUNCTION/PROCEDURE sno;
  - 查看状态信息：SHOW PROCEDURE/FUNCTION STATUS;（可以在后面加上LIKE '%salary%'模糊查询）
  - 存储过程函数存放在information_schema数据库下的Routines表中：
    - SELECT * FROM information_schema.Routines
    - WHERE ROUTINE_NAME='sno' AND ROUTINE_TYPE='FUNCTION/PROCEDURE'
- **修改**：ALTER PROCEDURE/FUNCTION 名称....

  不是修改函数体，只是修改特性characteristic，想修改函数体，删去重写
- **删除：**DROP PROCEDURE/FUNCTION IF EXISTS 名称

- 变量
  - 系统变量
    - 变量分为系统变量，用户自定义的变量
    - 系统变量按照作用域分为全局系统变量GLOBAL、会话系统变量SESSION、LOCAL，二者有交集
    - 客户机向MYSQL服务器发起连接，建立会话，MYSQL服务器内部生成与该会话对应的会话系统变量，交集部分

初始值是系统变量的复制，一个会话生成一次变量

- 查看系统变量
  - SHOW GLOBAL/SESSION VARIABLES（SHOW VARIABLES也是SESSION）
  - SHOW GLOBAL/SESSION VARIABLES LIKE '%标识符%'
  - 查看指定的某个系统变量
    - SELECT @@global.变量名
    - SELECT @@session.变量名
    - SELECT @@变量名（先查会话系统变量，不存在则查全局）
- 修改系统变量
  - 这里是暂时性的修改，结束会话、数据库实例就会失效
  - 修改配置文件，可实现永久性的改变，但需要重启
  - SET GLOBAL/SESSION 变量名=变量值
  - SET @@global.变量名/@@session.变量名=变量值
- 用户变量
  - 分为会话用户变量、局部变量
  - 会话用户变量使用@ 开头，对当前会话有效，不用指定类型；局部变量只在BEGIN END语块中有效，放在第一句，只使用在存储函数中
  - **查看**：SELECT @ 用户变量/局部变量名;
  - **用户、局部变量赋值**：
    - 用户变量这一步操作完成定义赋值，局部变量要先定义
    - SET @ 用户变量/局部变量 =/ := 值
    - SELECT 表达式 INTO @ 用户变量/局部变量[FROM 等子句];

        - SELECT AVG(salary) INTO @ avgsalary FROM employees;
    - **局部变量定义：**
        - DECLARE 变量名 INT [DEFAULT 默认值]；//没有默认值初始为NULL
- 定义条件与处理程序（错误处理）
    - 定义条件：为错误起名字
        - DECLARE 错误名 CONDITION FOR 错误码
        - 错误码有两种：MYSQL_error_code数值类型，sqlstate_value字符串类型

```
#举例1：定义"Field_Not_Be_NULL"错误名与MySQL中违反非空约束的错误类型
#是"ERROR 1048 (23000)"对应。
#方式1：使用MySQL_error_code
DECLARE Field_Not_Be_NULL CONDITION FOR 1048;

#方式2：使用sqlstate_value
DECLARE Field_Not_Be_NULL CONDITION FOR SQLSTATE '23000';
```

    - 处理程序：怎么处理错误
        - DECLARE 处理方式 HANDLER FOR 错误类型 处理方式
        - **处理方式：**CONTINUE不处理继续做；EXIT遇到错误马上退出；UNDO撤销，MYSQL不支持
        - **错误类型：**两种错误码、错误名
        - **处理方式：**SET 变量=值 or BEGIN...END
    - MYSQL默认程序执行出错，就不会向下执行
- 流程控制
    - 流程分为顺序、分支、循环
    - 分支IF
        - IF...THEN...END IF
        - IF...THEN...ELSE...END IF
        - IF...THEN...ELSEIF...THEN...ELSE...END IF
    - 分支CASE
        - 格式1：类似SWITCH

```
CASE val
     WHEN 1 THEN SELECT 'val is 1';
     WHEN 2 THEN SELECT 'val is 2';
     ELSE SELECT 'val is not 1 or 2';
END CASE;
```

- 格式2：类似IF

```
CASE
    WHEN val IS NULL THEN SELECT 'val is null';
    WHEN val < 0 THEN SELECT 'val is less than 0';
    WHEN val > 0 THEN SELECT 'val is greater than 0';
    ELSE SELECT 'val is 0';
END CASE;
```

- 循环LOOP
  - [label:] LOOP 循环语句 END LOOP [label]
  - 循环语句中用LEAVE跳出循环，相当于BREAK

```
DECLARE id INT DEFAULT 0;
add_loop:LOOP
    SET id = id +1;
    IF id >= 10 THEN LEAVE add_loop;
    END IF;

END LOOP add_loop;
```

- 循环WHILE：判断后执行
  - [label:] WHILE 条件 DO 循环体
  - END WHILE [label]
- 循环REPEAT：先执行再判断
  - REPEAT 语句
  - UNTIL 判断表达式 END REPEAT
- 跳转LEAVE、ITERATE：循环内使用，LEAVE=BREAK，ITERATE=CONTINUE

- 游标
  - 精确定位某一行的数据，类似指针
  - 占用系统资源，必须关闭，否则影响效率
  - **使用步骤**
    - 1、声明：DECLARE 游标名 CURSOR FOR 查询语句

```
DECLARE cur_emp CURSOR FOR
SELECT employee_id,salary FROM employees;
```

- 2、打开：OPEN 游标名
- 3、从游标里获得数据：FETCH 游标名 INTO 变量名列表
- 4、关闭：CLOSE 游标名

- ————————————————————————————————————
- 触发器
  - 由增删改INSERT/UPDATE/DELETE事件触发执行
  - 优点：保证了数据完整性，一个表更新，相关表也会更新，自动进行合法检查
  - 缺点：比较隐蔽
  - **创建**
    - CREATE TRIGGER 名称
    - BEFORE/AFTER INSERT/UPDATE/DELETE ON 表名
    - FOR EACH ROW
    - 做什么
  - **查看**
    - 查看所有：SHOW TRIGGERS/G
    - 查看某个：SHOW CREATE TRIGGER 名称
    - 从information_schema的触发器表中查看：SELECT*FROM information_schema.TRIGGERS
  - **删除**：DROP TRIGGER IF EXISTS 名称
- 事务
  - 概述
    - 不能回退 SELECT 语句，回退 SELECT 语句也没意义；也不能回退 CREATE 和 DROP 语句。
    - **MySQL 默认是隐式提交**，每执行一条语句就把这条语句当成一个事务然后进行提交。当出现 START TRANSACTION 语句时，会关闭隐式提交；当 COMMIT 或 ROLLBACK 语句执行后，事务会自动关闭，重新恢复隐式提交。

- 通过 set autocommit=0 可以取消自动提交，直到 set autocommit=1 才会提交；autocommit 标记是针对每个连接而不是针对服务器的。
- 指令
  - START TRANSACTION - 指令用于标记事务的起始点。
  - SAVEPOINT - 指令用于创建保留点。
  - ROLLBACK TO - 指令用于回滚到指定的保留点；如果没有设置保留点，则回退到 START TRANSACTION 语句处。
  - COMMIT - 提交事务。

```
-- 开始事务
START TRANSACTION;
-- 插入操作
AINSERT INTO `user`
VALUES (1, 'root1', 'root1', 'xxxx@163.com');
-- 创建保留点 updateA
SAVEPOINT updateA;
-- 插入操作 B
INSERT INTO `user`VALUES (2, 'root2', 'root2', 'xxxx@163.com');
-- 回滚到保留点 updateA
ROLLBACK TO updateA;
-- 提交事务，只有操作 A 生效
COMMIT;
```

(以下为 DCL 语句用法)

九、权限控制

概述

- GRANT 和 REVOKE 可在几个层次上控制访问权限：
  - 整个服务器，使用 GRANT ALL 和 REVOKE ALL；
  - 整个数据库，使用 ON database.*；
  - 特定的表，使用 ON database.table；
  - 特定的列；
  - 特定的存储过程。
- 新创建的账户没有任何权限。
- 账户用 username@host 的形式定义，username@% 使用的是默认主机名。
- MySQL 的账户信息保存在 mysql 这个数据库中。
  - `USE mysql;SELECT user FROM user;复制代码`

- 创建账户
  - `CREATE USER myuser IDENTIFIED BY 'mypassword';`
- 修改账户名
  - `UPDATE user SET user='newuser' WHERE user='myuser';`
  - `FLUSH PRIVILEGES;`
- 删除账户
  - `DROP USER myuser;`
- 查看权限
  - `SHOW GRANTS FOR myuser;`
- 授予权限
  - `GRANT SELECT, INSERT ON *.* TO myuser;`
- 删除权限
  - `REVOKE SELECT, INSERT ON *.* FROM myuser;`
- 更改密码
  - `SET PASSWORD FOR myuser = 'mypass';`

以上内容整理于 幕布文档