# [步步深入：MySQL架构总览->查询执行流程->SQL解析顺序](https://www.cnblogs.com/annsshadow/p/5037667.html)

<https://www.cnblogs.com/annsshadow/p/5037667.html>

数据库管理系统DBMS=DataBase Management System

SQL：结构化查询语言——》功能包括查询、操作、定义和控制，是一个综合的、通用的关系数据库语言，同时又是一种高级。。。

RDBMS关系型数据库:Oracle、DB2、MySQL、SQL Server

专业术语如下：

表、数据库、列、行、主建、外键（用来关联两个表的）、索引（相当于书的目录）

非关系型数据库NoSql：MongoDB、Redis、SQLlite、Hbase…

常用数据库命令：

MySql数据库分为两种、

系统数据库、

Information\_schema\performance\_schema\mysql\sys.

用户数据库。

创建数据库、

Create database 数据库名；

删除数据库、

Drop database 数据库名；

查看数据库、

使用数据库、

Use 数据库名；

查看数据库当中有多少张表、

Show tables;（查看数据库索引（目录名）信息使用show，如果是具体对象结构信息使用desc缩写） 区：select \* from tab;--🡪oracle数据库

注意写的顺序，先使用数据库，再去查看有多少张表。

MySql存储引擎：

数据库对象、

存储、管理和使用数据的不同结构形式：表、视图、存储过程、函数、触发器、

事件等。

数据库、

存储数据库对象的容器。

什么是存储引擎、

什么是事务、

是指作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。

存储引擎分类。

什么是存储引擎？

#mysql在的数据用各种不同的技术存储在文件或者内存中；每种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力；通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能；不同的存储引擎性能是不一样的。

MYISAM和INNODB的区别？

#第一个它不支持事务，也不支持外键，尤其是访问速度快，对事务完整性没有要求或者以SELECT、INSERT为主的应用基本都可以使用这个引擎来创建表；每个MyISAM在磁盘上存储3个文件，其中文件名和表名都相同，但是扩展名分别为：.frm（存储表定义）；.MYD（MYData，存储数据）；.MYI（MYIndex，存储索引）。

第二个innodb存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。但是对比MyISAM的存储引擎，InnoDB写的处理效率差一些并且会占用更多的磁盘空间以保留数据和索引。

第三个innodb是不支持hash索引方法的，只支持btree。hash索引用在memory存储引擎。

常用数据类型：

Int(11)表示存储时缺少的位用零补齐。

Double(5,2)表示最多5位，其中必须有2位小数。

Char:固定长度字符串类型，char(10)表示在存储的时候，虽然字符没有十位，缺少的会用空格填充。

varchar:可变长度字符串类型，varchar(10)，有多少位就存储多少位。

Text:字符串类型，可存大量文本。

Blob：二进制类型。

Date:日期类型：年-月-日

Time:时间类型：小时：分钟：秒

Datetime:日期时间类型：年-月-日 小时：分钟：秒

字符串和日期类型都必须要用单引号括起来。

DDL：[数据定义语言](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AE%9A%E4%B9%89%E8%AF%AD%E8%A8%80)(DDL)---->>字段添加add、删除drop、修改change；字段类型的修改modify。只针对数据库对象，不参与里面数据的操作。

使用数据库：

Use 数据库名；//快捷键ctl+/：注释掉一行

创建表：

Create table 表名（列名1 列的类型 [约束]， … …，列名2 列的类型 [约束]）；

添加一列：

alter table 表名 add 列名 数据类型；

查看表里面有多少个字段：

desc students; // 字段就是列 // desc 是 describe 的简写

show tables;//查看所有的表名 区：select \* from tab;--🡪oracle数据库

int（11）与varchar(25) //前者少于十一位，就在前面补零，后者是限制字符串不超过25位。

修改一个表的字段（列名）类型：

Alter table 表名 modify 字段名 数据类型；

删除一列：

Alter table 表名 drop 字段名；（删除使用drop）

怎么修改表名：

Rename table 原始表名 to 要修改的表名；

查看表的创建细节：

Show create table 表名；

修改表的字符集为gbk:

Alter table 表名 character set 字符集名称；

修改表的列名：//可以顺带修改表的对应列里面存放的数据类型。

Alter table 表名 change 原始列名 新列名 数据类型；

删除表：

Drop table 表名；

DML：数据操纵语言---->>增加行insert into values、删除行delete from where、修改列update set where、查找数据select \* from。DML是对表中的数据进行增、删、改、查的操作。

用来操作数据库表中的记录（针对数据的操作）。

查询表中的所有数据：

Select \*（或字段名） from 表名；// Select \* from 表名\G；（未能弄明白另外一种显示方式2019-1-22）以另外一种方式呈现（如果加上where,在where子句中可以使用一些运算符及关键字）

插入操作：//注意：列名与列值的类型、个数、顺序要一一对应；值不要超出定义的长度；插入的日期和字符一样，都使用引号括起来。

Insert into 表名 （列名， 列名 。。。。） values （列值1，列值，。。。）；//可以添加对应字段的值。//都是重新添加一行值

添加多条记录：

Insert into 表名 （列名， 列名 。。。。） values //回车// （列值1，列值，。。。），//回车//（列值2，列值2，。。。）； //添加两条记录

更新操作：

Update 表名 set 列名1=列值1，列名2=列值2。。。。。。where 列名=值 //如果没有用where指定哪一行，就默认为把这个字段统一都改为一样的。

Update 表名 set 列名=列名+1 where 列名=列值； //给指定行加值（2019/4/23修改）

修改数据库密码：

Use mysql;

Update user set password=password(“密码”) where user=”root”; //这是之前的版本，现在用是不行的，user表里面就没有password这个字段了，但是password()函数还是一样。

Update mysql.user set authentication\_string=password(“密码”) where user=’root’ and host=’localhost’; //mysql.user中的mysql是指mysql这个数据库，如果已经在mysql里面就可以不加，直接字段user。

注意：只是完成上述步骤是不行的，必须刷新MySQL的系统权限相关表，使用如下命令：

Flush privileges;

mysqladmin -u root -p password 密码 //此处改密码是必须退出mysql数据库。（另外一种改密码的方式）

删除操作：

Delete from 表名 【where 列名=值】; //可以找回 //可以删除指定的，也可以删除所有的。//删除表中的数据，表结构还在。

Truncate table 表名； //找不回来 //删除所有的 。（Truncate截短）删除是把表直接drop掉，然后再创建一个同样的新表。删除的数据不能再找回。执行速度比delete快。

DQL：//Ctrl + 鼠标滚轮可以放大命令字体大小（技能）

用来查询数据的（取出数据）。

查询所有列；

结果集；

通过查询语句查询出来的数据以表的形式展示，我们称这个表为虚拟结果集。存放在内存中。查询返回的结果集是一张虚拟表。

查询指定列的数据：

Select 列名1，列名2…… from 表名；

-- SHOW TABLES;

-- DESC students;

-- SELECT \* FROM students;

-- SELECT \* FROM students WHERE age in (18,19);

-- SELECT \* FROM students WHERE age BETWEEN 18 AND 19;

SELECT \* FROM students WHERE age = 19;

条件查询就是在查询时给出where子句，在where子句中可以使用一些运算符及关键字。

条件查询运算符及关键字：

=（等于）、!=（不等于）、<>（不等于）、<、>、<=、>=;

Between …… and ……; //值在什么范围

In（set）; //固定的范围

Is NULL; (为空) is not NULL(不为空)

And; //与

Or; //或

Not; //非

模糊查询：’%内容%’

根据指定的关键进行查询：

Select from \* 表名 where 字段名 like 通配符；

使用LIKE关键字后跟通配符：

通配符：（\_任意一个字符//汉字也是可以的）（%任意0-n个字符）

字段控制查询：

去除重复记录；

Select distinct 字段 from 表名；// 关键字distinct，用于去重。

把查询字段的结果进行计算，必须都要是数据型；

Select \*，字段1 + 字段2 from 表名；//两个字段相加，生成一个新的字段。放在表的最后字段后面。

//select \*，ifnull(字段，0) + ifnull(字段，0) from 表名；

对查询结果起别名；

//select \*，ifnull(字段，0) + ifnull(字段，0) as 别名 from 表名；//as可以省略

**if(isnull(字段),0,字段)==ifnull(字段,0)**

排序：

创建表：

对查询的结果进行排序：

使用关键字order by:

排序类型：

升序asc:（从小到大，默认） Ascending

降序desc:（从大到小） Descending

使用：

CREATE TABLE `employee` (

`id` int(11) NOT NULL,

`name` varchar(50) DEFAULT NULL,

`gender` varchar(1) DEFAULT NULL,

`hire\_date` date DEFAULT NULL,

`salary` decimal(10,0) DEFAULT NULL,

`performance` double(255,0) DEFAULT NULL,

`manage` double(255,0) DEFAULT NULL,

`department` varchar(255) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

//SELECT \* FROM employee ORDER BY salary ASC; //升序asc:（从小到大，默认）

// SELECT \* FROM employee ORDER BY salary DESC,id DESC;// 降序desc:（从大到小）多字段排序，用逗号链接。

聚合函数：

注意事项：聚合函数不可以用在条件中

对查询的结果进行统计计算：

常用聚合函数：

Count（）：统计指定列不为NULL的记录行数（数据行数）；

-- SELECT \* FROM employee ORDER BY salary DESC;

-- SELECT COUNT(\*) total\_count FROM employee;

-- SELECT COUNT(performance) FROM employee;

-- SELECT COUNT(\*) total FROM employee WHERE salary > 2500;

-- SELECT \* FROM employee WHERE

-- IFNULL(salary,0)+IFNULL(performance,0) > 4000;

-- SELECT COUNT(performance) to1,COUNT(manage) to2 FROM employee;

Max()：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；

Min()：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；

Sum()：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0；

Avg()：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0；

使用：

SQL进阶部分1：

分组查询：

什么是分组查询：将查询结果按照一个或多个字段进行分组，字段值相同的为一组。

分组使用：

SELECT \* FROM employee GROUP BY gender;//根据gender字段来分组，gender字段的全部值只有两个（男和女），所以分为了两组， 当group by 单独使用时，只显示每组的第一条记录，所以group by 单独使用时的实际意义不大。

分组注意事项：

在使用分组时，select后面直接跟的字段一般都会出现在group by 后。

Group by + group\_concat()：group\_concat(字段名)可以作为一个输出字段来使用，表示分组之后，根据分组结果，使用group\_concat()来放置每一组的某字段的值的集合。

SELECT gender,GROUP\_CONCAT(NAME) FROM employee GROUP BY gender;

SELECT department,GROUP\_CONCAT(salary),SUM(salary) FROM employee GROUP BY department;

//通过group\_concat()的启发，我们既然可以统计出每个分组的某字段的值的集合，那么我们也可以通过集合函数来对这个“值的集合”做一些操作。

Group by+having：

用来分组查询后指定一些条件来输出查询结果，having作用和where一样，但having只能用于group by。

查询工资总和大于9000的部门名称以及工资和：

Having与where的区别：

Having是在分组后对数据进行过滤；

Where是在分组前对数据进行过滤；

Where后面不可以使用分组函数；

Where是对分组前记录的条件，如果某行记录没有满足where子句的条件，那么这行记录不会参加分组；而having是对分组后数据的约束。

-- SELECT department,GROUP\_CONCAT(salary),SUM(salary) FROM employee

-- GROUP BY department

-- HAVING SUM(salary) > 5000;

-- 查询工资大于2000的，工资总和大于6000的部门名称以及工资和

SELECT department,GROUP\_CONCAT(salary),SUM(salary)

from employee

WHERE salary > 1000

GROUP BY department

HAVING SUM(salary) > 5000;

书写顺序：

Select — from — where — group by — having — order by — limit

SELECT DISTINCT <select\_list>

FROM <left\_table>

<join\_type> JOIN <right\_table>

ON <join\_condition>

WHERE <where\_condition>

GROUP BY <group\_by\_list>

HAVING <having\_condition>

ORDER BY <order\_by\_condition>

LIMIT <limit\_number>

查询工资大于2000 的，工资总和大于6000的部门名称以及工资和：

LIMIT：

从哪一行开始查，总共要查几行；

Limit 参数1，参数2；//参数1：从哪一行开始查；参数2：一共要查几行。

角标是从0开始；

格式；

分页思路；

数据的完整性：

什么是数据完整性：

保证用户输入的数据保存到数据库中是正确的。

如何添加数据完整性：

在创建表时给表中添加约束。

完整性分类：（三类）

**实体完整性**：

什么是实体完整性：

表中的一行（一条记录）代表一个实体（entity）。

实体完整性的作用：

标识每一行数据不重复。行级约束。

约束类型：（三种约束类型）

主建约束(primary key)：

特点：每一个表中要有一个主建；数据唯一，且不能为NULL。

添加方式：

Create table 表名 (字段名1 数据类型 primary key, 字段名2 数据类型…);

Create table 表名 (字段名1 数据类型, 字段名2 数据类型…, primary key(字段名，字段2，。。。));（可以设置联合主建）

*联合主键就是用2个或2个以上的字段组成主键。用这个主键包含的字段作为主键，这个组合在数据表中是唯一，且加了主键索引。  
可以这么理解，比如，你的订单表里有很多字段，一般情况只要有个订单号bill\_no做主键就可以了，但是，现在要求可能会有补充订单，使用相同的订单号，那么这时单独使用订单号就不可以了，因为会有重复。那么你可以再使用个订单*[*序列号*](https://www.baidu.com/s?wd=%E5%BA%8F%E5%88%97%E5%8F%B7&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)*bill\_seq来作为区别。把bill\_no和bill\_seq设成联合主键。即使bill\_no相同，bill\_seq不同也是可以的。*

也可以1.先创建表；2.再去修改表，添加主建：

alter table 表名 add constraint primary key(字段1,字段2，。。。)；

约束

唯一约束(unique)：

特点：指定列的数据不能重复；可以为空。

格式：

create table 表名 (字段名1 数据类型，字段名2 数据类型 unique，.。。。);

自动增长列(auto\_increment)：

特点：指定列的数据字段增长；即使数据删除，还是从删除的序号继续往下。（一般主建设置自动增长列，必须是整形类型）

**域完整性**：

使用：限制此单元格的数据正确，不对照此列的其他单元格比较；域代表当前单元格。

域完整性约束：

数据类型：数值类型、日期类型、字符串类型。

非空约束（not null）：

默认值约束（default）：

CREATE TABLE stu (

id int PRIMARY KEY auto\_increment,

name VARCHAR(20) UNIQUE NOT NULL,

gender VARCHAR(1) DEFAULT '男'

);

**参照完整性**：

什么是参照完整性：是指表与表之间的一种对应关系；通常情况下可以通过设置两表之间的主建、外键关系，或者编写两表的触发器来实现；有对应参照完整性的两张表格，在对他们进行数据插入、更新、删除的过程中，系统都会将被修改表格与另一张对应表格进行对照，从而阻止一些不正确的数据的操作。（增加表的安全性）

数据库的主建和外键类型一定要一致；

两个表必须得要是innodb类型；

设置参照完整性后，外键当中的内值，必须得是主建当中的内容；

一个表设置当中的字段设置为主键，设置主键的为主表：

创建表时，设置外键，设置外键的为子表：

CREATE TABLE score (sid int, score int,

CONSTRAINT sc\_st\_fk FOREIGN KEY(sid) REFERENCES stu(id)

);（第一种方法）

ALTER TABLE score ADD CONSTRAINT 外键名 FOREIGN KEY(sid) REFERENCES stu(id);

子表 主表

（第二种方法-----表已经建好了）

SQL进阶部分2：

（设置外键的那张表里面的数据必须是关联的那张表里面的数据，关联的那张表里面的字段就一份，设置外键的那张表里面的关联字段可能重复多次出现。）

表之间关系：

一对一：一夫一妻（在开发过程中比较少）

一对多关系：（一个学生可以考多个分数，语文，数学，英语）

在多的那张表里面添加外键，实现多对一。

多对多关系：一个学生可以有多个老师，而一个老师也可以有多个学生。

需要创建一个中间表-------创建学生与老师关系表（多对一，多对一，从而实现多对多）

CREATE TABLE teacher(tid INT PRIMARY KEY auto\_increment, name VARCHAR(50), age INT, gender CHAR(1) DEFAULT '男');

CREATE TABLE student(sid INT PRIMARY KEY auto\_increment, name VARCHAR(50) NOT NULL, age INT, gender CHAR(1) DEFAULT '男');

CREATE TABLE tea\_stu\_rel(tid INT, sid INT);

ALTER TABLE tea\_stu\_rel ADD CONSTRAINT fk\_tid FOREIGN KEY(tid) REFERENCES teacher(tid);

ALTER TABLE tea\_stu\_rel ADD CONSTRAINT fk\_sid FOREIGN KEY(sid) REFERENCES student(sid);（外键的名字可以不写，默认生成）

为什么要拆分表，避免大量冗余数据的出现。

**多表查询**：

合并结果集：

什么是合并结果集：合并结果集就是把两个select语句的查询结果合并到一起。

合并结果集的两种方式：UNION合并时去除重复记录

UNION ALL合并时不去除重复记录

格式：

UNION: SELECT \* FROM 表1 UNION SELECT \* FROM 表2

SELECT \* FROM 表1 UNION ALL SELECT \* FROM 表2

连接查询：

什么是连接查询：也可以叫跨表查询，需要关联多个表进行查询

什么是笛卡尔集：假设集合A={a,b}，集合B={0,1,2}，则两个集合的笛卡尔集为{(a,0), (b,0), (a,1) ,(b,1), (a,2), (b,2)}。可以扩展到多个集合的情况。

去除笛卡尔集：

SELECT \* FROM stu,score WHERE stu.id = score.sid; //**多表联合查询**

根据连接方式分类：（内连接、外连接、自然连接三类）

**内链接**：（分为三类：等值连接；非等值连接；自连接。）

等值连接：两个表同时出现的id号（值）才显示；

SELECT \* FROM stu st INNER JOIN score sc ON st.id = sc.sid;

（INNER可以省略）

与多表联查约束主外键是一样，只是写法改变了。

ON后面只写主外键；如果还有条件直接在后面写where；多表联查后面还有条件就直接写and。

SELECT \* FROM stu

INNER JOIN score

ON stu.id = score.sid

WHERE score.score > 70

AND stu.age > 20; //**内链接查询**

**外连接**：（分为三类：左外连接（左连接）；右外连接（右连接）；多表连接。）

左连接：两表满足条件相同的数据查出来，如果左边表当中有不相同的数据，也把左边表当中的数据查出来。（左边表当中的数据全部查出，右边表当中，只查出满足条件的内容。）

SELECT \* FROM stu

LEFT OUTER JOIN score

outer可以省略

ON stu.id = score.sid;



**自然连接**：连接查询会产生无用笛卡尔集，我们通常使用主外键关系等式来去除它；而自然连接无需你去给出主外键等式，它会自动找到这一等式；也就是说不用去写条件。前提条件或要求：（两张连接的表中列名称和类型完全一致作为条件会去除相同的列。）

**子查询**：

什么是子查询：一个select语句中包含另一个完整的select语句；或两以上select，那么就是子查询语句了。

子查询出现的位置：where后，把select查询出的结果当作另一个select的条件；from后，把查询出的结果当作一个新表。

SELECT ename,deptno FROM emp

WHERE deptno = (SELECT deptno FROM emp WHERE ename = '项羽');

SELECT ename,salary FROM

(SELECT ename,salary,deptno FROM emp WHERE deptno = 30) s

WHERE s.salary > 2000;

工资高于30号部门所有人的员工信息

1.先查出30号部门工资最高的那个人

SELECT MAX(salary) FROM emp WHERE deptno = 30;

2.再到整个表中查询大于30号部门工资最高的那个人

SELECT ename,salary FROM emp

WHERE salary > (SELECT MAX(salary) FROM emp WHERE deptno = 30);

查询工作和工资与妲己完全相同的员工信息

SELECT job,salary FROM emp WHERE ename = '妲己'；

SELECT \* FROM emp

WHERE (job,salary)

in (SELECT job,salary FROM emp WHERE ename = '妲己');

（下面是多表查询）

SELECT \* FROM emp e,(SELECT job,salary FROM emp WHERE ename = '妲己') r

WHERE e.job = r.job AND e.salary = r.salary;

有2个以上直接下属的员工信息

SELECT mgr,GROUP\_CONCAT(mgr),COUNT(mgr) FROM emp

GROUP BY mgr

HAVING COUNT(mgr) >= 2;

SELECT \* FROM emp

WHERE empno in (SELECT mgr,GROUP\_CONCAT(mgr),COUNT(mgr) FROM emp

GROUP BY mgr

HAVING COUNT(mgr) >= 2);

查询员工编号为7788的员工名称、员工工资、部门名称、部门地址

SELECT e.ename,e.salary,d.dname,d.'local' FROM emp e,dept d

WHERE e.deptno = d.deptno AND e.emptno = 7788;

自连接：

求7369员工编号、姓名、经理编号和经理姓名

SELECT e1.empno,e1.ename,e2.empno,e2.ename FROM emp e1,emp e2

WHERE e1.mgr = e2.empno

AND e1.empno = 7369;

SQL进阶部分3：

（包括：常用函数、事务、权限操作、索引、视图）

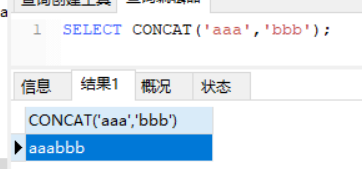
**常用函数**：

函数介绍：事先提供好的一些功能可以直接使用；函数可以用在select语句及其子句；也可以用在update，delete语句当中。

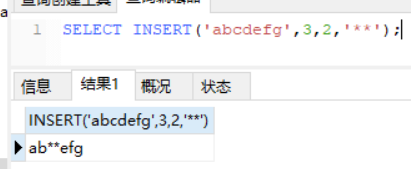
函数分类：字符串函数、数值函数、日期和时间函数、流程函数、其他常用函数。

字符串函数：

concat(s1,s2,…sn):将传入的字符连接成一个字符串；任何字符串与null进行连接结果都是null。



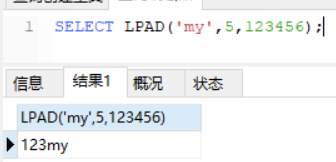
insert(str,x,y,instr):将字符串str从x位置开始，y个字符串的子串替换为指定的字符。



LOWER(str)和UPPER(str):将字符串转成小写或大写。

LEFT(str,x)和RIGHT(str,x):分别返回字符串最左边的x个字符和最右边的x个字符；如果第二个参数为null，那么不返回任何字符。

LPAD(str,n,pad)和RPAD(str,n,pad)用字符串pad对str最左边或最右边进行填充，直接到长度为n个字符长度。（padding：填充）



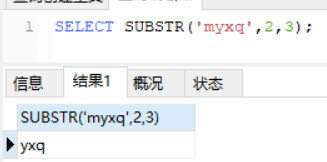
LTRIM(str)和RTRIM(str):去掉字符串当中最左侧和最右侧的空格。

TRIM(str):去掉字符串左右的空格。（trim：修剪）

REPEAT(str,x):返回str重复x次的结果。

REPLACE(str,a,b):用字符串b替换字符串str中所有出现的字符串a。

SUBSTRING(str,x,y):返回字符串str中第x位置起y个字符长度的字符。



数值函数：

ABS(x):返回x的绝对值。（Absolute value：绝对值）

CEIL(x):小数不为零部分上取整，即向上取最近的整数。

FLOOR(x): 小数不为零部分下取整，即向下取最近的整数。

MOD(X,Y):返回X/Y的模；也就是求余数。

RAND():返回0-1内容的随机值。



日期和时间函数：

CURDATE():

CURTIME():

NOW():

UNIX\_TIMESTAMP():

FROM\_UNIXTIME(unixtime):

WEEK(DATE):

YEAR(DATE):

HOUR(TIME):

MINUTE(TIME):

DATE\_FORMAT(date,fmt):

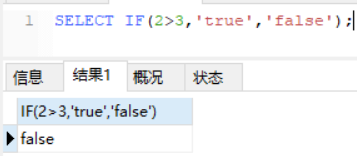
DATE\_ADD(date,interval expr type):

DATEDIFF(date1,date2):



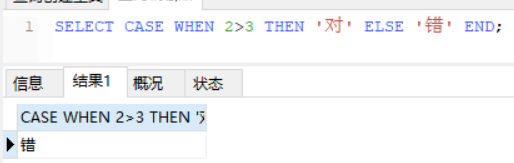
流程函数：

IF(value,t,f):如果value是真，返回t，否则返回f。



IFNULL(value1,value2):如果value1不为空，返回value1，否则返回value2。

CASE WHEN THEN ELSE END:



其他常用函数：

DATABASE():返回当前数据库名。

VERSION():返回当前数据库版本。

USER():返回当前登陆用户名。

PASSWORD(STR):对STR进行加密。

MD5():返回str的MD5值。

**事务**：

（什么是事务：事务的ACID：事务的使用：执行流程：事务的并发问题：事务隔离级别：查看隔离级别：设置隔离级别：）

什么是事务：不可分割的操作，假设该操作有ABCD四个步骤组成；若ABCD四个步骤都成功完成，则认为事务成功。若ABCD中任意一个步骤操作失败，则认为事务失败；每条sql语句都是一个事务；事务只对DML语句有效，对DQL无效。

事务的ACID：（巧记acid：酸）

原子性（atomicity）:一个事务必须被视为一个不可分割的最小工作单元，整个事务中的所有操作要么全部提交成功，要么全部失败回滚，对于一个事务来说，不可能只执行其中的一部分操作，这就是事务的原子性。

一致性（consistency）:一个商品出库时，仓库商品数量减1，对应用户的购物车中商品加1。

隔离性（isolation）: 一个事务所做的修改在最终提交以前，对其他事务是不可见的。

持久性（durability）：一旦事务提交，则其所做的修改会永久保存到数据库。

事务的使用：

开启事务：start transaction

提交事务：commit----所有语句全部执行完毕，没有发生异常，提交事务，更新到数据库当中。

回滚事务：rollback----当遇到一突发情况，撤销执行的sql语句。

START transaction;

UPDATE zs\_count SET money = money - 2000;

UPDATE ls\_count SET money = money + 2000;

COMMIT;

START transaction;

UPDATE zs\_count SET money = money - 2000;

UPDATE ls\_count SET money = money + 2000;

ROLLBACK;

执行流程：

事务的并发问题：（脏读；不可重复读；重复读；幻读；对应关系）

脏读：

*A事务读取B事务尚未提交的数据，此时如果B事务发生错误并执行回滚操作，那么A事务读取到的数据就是脏数据。就好像原本的数据比较干净、纯粹，此时由于B事务更改了它，这个数据变得不再纯粹。这个时候A事务立即读取了这个脏数据，但事务B良心发现，又用回滚把数据恢复成原来干净、纯粹的样子，而事务A却什么都不知道，最终结果就是事务A读取了此次的脏数据，称为脏读。*

解决办法：read committed！读提交，能解决脏读问题。

不可重复读：

*程序员拿着工资卡（卡里当然是只有3.6万），当他买单时（程序员事务开启），收费系统事先检测到他的卡里有3.6万，就在这个时候！！程序员的妻子要把钱全部转出充当家用，并提交。当收费系统准备扣款时，再检测卡里面的金额，发现已经没钱了。程序员就会很郁闷，明明卡里是有钱的…*

一个事务范围内两个相同的查询却返回了不同数据，这就是不可重复读；解决办法：repeatable read

重复读：

*程序员拿着工资卡（卡里还是有3.6万），当他买时（事务开启，不允许其他事务的update修改操作），收费系统事先检测到他的卡里有3.6万。这个时候他的妻子不能转出金额了。接下来的收费系统就可以扣款了。*

幻读：

*程序员某一天区消费，花了2千元，然后他的妻子去查看他今天的消费记录，（他妻子事务开启），看到确实是花了2千元，就在这个时候，程序员花了1万买了一部电脑，即新增insert了一条消费记录，并提交。当妻子打印程序员的消费记录清单时（妻子事务提交），发现花了1.2万元，似乎出现了幻觉，这就是幻读。*

解决办法：serializable（但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。）

对应关系：



事务隔离级别：（read uncommitted;read committe;repeatable read;serializable）

read uncommitted:就是一个事务可以读取另一个未提交事务的数据。

read committee：一个事务要等另一个事务提交后才能读取数据。

repeatable read:系统默认级别----重复读

查看隔离级别：

设置隔离级别：

• 事务隔离级别的作用范围分为两种：

– 全局级：对所有的会话有效

– 会话级：只对当前的会话有效

• 例如，设置会话级隔离级别为READ COMMITTED ：

mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED；

或：

mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED；

• 设置全局级隔离级别为READ COMMITTED ：

mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED；

**权限操作：**

**视图：**

**存储过程：**

什么是存储过程：一组可编程的函数，是为了完成特定功能的SQL语句集；存储过程就是具有名字的一段代码，用来完成一个特定的功能；创建的存储过程保存在数据库的数据字典中。

为什么要用存储过程：将重复性很高的一些操作，封装到一个存储过程中，简化了对这些SQL的调用；批量处理；统一接口，确保数据的安全；相对于Oracle数据库来说，Mysql的存储过程相对功能较弱，使用较少。

存储过程的创建和调用：

DELIMITER $$:它与存储过程语法无关；DELIMITER语句将标准分隔符-符号（；）更改为：$$；因为我们想将存储过程作为整体传递给服务器；而不是让MySQL工具一次解释每个语句；告诉MySQL解释器，该段命令是否已经结束了，MySQL是否可以执行了。默认情况下，delimiter是分号。在命令行客户端中。如果有一行命令以分号结束，那么回车后，MySQL将会执行该命令。但有时候，不希望MySQL这么做。在为可能输入较多的语句，且语句中包含有分号。使用delimiter $$，这样只有当$$出现之后，MySQL解释才会执行这段语句。

创建存储过程：

delimiter $$ // delimiter:分隔符

CREATE PROCEDURE show\_stu()

BEGIN

SELECT \* FROM stu;

END $$

delimiter ;

调用存储过程：

CALL show\_stu();

查看存储过程：

查看所有存储过程：SHOW PROCEDURE STATUS;

查看指定数据库中的存储过程：

SHOW PROCEDURE STATUS WHERE db = 'my\_test';

查看指定存储过程源代码：SHOW CREATE PROCEDURE 函数名;

Show create table 表名；//查看表的创建细节

删除存储过程：DROP PROCEDURE show\_stu;

存储过程变量：

在存储过程中声明一个变量：

使用DECLARE语句：

delimiter $$

CREATE PROCEDURE show\_stu()

BEGIN

DECLARE 变量名 数据类型（大小） defaul 默认值；

SELECT \* FROM stu;

END $$

（可以什么一个名为total\_sale的变量，数据类型为INT，默认值为0；声明共享相同数据类型的两个或多个变量：

DECLARE 变量名1，变量名2 数据类型（大小） defaul 默认值；）

分配变量值：要为变量分配一个值，可以使用set语句；使用select into语句将查询的结果分配给一个变量。

delimiter $$

CREATE PROCEDURE show\_stu()

BEGIN

-- 声明变量

DECLARE res VARCHAR(255) DEFAULT '';

DECLARE x,y INT DEFAULT 0;

SET x = 3;

SET y = 4;

DECLARE avgRes DOUBLE DEFAULT 0;

SELECT AVG(salary) INTO avgRes FROM employee;

END $$

变量的范围：

存储过程参数：

三种类型：（in、out、inout）

-- 根据传入的名称，获取对应的信息

delimiter $$

CREATE PROCEDURE getName(in name varchar(255))

BEGIN

SELECT \* FROM employee WHERE ename = name;

END $$

delimiter ;

CALL getName('张三');

-- 给我们一个名称，可以把它的薪资给你

delimiter $$

CREATE PROCEDURE getSalary(in n varchar(255),OUT osalary INT)

BEGIN

SELECT salary INTO osalary FROM employee WHERE ename = n;

END $$

delimiter ;

CALL getSalary('张三'，@s);

SELECT @s FROM DUAL;

***补充****：在Mysql和SQL Server中可以直接：*

*select 1; 不需要加from 表名就可以执行*

*而在Oracle中要满足select from 的结构：*

*select 1不能实现满足上述结构，引入dual来满足结构。*

*Oracle这样表示：select 1 from dual;*

*注：上述的数据库中都存在dual表，只是mysql中的dual表没有值select \* from dual; 这种语句会报错，而在Oracle中会返回一行记录（dual表有一条默认值），二者都可以用dual来满足语法结构。*

delimiter $$

CREATE PROCEDURE test1(INOUT num INT,in inc INT)

BEGIN

SET num = num + inc;

END $$

delimiter ;

SET @num1 = 20;

CALL test1(@num1, 10);

SELECT @num1;

定义参数：

存储语句：

IF语句：

CASE语句：

循环：

**自定义函数**：

-- 随机生成一个指定个数的字符串

delimiter $$

CREATE FUNCTION rand\_str(n INT) RETURNS VARCHAR(255)

BEGIN

-- 声明一个str 52字母

DECLARE str VARCHAR(100) DEFAULT 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';

-- 记录当前是第几个

DECLARE i INT DEFAULT 0;

-- 生成的结果

DECLARE res\_str VARCHAR(255) DEFAULT '';

WHILE i < n DO

-- 随机生成一个指定个数的字符串

-- RAND()\*52 + 1 FLOOR(RAND()\*52 + 1)

-- SUBSTR(str,FLOOR(RAND()\*52 + 1),1);

SET res\_str = CONCAT(res\_str,SUBSTR(str,FLOOR(RAND()\*52 + 1),1));

SET i = i + 1;

END WHILE;

RETURN res\_str;

END $$

delimiter ;

-- 调用自定义函数的方法：

SELECT rand\_str(5);

* 往表中插入海量数据

delimiter $$

CREATE PROCEDURE insert\_emp(in startNum INT, in max\_num INT)

BEGIN

-- 声明一个变量记录当前是第几条数据

DECLARE i INT DEFAULT 0;

-- 默认情况是自动提交sql

SET autocommit = 0; -- 不让它自动提交sql

REPEAT

SET i = i + 1;

-- 插入数据

INSERT INTO tmp VALUES(startNum + i,rand\_str(5),FLOOR(10+RAND()\*10));

UNTIL i = max\_num

END REPEAT;

COMMIT;-- 整体提交所有的sql 提高效率

END $$

delimiter ;

**索引**：

什么是索引：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一个索引，MySQL能够快速到达一个位置区搜索数据文件，而不必查看所有数据，那么将会节省很大一部分时间。

索引的优势与劣势：

优势：类似大学图书馆建书目录索引，提高数据检索效率，降低数据库的IO成本。通过索引对数据进行排序，降低数据排序的成本，降低CPU的消耗。

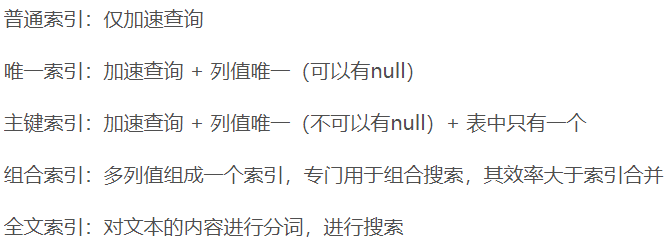
劣势：实际上索引也是一张表，该表保存了主建与索引字段，并指向实体表的记录，索引列也是要占空间的。虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT\UPDATE\DELETE。

索引不适合的情况：1.少量数据

2.频繁更新的字段

3.很少使用的字段

索引的分类：



（单值索引：唯一索引：复合索引：全文索引：空间索引：）

全文索引和空间索引只能用在MyISAM引擎上，只能在CHAR\VARCHAR\TEXT类型字段上使用全文索引。

索引操作：（创建、删除、查看、自动创建索引）

创建：

CREATE INDEX 索引名称 ON 表名(字段名); //INDEX默认是单值索引

EXPLAIN：模拟Mysql优化器是如何执行SQL查询语句的，从而知道Mysql是如何处理你的SQL语句的。分析你的查询语句或是表结构的性能瓶颈。

EXPLAIN SELECT \* FROM tmp WHERE id = 101\G;

索引结构：（先会对数据进行排序，btree索引，hash索引。）

B+树是一个平衡的多叉树，从根节点到每个叶子节点的高度差值不超过1，而同层级的节点间有指针相互连接。

HASH索引就是采用一定的哈希算法，把键值换算成新的哈希值，检索时不需要类似B+树那样从根节点到叶子节点逐级查找，只需一次哈希算法即可立刻定位到相应的位置，速度非常快。

hash索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree索引需要从根节点到枝节点，最后才能访问到叶节点这样多次的IO访问，所以hash索引的查询效率要远远高于B-Tree索引。

哪些情况需要创建索引：

哪些情况不需要创建索引：

B+树中查询任意的数据次数：n次（B+树的高度）

创建索引：（DDL语句，事务是针对DML语句，DML才需要commit;提交）

方式一：

create 索引类型 索引名 on 表名（字段）；

index----单值索引

unique index-----唯一索引

index---------（字段1，字段2）------多字段，自动成复合索引

方式二：

alter table 表名 add 索引类型 索引名字（字段）；

index -----单值索引

unique index -----唯一索引

index------（字段1，字段2）---复合索引

删除索引：

drop index 索引名 on 表名；

查询索引：

show index from 表名；

select tc.tcdesc from teacherCard tc,course c,teacher t

where c.tid = t.tid

and t.tcid = tc.tcid

and c.cname = ‘sql’;

将上述多表查询转为子查询形式

select tc.tcdesc from teacherCard tc where tc.tcid =

(select t.tcid from teacher t where t.tid =

(select c.tid from course c where c.cname = ‘sql’)

);

(2)select\_type查询类型

PRIMARY:包含子查询SQL中的 主查询 （最外层）

SUBQUERY:包含子查询SQL中的 子查询 （非最外层）

simple:简单查询（不包含子查询、union）

derived:衍生查询（使用到了临时表）

a.在from子查询中只有一张表

b.在from子查询中，如果有table1 union table2，则table1就是derived，table2就是union

(3)type：索引类型

system->const->eq-ref->ref->range->index->all //越往左边，性能越高

all：查询全部表中的数据

index：查询全部索引中的数据

range：

ref：

eql-ref：

(4)possible\_keys：可能用到的索引

(5)key：实际使用到的索引

(6)key\_ken：索引的长度 //utf8:一个字符占三个字节；

//gbk:一个字符两个字节；

//latin:一个字符一个字节。