# MySQL启动与停止（默认启动）

Win+R 输入‘services.msc’找到MySQL80 可启动与停止

开始界面搜索 cmd 以管理员身份启动‘net start mysql80’开始；‘net stop mysql80’停止

MySQL的客户端连接

直接启动‘MySQL 8.0 Command Line Client - Unicode’

开始界面搜索 cmd 以管理员身份启动，输入‘mysql -u root -p’和密码启动

（-h地址\-P端口\-u用户名\-p密码）

SQL通用语法

1.SQL语句可以单行或多行书写，以分号结尾。

2. SQL语句可以使用空格/缩进来增强语句的可读性。

3. MySQL数据库的SQL语句不区分大小写，关键字建议使用大写。

4.注释:

。单行注释:--注释内容或#注释内容(MySQL特有)

。多行注释:/\*注释内容\*/

# SQL分类

|  |  |  |
| --- | --- | --- |
| 分类 | 全称 | 说明 |
| DDL | Data Definition Language | 数据定义语言，用来定义数据库对象(数据库，表，字段) |
| DML | Data Manipulation Language | 数据操作语言，用来对数据库表中的数据进行增删改 |
| DQL | Data Query Language | 数据查询语言，用来查询数据库中表的记录 |
| DCL | Data Control Language | 数据控制语言，用来创建数据库用户、控制数据库的访问权限 |

# DDL

## DDL-数据库操作

查询所有数据库

SHOW DATABASES ;

查询当前数据库

SELECT DATABASE();

创建

CREATE DATABASE [ IF NOT EXISTS] 数据库名[DEFAULT CHARSET字符集] [ COLLATE 排序规则];

删除

DROP DATABASE [ IF EXISTS]数据库名;

使用

USE数据库名;

## DDL-表操作-查询-创建-数据类型-修改-删除

查询当前数据库所有表

SHOW TABLES;

查询表结构入

DESC表名;

查询指定表的建表语句

SHOW CREATE TABLE表名;

DDL-表操作-创建

CREATE TABLE表名(

字段1字段1类型[ COMMENT字段1注释]，

字段2字段2类型[COMMENT 字段2注释],

字段3字段3类型[COMMENT字段3注释]，

字段n字段n类型[COMMENT字段n注释]) [ COMMENT 表注释];

注意:[ .….]为可选参数,最后一个字段后面没有逗号

创建一个结构像老表的新表

CREATE TABLE 新表 LIKE 老表；

将老表的数据复制到新表

INSERT INTO 新表名 SELECT \*FROM 老表名[where ...]；

添加字段

ALTER TABLE表名ADD字段名类型(长度)[COMMENT注释][约束];

修改数据类型

ALTER TABLE表名MODIFY字段名新数据类型(长度);

修改字段名和字段类型

ALTER TABLE表名CHANGE旧字段名新字段名类型(长度)[COMMENT注释][约束];

删除字段

ALTER TABLE表名DROP字段名;

修改表名

ALTER TABLE表名RENAME TO 新表名;

删除表

DROP TABLE[ IF EXISTS]表名;

删除指定表，并重新创建该表

TRUNCATE TABLE表名;

# DML

## DML-添加数据

1.给指定字段添加数据

INSERT INTO表名(字段名1，字段名2,...) VALUES(值1,值2,...);

2.给全部字段添加数据

INSERT INTO表名VALUES(值1,值2,...);

3.批量添加数据

INSERT INTO表名(字段名1,字段名2,.…) VALUES (值1,值2,...),(值1,值2,...)，(值1，值2.…) ;

INSERT INTO表名VALUES(值1,值2,...),(值1,值2,...),(值1,值2,...);

注意:

·插入数据时，指定的字段顺序需要与值的顺序是一一对应的。

·字符串和日期型数据应该包含在引号中。

·插入的数据大小，应该在字段的规定范围内。

## DML-修改数据

UPDATE 表名SET字段名1=值1，字段名2=值2 ,....[ WHERE条件];

注意:

修改语句的条件可以有，也可以没有，如果没有条件，则会修改整张表的所有数据。

## DML-删除数据

DELETE FROM表名[WHERE条件]

注意:

DELETE语句的条件可以有，也可以没有，如果没有条件，则会删除整张表的所有数据。DELETE语句不能删除某一个字段的值(可以使用UPDATE)。

# DQL

## DQL-语法

SELECT

字段列表

FROM

表名列表心

WHERE

条件列表

\*聚合函数(count、max、min、avg.sum)

GROUP BY

分组字段列表

HAVING

分组后条件列表

ORDER BY

排序字段列表

LIMIT

分页参数

## DQL-基本查询

1.查询多个字段

SELECT字段1,字段2,字段3... FROM 表名;

SELECT \* FROM表名;

2.设置别名

SELECT字段1[AS 别名1],字段2[AS 别名2]... FROM表名;

3．去除重复记录

SELECT DISTINCT字段列表FROM表名;

## DQL-条件查询

1.语法

SELECT 字段列表 FROM 表名WHERE条件列表;

2.条件

|  |  |  |  |
| --- | --- | --- | --- |
| 比较运算符 | 功能 | 逻辑运算符 | 功能 |
| > | 大于 | AND或&& | 并且(多个条件同时成立) |
| >= | 大于等于 | OR或|| | 或者(多个条件任意一个成立) |
| < | 小于 | NOT或! | 非，不是 |
| <= | 小于等于 |  |  |
| = | 等于 |  |  |
| <>或!= | 不等于 |  |  |
| BETWEEN ...AND ... | 在某个范围之内(含最小、最大值) |  |  |
| IN(...) | 在in之后的列表中的值，多选一 |  |  |
| LIKE占位符 | 模糊匹配(\_匹配单个字符,%匹配任意个字符) |  |  |
| IS NULL | 是NULL |  |  |

## DQL-聚合函数

1.介绍

将一列数据作为一个整体，进行纵向计算。

2.常见聚合函数

|  |  |
| --- | --- |
| 函数 | 功能 |
| count | 统计数量 |
| max | 最大值 |
| min | 最小值 |
| avg | 平均值 |
| sum | 求和 |

3.语法

SELECT 聚合函数(字段列表)FROM表名;

注意:

null值不参与所有聚合函数运算.

## DQL-分组查询

1.语法

SELECT 字段列表FROM表名「WHERE 条件]GROUP BY分组字段名〔 HAVING分组后过滤条件];

2. where与having区别

执行时机不同: where是分组之前进行过滤，不满足where条件，不参与分组;而having是分组之后对结果进行过滤。

判断条件不同: where不能对聚合函数进行判断，而having可以。

注意

·执行顺序: where >聚合函数>having 。

·分组之后，查询的字段一般为聚合函数和分组字段，查询其他字段无任何意义。

## DQL-排序查询

1.语法

SELECT 字段列表FROM表名ORDER BY 字段1排序方式1，字段2排序方式2;

2.排序方式

ASC:升序（默认值)

DESC:降序

注意:

如果是多字段排序，当第一个字段值相同时，才会根据第二个字段进行排序。

## DQL-分页查询

1.语法

SELECT字段列表FROM表名LIMIT 起始索引,查询记录数;

注意

·起始索引从0开始，起始索引=(查询页码-1)\*每页显示记录数。

·分页查询是数据库的方言，不同的数据库有不同的实现，MySQL中是LIMIT。

·如果查询的是第一页数据，起始索引可以省略，直接简写为limit 10。

## DQL-编写顺序DQL-执行顺序

SELECT 4

字段列表

FROM 1

表名列表

WHERE 2

条件列表

GROUP BY 3

分组字段列表

HAVING

分组后条件列表

ORDER BY 5

排序字段列表

LIMIT 6

分页参数

# DCL

## DCL-管理用户

1．查询用户

USE mysql;

SELECT \*FROM user;

2．创建用户

CREATE USER ‘用户名’@’主机名’ IDENTIFIED BY‘密码’;

3．修改用户密码

ALTER USER ‘用户名’@’主机名’IDENTIFIED WITH mysql\_native\_password BY‘新密码’;

4．删除用户

DROP USER’用户名’@’主机名’;

--创建用户 heima ，可以在任意主机访问该数据库，密码123456 ;

create user ‘heima‘@‘%’identified by '123456';

注意:

·主机名可以使用%通配（任意主机都可访问）。

·这类SQL开发人员操作的比较少，主要是DBA ( Database Administrator 数据库管理员）使用。

## DCL-权限控制

MySQL中定义了很多种权限，但是常用的就以下几种:

|  |  |
| --- | --- |
| 权限 | 说明 |
| ALL,ALL PRIVILEGES | 所有权限 |
| SELECT | 查询数据 |
| INSERT | 插入数据 |
| UPDATE | 修改数据 |
| DELETE | 删除数据 |
| ALTER | 修改表 |
| DROP | 删除数据库/表/视图 |
| CREATE | 创建数据库/表 |

## DCL-权限控制

1．查询权限

SHOW GRANTS FOR‘用户名'@'主机名';

2．授予权限

GRANT权限列表ON数据库名.表名TO‘用户名’@’主机名’;

3.撤销权限

REVOKE 权限列表 ON数据库名.表名FROM ‘用户名’@’主机名’;

注意:

·多个权限之间，使用逗号分隔

·授权时，数据库名和表名可以使用\*进行通配，代表所有。

# 函数

语法：select 函数(参数);

## 字符串函数

|  |  |
| --- | --- |
| 函数 | 功能 |
| CONCAT(S1,S2.…Sn) | 字符串拼接，将S1，S2，...Sn拼接成一个字符串 |
| LOWER(str) | 将字符串str全部转为小写 |
| UPPER(str) | 将字符串str全部转为大写 |
| LPAD(str,n,pad) | 左填充，用字符串pad对str的左边进行填充，达到n个字符串长度 |
| RPAD(str,n,pad) | 右填充，用字符串pad对str的右边进行填充，达到n个字符串长度 |
| TRIM(str) | 去掉字符串头部和尾部的空格 |
| SUBSTRING(str,start,len) | 返回从字符串str从start位置起的len个长度的字符串 |

## 数值函数

|  |  |
| --- | --- |
| 函数 | 功能 |
| CEIL(x) | 向上取整 |
| FLOOR(x) | 向下取整 |
| MOD(x,y) | 返回x/y的模(取余数) |
| RAND() | 返回0~1内的随机数 |
| ROUND(x,y) | 求参数x的四舍五入的值，保留y位小数 |

## 日期函数

|  |  |
| --- | --- |
| 函数 | 功能 |
| CURDATE() | 返回当前日期 |
| CURTIME() | 返回当前时间 |
| NOW() | 返回当前日期和时间 |
| YEAR(date) | 获取指定date的年份 |
| MONTH(date) | 获取指定date的月份 |
| DAY(date) | 获取指定date的日期 |
| DATE\_ADD(date, INTERVAL expr type) | 返回一个日期/时间值加上一个时间间隔expr后的时间值 |
| DATEDIFF(date1,date2) | 返回起始时间date1和结束时间date2之间的天数(date1-date2) |

## 流程函数

|  |  |
| --- | --- |
| 函数 | 功能 |
| IF(value , t , f)  IFNULL(value1 , value2) | 如果value为true，则返回t，否则返回f  如果value1不为空，返回value1，否则返回value2 |
| CASE WHEN [ val1 ] THEN [res1] ...ELSE [ default ] END | 如果val1为true，返回res1，...否则返回default默认值 |
| CASE [ expr ] WHEN [ val1 ] THEN [res1] ...ELSE [ default ] END | 如果expr的值等于vall，返回res1，...否则返回default默认值 |

# 约束

概述

1．概念:约束是作用于表中字段上的规则，用于限制存储在表中的数据。

2.目的:保证数据库中数据的正确、有效性和完整性。

3．分类:

|  |  |  |
| --- | --- | --- |
| 约束 | 描述 | 关键字 |
| 非空约束 | 限制该字段的数据不能为null | NOT NULL |
| 唯一约束 | 保证该字段的所有数据都是唯一、不重复的 | UNIQUE |
| 主键约束 | 主键是一行数据的唯一标识，要求非空且唯一 | PRIMARY KEY |
| 默认约束 | 保存数据时，如果未指定该字段的值，则采用默认值 | DEFAULT |
| 检查约束(8.0.16版本之后) | 保证字段值满足某一个条件 | CHECK |
| 外键约束 | 用来让两张表的数据之间建立连接，保证数据的一致性和完整性 | FOREIGN KEY |
| 自增约束 | 用于主键，由一开始自动增长，只有MySQL中有 | AUTO\_INCREMENT |

注意:约束是作用于表中字段上的，可以在创建表/修改表的时候添加约束。

## 外键约束

外键用来让两张表的数据之间建立连接，从而保证数据的一致性和完整性。

有外键的表叫子表（从表），另一个叫父表（主表）

添加外键

CREATE TABLE表名(

字段名数据类型,

…..

[CONSTRAINT][外键名称]FOREIGN KEY (外键字段名) REFERENCES主表(主表列名)

);

ALTER TABLE 表名 ADD CONSTRAINT 外键名称FOREIGN KEY(外键字段名)REFERENCES主表(主表列名);

删除外键

ALTER TABLE 表名DROP FOREIGN KEY外键名称;

# 删除/更新行为

|  |  |
| --- | --- |
| 行为 | 说明 |
| NO ACTION | 当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不允许删除/更新。(与RESTRICT一致) |
| RESTRICT | 当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不允许删除/更新。(与NO ACTION一致) |
| CASCADE | 当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有，则也删除/更新外键在子表中的记录。 |
| SET NULL | 当在父表中删除对应记录时，首先检查该记录是否有对应外键，如果有则设置子表中该外键值为null(这就要求该外键允许取null)。 |
| SET DEFAULT | 父表有变更时，子表将外键列设置成一个默认的值(Innodb不支持) |

CASCADE/SET NULL用法：

ALTER TABLE 表名 ADD CONSTRAINT外键名称FOREIGNKEY(外键字段)REFEENCES主表名(主表字段名)ON UPDATE （CASCADE/SET NULL） ON DEIETE （CASCADE/SET NULL）;

# 连接查询

## 连接查询-内连接

内连接查询语法:

隐式内连接

SELECT字段列表FROM表1,表2WHERE条件...;

显式内连接

SELECT字段列表FROM 表1[ INNER ] JOIN表2 ON连接条件...;

内连接查询的是两张表交集的部分

## 连接查询-外连接

外连接查询语法:

左外连接

SELECT字段列表FROM 表1 LEFT [ OUTER ] JOIN 表2 ON条件...;

相当于查询表1(左表)的所有数据包含表1和表2交集部分的数据

右外连接

SELECT 字段列表FROM 表1 RIGHT [ OUTER ] JOIN表2 ON条件...;

相当于查询表2(右表)的所有数据包含表1和表2交集部分的数据

## 连接查询-自连接

自连接查询语法:

SELECT字段列表FROM 表A别名A JOIN 表A 别名B ON 条件....;

自连接查询,可以是内连接查询，也可以是外连接查询。

必须取别名！

# 联合查询-union , union all

对于union查询，就是把多次查询的结果合并起来，形成一个新的查询结果集。

SELECT字段列表 FROM 表A ....

UNION[ ALL ]

SELECT字段列表 FROM 表B ....;

对于联合查询的多张表的列数必须保持一致，字段类型也需要保持一致。

union all会将全部的数据直接合并在一起，union会对合并之后的数据去重。

# 子查询

概念:SQL语句中嵌套SELECT语句，称为嵌套查询，又称子查询。

SELECT \*FROM t1 WHERE column1 = ( SELECT column1 FROM t2);

子查询外部的语句可以是INSERT / UPDATE /DELETE / SELECT的任何一个。

根据子查询结果不同，分为:

### 标量子查询（子查询结果为单个值)

子查询返回的结果是单个值（数字、字符串、日期等)，最简单的形式，这种子查询成为标量子查询。

常用的操作符:= <> > >= < <=

### 列子查询(子查询结果为一列)

子查询返回的结果是一列（可以是多行），这种子查询称为列子查询。

常用的操作符:IN 、NOT IN 、ANY 、SOME 、ALL

|  |  |
| --- | --- |
| 操作符 | 描述 |
| IN | 在指定的集合范围之内，多选一 |
| NOT IN | 不在指定的集合范围之内 |
| ANY | 子查询返回列表中，有任意一个满足即可 |
| SOME | 与ANY等同，使用SOME的地方都可以使用ANY |
| ALL | 子查询返回列表的所有值都必须满足 |

### 行子查询(子查询结果为一行)

子查询返回的结果是一行（可以是多列)，这种子查询称为行子查询。

常用的操作符:= 、<>、IN 、NOT IN

### 表子查询(子查询结果为多行多列)

子查询返回的结果是多行多列，这种子查询称为表子查询。

常用的操作符:IN

根据子查询位置，分为:WHERE之后、FROM之后、SELECT之后。

# 多表关系

一对多:在多的一方设置外键,关联一的一方的主键

多对多:建立中间表，中间表包含两个外键,关联两张表的主键

一对一:用于表结构拆分，在其中任何一方设置外键(UNIQUE)，关联另一方的主键

# 事务

事务是一组操作的集合，它是一个不可分割的工作单位，事务会把所有的操作作为一个整体一起向系统提交或撤销操作请求，即这些操作要么同时成功，要么同时失败。

默认MySQL的事务是自动提交的，也就是说，当执行一条DML语句，MySQL会立即隐式的提交事务。

## 事务操作

·查看/设置事务提交方式

SELECT @@autocommit;

SET @@autocommit=0;

//0为手动，1为自动

·提交事务

COMMIT ;

·回滚事务

ROLLBACK;

·开启事务

START TRANSACTION或 BEGIN;

## 事务四大特性（ACID）

·原子性（Atomicity)：事务是不可分割的最小操作单元，要么全部成功，要么全部失败。

·一致性（Consistency)：事务完成时，必须使所有的数据都保持一致状态。

·隔离性（Isolation)：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立环境下运行。

·持久性〈Durability)：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。

## 并发事务问题

|  |  |
| --- | --- |
| 问题 | 描述 |
| 脏读 | 一个事务读到另外一个事务还没有提交的数据。 |
| 不可重复读 | 一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读 |
| 幻读 | 一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据已经存在，好像出现了“幻影”。 |

## 事务隔离级别

|  |  |  |  |
| --- | --- | --- | --- |
| 隔离级别 | 赃读 | 不可重复读 | 幻读 |
| Read uncommitted（读未提交） | √ | √ | √ |
| Read committed（读已提交） | × | √ | √ |
| Repeatable Read（可重复读，MySQL中默认） | × | × | √ |
| Serializable（串行化，只允许一个事务单独运行） | × | × | × |

--查看事务隔离级别

SELECT @@TRANSACTION\_ISOLATION;

--设置事务隔离级别

SET [SESSION|GLOBAL] TRANSACTION SOLATION LEVEL { READ UNCOMMTTED | READ COMMITED | REPEATABLE READ | SERALIZABLE }

注意:事务隔离级别越高，数据越安全，但是性能越低。

# 视图

视图（View）是一种虚拟存在的表。视图中的数据并不在数据库中实际存在，行和列数据来自定义视图的查询中使用的表，并且是在使用视图时动态生成的。

通俗的讲，视图只保存了查询的SQL逻辑，不保存查询结果。所以我们在创建视图的时候，主要的工作就落在创建这条SQL查询语句上。

## 创建

CREATE [OR REPLACE] VIEW 视图名称[(列名列表)] AS SELECT 语句[WITH[ CASCADED | LOCAL] CHECK OPTION]

## 查询

查看创建视图语句:SHOW CREATE VIEW视图名称

查看视图数据:SELECT\* FROM 视图名称......;

## 修改

方式一:CREATE [OR REPLACE] VIEW视图名称[(列名列表)]AS SELECT语句[WITH[ CASCADED | LOCAL] GHECK OPTION]

方式二:ALTER VIEW视图名称[(列名列表)]AS SELECT语句[WITH [ CASCADED | LOCAL]CHECK OPTION ]

CASCADED级联

## 删除

DROP VIEW [IF EXISTS] 视图名称 [,视图名称]...

## 视图的检查选项

当使用WITH CHECK OPTION子句创建视图时，MySQL会通过视图检查正在更改的每个行,例如插入，更新，删除，以使其符合视图的定义。MySQL允许基于另一个视图创建视图，它还会检查依赖视图中的规则以保持一致性。为了确定检查的范围，mysql提供了两个选项;CASCADED和LOCAL，默认值为CASCADED。

CASCADED 不管基图是否检查，只要新图检查就会检查基图

LOCAL 新图检查会看基图是否检查，检查则一并检查，否则只检查新图不检查基图

插入的数据实际上是插入了基表，而不是视图，因为视图是虚拟表

## 视图的更新

要使视图可更新，视图中的行与基础表中的行之间必须存在一对一的关系。如果视图包含以下任何一项，则该视图不可更新:

1.聚合函数或窗口函数(SUM()、MIN()、MAX()、COUNT()等)

2.DISTINCT

3.GROUP BY4.HAVING

5. UNION或者UNION ALL

## 作用

简单

视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而使得用户不必为以后的操作每次指定全部的条件。

安全

数据库可以授权，但不能授权到数据库特定行和特定的列上。通过视图用户只能查询和修改他们所能见到的数据。

数据独立

视图可帮助用户屏蔽真实表结构变化带来的影响。

# 存储过程

修改结束符：DELIMITER 新的换行符

## 介绍

存储过程是事先经过编译并存储在数据库中的一段SQL语句的集合，调用存储过程可以简化应用开发人员的很多工作，减少数据在数据库和应用服务器之间的传输，对于提高数据处理的效率是有好处的。

存储过程思想上很简单，就是数据库SQL语言层面的代码封装与重用。

## 特点

封装，复用

可以接收参数，也可以返回数据

减少网络交互，效率提升

## 创建

CREATE PROCEDURE存储过程名称([参数列表])

BEGIN

--SQL语句

END;

## 调用

CALL名称([参数]);

## 查看

查询指定数据库的存储过程及状态信息

SELECT\*FROM INFORMATION\_SCHEMA.ROUTINES WHERE ROUTINE \_SCHEMA ='xxx';

查询某个存储过程的定义

SHOW CREATE PROCEDURE存储过程名称;

## 删除

DROP PROCEDURE [IF EXISTS]存储过程名称;

注意:在命令行中，执行创建存储过程的SQL时，需要通过关键字delimiter指定SQL语句的结束符。（一般改为$$）

# 变量

## 系统变量

系统变量是MySQL服务器提供，不是用户定义的，属于服务器层面。分为全局变量(GLOBAL)、会话变量(SESSION)。（默认会话级别）

### 查看系统变量

查看所有系统变量

SHOW [SESSION |GLOBAL ] VARIABLES;

可以通过LIKE模糊匹配方式查找变量

SHOW [ SESSION | GLOBAL ] VARIABLES LIKE..... ;

查看指定变量的值

SELECT @@[SESSION .|GLOBAL.]系统变量名;

### 设置系统变量

SET [SESSION | GLOBAL]系统变量名=值;

SET @@[SESSION| GLOBAL]系统变量名=值;

注意:

如果没有指定SESSION/GLOBAL，默认是SESSION，会话变量。

mysql服务重新启动之后，所设置的全局参数会失效，要想不失效，可以在/etc/my.cnf中配置。

## 用户定义变量

是用户根据需要自己定义的变量，用户变量不用提前声明，在用的时候直接用“@变量名”使用就可以。其作用域为当前连接（会话）。

### 赋值

SET @var\_name = expr [, @var\_name = expr ] ... ;

SET @var\_name := expr [, @var\_name := expr] ... ;

SELECT @var\_name := expr [, @var\_name := expr] ... ;

SELECT字段名INTO @var\_name FROM表名;

### 使用

SELECT @var\_name ;

注意:

用户定义的变量无需对其进行声明或初始化，只不过获取到的值为NULL。

## 局部变量

是根据需要定义的在局部生效的变量，访问之前，需要DECLARE声明。可用作存储过程内的局部变量和输入参数，局部变量的范围是在其内声明的BEGIN....END块。

### 声明

DECLARE变量名变量类型[DEFAULT ...];

变量类型就是数据库字段类型:INT、BIGINT、CHAR、VARCHAR、DATE、TIME等。

### 赋值

SET变量名=值;

SET变量名:=值;

SELECT字段名INTO变量名FROM表名...;

# if语法:

IF条件1 THEN

......

ELSEIF 条件2 THEN

......

ELSE

......

END IF;

## 参数

|  |  |  |
| --- | --- | --- |
| 类型 | 含义 | 备注 |
| IN | 该类参数作为输入,也就是需要调用时传入值 | 默认 |
| OUT | 该类参数作为输出，也就是该参数可以作为返回值 |  |
| INOUT | 既可以作为输入参数,也可以作为输出参数 |  |

## 用法:

CREATE PROCEDURE存储过程名称([ IN/OUT/INOUT 参数名参数类型])

BEGIN

--SQL语句

END;

# case语法:

语法一

CASE case\_value

WHEN when\_value1 THEN statement\_list1

[ WHEN when\_value2 THEN statement\_list 2] ...

[ ELSE statement\_list ]

END CASE;

语法二

CASE

WHEN search\_condition1 THEN statement\_list1

[WHEN search\_condition2 THEN statement\_list2] ...

[ELSE statement\_list]

END CASE;

# while

while循环是有条件的循环控制语句。满足条件后，再执行循环体中的SQL语句。具体语法为:

#先判定条件，如果条件为true，则执行逻辑，否则，不执行逻辑

WHILE条件DO

SQL逻辑..

END WHILE;

# repeat

repeat是有条件的循环控制语句,当满足条件的时候退出循环。具体语法为:

#先执行一次逻辑，然后判定逻辑是否满足，如果满足，则退出。如果不满足，则继续下一次循环

REPEAT

SQL逻辑...

UNTIL条件

END REPEAT;

# loop

LOOP实现简单的循环，如果不在SQL逻辑中增加退出循环的条件，可以用其来实现简单的死循环。LOOP可以配合一下两个语句使用:

LEAVE:配合循环使用，退出循环。

ITERATE:必须用在循环中，作用是跳过当前循环剩下的语句，直接进入下一次循环

[begin\_label:] LOOP

SQL逻辑…

END LOOP [end\_label];

LEAVE label; --退出指定标记的循环体

ITERATE label; --直接进入下一次循环

# 游标(CURSOR)

游标(CURSOR)是用来存储查询结果集的数据类型,在存储过程和函数中可以使用游标对结果集进行循环的处理。游标的使用包括游标的声明、OPEN、FETCH 和CLOSE，其语法分别如下。

## 声明游标

DECLARE 游标名称CURSOR FOR查询语句;

## 打开游标

OPEN游标名称;

## 获取游标记录

FETCH 游标名称INTO变量[,变量];

## 关闭游标

CLOSE 游标名称;

# 条件处理程序(Handler)

条件处理程序(Handler)可以用来定义在流程控制结构执行过程中遇到问题时相应的处理步骤。具体语法为:

DECLARE handler\_action HANDLER FOR condition\_value [, condition\_value] ... statement ;

handler\_action

CONTINUE:继续执行当前程序

EXIT:终止执行当前程序

condition\_value

SQLSTATE sqlstate\_value:状态码，如02000

SQLWARNING:所有以01开头的SQLSTATE代码的简写

NOT FOUND:所有以02开头的SQLSTATE代码的简写

SQLEXCEPTION:所有没有被SQLWARNING或 NOT FOUND捕获的SQLSTATE代码的简写

# 存储函数

存储函数是有返回值的存储过程，存储函数的参数只能是IN类型的（存储函数可以被存储过程替代）。具体语法如下:

CREATE FUNCTON存储函数名称([参数列表])

RETURNS type [characteristic ...]

BEGIN

-- SQL语句

RETURN ...;

END;

characteristic说明:

·DETERMINISTIC:相同的输入参数总是产生相同的结果

·NO SQL:不包含SQL语句。

·READS SQL DATA:包含读取数据的语句，但不包含写入数据的语句。

# 触发器

触发器是与表有关的数据库对象，指在insert/update/delete之前或之后，触发并执行触发器中定义的SQL语句集合。触发器的这种特性可以协助应用在数据库端确保数据的完整性，日志记录，数据校验等操作。

使用别名OLD和NEW来引用触发器中发生变化的记录内容，这与其他的数据库是相似的。现在触发器还只支持行级触发，不支持语句级触发。

|  |  |
| --- | --- |
| 触发器类型 | NEW 和OLD |
| INSERT型触发器 | NEW表示将要或者已经新增的数据 |
| UPDATE型触发器 | OLD表示修改之前的数据，NEW表示将要或已经修改后的数据 |
| DELETE型触发器 | OLD表示将要或者已经删除的数据 |

## 语法

### 创建

CREATE TRIGGER trigger\_name

BEFORE/AFTER INSERT/UPDATE/DELETE

ON tbl\_name FOR EACH ROW --行级触发器

BEGIN

trigger\_stmt ;

END;

### 查看

SHOW TRIGGERS ;

### 删除

DROP TRIGGER [schema\_name.]trigger\_name ;

-- 如果没有指定schema\_name，默认为当前数据库。

# 锁

锁是计算机协调多个进程或线程并发访问某一资源的机制。在数据库中，除传统的计算资源（CPU、RAM、I/O)的争用以外，数据也是一种供许多用户共享的资源。如何保证数据并发访问的一致性、有效性是所有数据库必须解决的一个问题，锁冲突也是影响数据库并发访问性能的一个重要因素。从这个角度来说，锁对数据库而言显得尤其重要，也更加复杂。

在并发访问时，解决数据访问的一致性、有效性问题

MySQL中的锁，按照锁的粒度分，分为以下三类:

1.全局锁:锁定数据库中的所有表。

2.表级锁:每次操作锁住整张表。

3.行级锁:每次操作锁住对应的行数据。

## 全局锁

全局锁就是对整个数据库实例加锁，加锁后整个实例就处于只读状态，后续的DML的写语句，DDL语句，已经更新操作的事务提交语句都将被阻塞。

其典型的使用场景是做全库的逻辑备份，对所有的表进行锁定，从而获取一致性视图，保证数据的完整性。性能较差，数据逻辑备份时使用

上锁

flush tables with read lock ;

备份

mysqldump -uroot -p1234 itcast > itcast.sql

解锁

unlock tables ;

特点

数据库中加全局锁，是一个比较重的操作，存在以下问题:

1．如果在主库上备份，那么在备份期间都不能执行更新，业务基本上就得停摆。

2．如果在从库上备份，那么在备份期间从库不能执行主库同步过来的二进制日志(binlog)，会导致主从延迟。

在InnoDB引擎中，我们可以在备份时加上参数--single-transaction参数来完成不加锁一致性数据备份。

语法：

mysqldump --single-transaction -uroot -p123456 itcast > itcast.sql

## 表级锁

表级锁，每次操作锁住整张表。锁定粒度大，发生锁冲突的概率最高，并发度最低。应用在MyISAM、InnoDB、BDB等存储引擎中。

对于表级锁，主要分为以下三类:

1.表锁

2.元数据锁( meta data lock,MDL)

3.意向锁

·表锁

对于表锁，分为两类:

1.表共享读锁( read lock )

2.表独占写锁(write lock )

语法:

1.加锁:lock tables表名... read/write。

2.释放锁:unlock tables/客户端断开连接。



读锁不会阻塞其他客户端的读，但是会阻塞写。

写锁既会阻塞其他客户端的读，又会阻塞其他客户端的写。

·元数据锁( meta data lock,MDL)

MDL加锁过程是系统自动控制，无需显式使用，在访问一张表的时候会自动加上。MDL锁主要作用是维护表元数据的数据一致性，在表上有活动事务的时候，不可以对元数据进行写入操作。

为了避免DML与DDL冲突，保证读写的正确性。

在MySQL5.5中引入了MDL，当对一张表进行增删改查的时候，加MDL读锁(共享);当对表结构进行变更操作的时候，加MDL写锁(排他)。



查看元数据锁:

select object\_type,object schema,object\_name,lock type,lock\_duration from performance\_schema.metadata\_locks ;

·意向锁

为了避免DML在执行时，加的行锁与表锁的冲突，在InnoDB中引入了意向锁，使得表锁不用检查每行数据是否加锁，使用意向锁来减少表锁的检查。

分类

1.意向共享锁（IS）：由语句select ... lock in share mode添加。

2.意向排他锁（IX）：由insert、update、delete、select ... for update添加。

兼容性

1.意向共享锁（lS）：与表锁共享锁（read）兼容，与表锁排它锁（write)互斥。

2.意向排他锁（IX）：与表锁共享锁（read）及排它锁（write）都互斥。意向锁之间不会互斥。

可以通过以下SQL，查看意向锁及行锁的加锁情况:

select object\_schema,object\_name,index\_name,lock\_type,lock\_mode,lock\_data from performance\_schema.data\_locks;

## 行级锁

行级锁，每次操作锁住对应的行数据。锁定粒度最小，发生锁冲突的概率最低，并发度最高。应用在InnoDB存储引擎中。

InnoDB的数据是基于索引组织的，行锁是通过对索引上的索引项加锁来实现的，而不是对记录加的锁。对于行级锁，主要分为以下三类:

1.行锁(Record Lock):锁定单个行记录的锁，防止其他事务对此行进行update和delete。在RC、RR隔离级别下都支持。

2.间隙锁(Gap Lock)∶锁定索引记录间隙（不含该记录)，确保索引记录间隙不变，防止其他事务在这个间隙进行insert,产生幻读。在RR隔离级别下都支持。

3.临键锁（Next-Key Lock):行锁和间隙锁组合，同时锁住数据，并锁住数据前面的间隙Gap。在RR隔离级别下支持。

·行锁

lnnoDB实现了以下两种类型的行锁:

1.共享锁(S）︰允许一个事务去读一行，阻止其他事务获得相同数据集的排它锁。

2.排他锁（X)︰允许获取排他锁的事务更新数据，阻止其他事务获得相同数据集的共享锁和排他锁。





行锁-演示

默认情况下，InnoDB在REPEATABLE READ事务隔离级别运行，InnoDB使用next-key锁进行搜索和索引扫描，以防止幻读。

1．针对唯一索引进行检索时，对已存在的记录进行等值匹配时，将会自动优化为行锁。

2. InnoDB的行锁是针对于索引加的锁，不通过索引条件检索数据，那么InnoDB将对表中的所有记录加锁，此时就会升级为表锁。

可以通过以下SQL，查看意向锁及行锁的加锁情况:

select object\_schema,object\_name,index\_name,lock\_type,lock\_mode,lock\_data from performance\_schema.data\_locks;

间隙锁/临键锁-演示

默认情况下，InnoDB在REPEATABLE READ事务隔离级别运行，InnoDB使用next-key锁进行搜索和索引扫描，以防止幻读。

1.索引上的等值查询(唯一索引)，给不存在的记录加锁时,优化为间隙锁。

2.索引上的等值查询(普通索引)，向右遍历时最后一个值不满足查询需求时，next-key lock退化为间隙锁。

3.索引上的范围查询(唯一索引)--会访问到不满足条件的第一个值为止。

注意:间隙锁唯一目的是防止其他事务插入间隙。间隙锁可以共存，一个事务采用的间隙锁不会阻止另一个事务在同一间隙上采用间隙锁。