

MYSQL数据库

MYSQL的三层结构

MYSQL的三层结构



所用的数据库终端程序本质就是向3306端口通过网络发送指令。

DBMS的主体程序是mysql.exe，监听3306端口，并执行来自3306端口的命令。(3306端口是数据库的默认端口，在实际业务中，为了防止攻击，可以换成其他的端口)

数据库-普通表的本质依然是文件（所有的数据库最终都要进行数据持久化）。

所谓安装mysql数据库，就是在主机安装一个数据库管理系统(DBMS),这个管理系统可以管理多个数据库。

数据在表中的存储方式

表的一行称为一条记录，在java程序中，往往使用一个对象。

SQL语句分类

- DDL: 数据定义语句[create 表, 库...]
- DML: 数据操作语句[增加insert,修改update,删除delete]
- DQL: 数据查询语句[select]
- DCL: 数据控制语句[管理数据库：比如用户权限 grant revoke]

创建数据库

```
CREATE DATABASE [IF NOT EXISTS] db_name [DEFAULT] CHARACTER SET charest_name
[DEFAULT]COLLATE collation_name
/*[]里面的代表可写可不写,mysql会默认填写*/
```

1. CHARACTER SET: 指定数据库采用的字符集，如果不指定字符集，默认utf8。
2. COLLATE: 指定数据库字符集的校对规则(常用的utf8_bin[区分大小写]、utf8_general_ci[不区分大小写])，默认使用utf8_general_ci。

例：

```
#演示数据库的操作
#创建一个名称为db01的数据库
CREATE DATABASE db01;
#创建一个使用utf8字符集的db02数据库
CREATE DATABASE db02 CHARACTER SET utf8;
#创建一个使用utf8字符集，并带校对规则的db03数据库
CREATE DATABASE db03 CHARACTER SET utf8 COLLATE utf8_bin;
/*TIPS: 当创建一张表时，也可以指定校对规则，如果没有指定，则默认使用数据库的校对规则*/
/*一个表的校对规则可以与其数据库的规则相违背（比如数据库的校对规则是表的真子集），但是这样写可能导致不好的结果*/
```

查看、删除数据库

```
#显示数据库语句
SHOW DATABASES;
#显示数据库创建语句
SHOW CREATE DATABASE db_name;
#数据库删除语句【慎用】（从删库到跑路）
DROP DATABASE [IF EXISTS] db_name;
```

例：

```
#演示查看和删除数据库
#查看当前数据库服务器里的所有数据库
SHOW DATABASES;
#查看db01数据库的定义信息
SHOW CREATE DATABASE db01;
#删除db01数据库
DROP DATABASE db01;
/*TIPS: 在创建数据库，表的时候，为了规避关键字，可以用反引号解决*/
CREATE DATABASE `CREATE`;
```

备份恢复数据库

```
#备份数据库（注意：在DOS执行）mysqldump指令在mysql安装目录\bin下面（使用命令行的原因）
mysqldump -u 用户名 -p密码 -B 数据库1 数据库2 数据库n > 路径名+文件名.sql
#恢复数据库（注意：进入mysql命令行再执行）
Source 路径名+文件名.sql
#TIPS: 也可以将所有sql语句粘贴进查询编辑器，再执行
```

例：

```
#备份db02,db03库中的数据，并恢复
mysqldump -u root -p -B db01 db02 > d:\\save.sql
Source d:\\save.sql
#其实后缀名不重要（学完操作系统懂得都懂）
```

创建表

```
CREATE TABLE table_name(
    field1 datatype,
    field2 datatype,
    field3 datatype
)CHARACTER SET 字符集 COLLATE 校对规则 ENGINE 引擎;
```

- field:指定列名
- datatype:指定列类型 (字段类型)
- character set:如不指定则为所在数据库字符集
- collate: 如不指定则为所在数据库校对规则
- engine:引擎

注意: 创建表时, 要根据需要保存的数据创建相应的列, 并根据数据类型定义相应的列类型。

例:

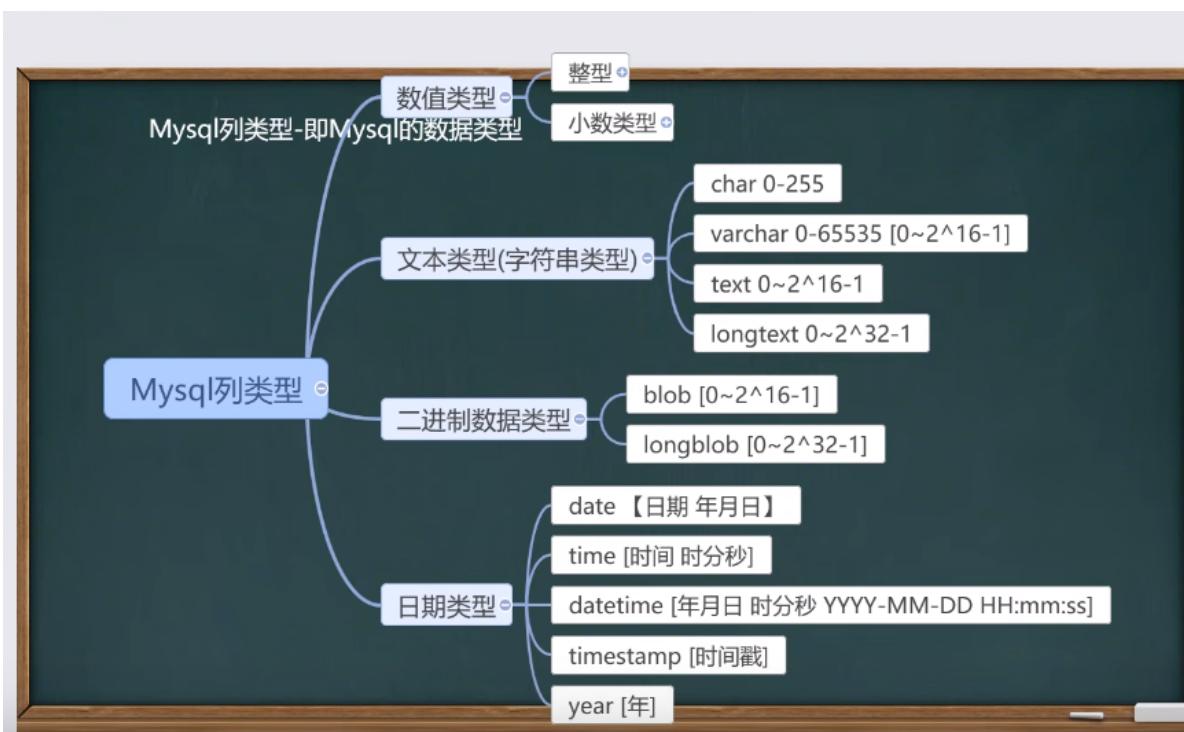
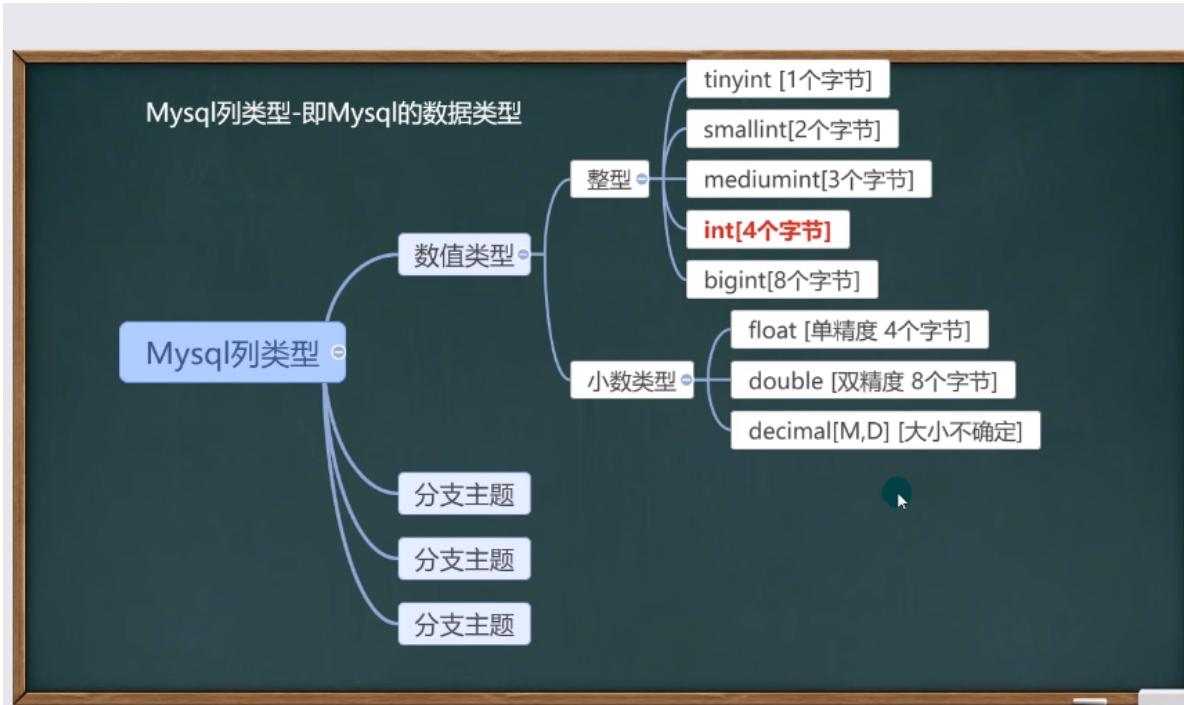
创建user表:

id	整形
name	字符串
password	字符串
birthday	日期

```
CREATE TABLE `user`(
    `id` INT,
    `name` VARCHAR(255),
    `password` VARCHAR(255),
    `birthday` DATE
)CHARACTER SET utf8 COLLATE utf8_bin ENGINE INNODB;
```

MYSQL常用数据类型

分类	数据类型	说明
数值类型	BIT(M)	位类型。M指定位数, 默认值1, 范围1-64
	TINYINT [UNSIGNED]	占1个字节 带符号的范围是-128到127。无符号0到255。默认是有符号
	SMALLINT [UNSIGNED]	2个字节 带符号是 负的 2^15 到 2^15-1 ,无符号 0 到 2^16 -1
	MEDIUMINT[UNSIGNED]	3个字节 带符号是 负的 2^23 到 2^23-1 ,无符号 0 到 2^24 -1
	INT [UNSIGNED]	4个字节 带符号是 负的 2^31 到 2^31-1 ,无符号 0 到 2^32 -1
	BIGINT [UNSIGNED]	8个字节 带符号是 负的 2^63 到 2^63-1 ,无符号 0 到 2^64 -1
文本、二进制类型	FLOAT [UNSIGNED]	占用空间4个字节
	DOUBLE [UNSIGNED]	表示比float精度更大的小数,占用空间8个字节
	DECIMAL(M,D) [UNSIGNED]	定点数 M指定长度, D表示小数点的位数,
	CHAR(size) char(20)	固定长度字符串 最大255
时间日期	VARCHAR(size) varchar(20)	可变长度字符串 0~65535 [即: 2^16-1]
	BLOB LONGBLOB	二进制数据 BLOB 0~2^16-1 LONGBLOB 0~2^32-1
	TEXT LONGTEXT	文本 Text 0~2^16 LONGTEXT 0~2^32
时间日期	DATE/DATETIME/TimeStamp	日期类型(YYYY-MM-DD) (YYYY-MM-DD HH:MM:SS), TimeStamp表示时间戳, 它可用于自动记录insert、 update操作的时间



数值型（整形）的基本使用

1. 使用规范：在满足需求的情况下，尽量选用占用空间小的类型。

类型	字节	最小值	最大值
		(带符号的/无符号的)	(带符号的/无符号的)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

例：

```
#演示TINYINT来演示范围
#1.如果没有指定unsigned，默认时有符号的。
CREATE TABLE t3(
    id TINYINT UNSIGNED
);
INSERT INTO t3 VALUES(256); #超过范围无法添加
```

数值型 (bit) 的基本使用

例：

```
#演示BIT类型的使用
#1、bit(m) m 在1-64
#2、显示按照bit方式显示(二进制方式显示)

CREATE TABLE t5(
    num BIT(8)
);
INSERT INTO t5 VALUES(255);
SELECT * FROM t5;
SELECT * FROM t5 WHERE num = 255; #仍然能按照十进制筛选
```

数值型 (浮点数) 的基本使用

1.FLOAT/DOUBLE [UNSIGNED] FLOAT单精度， DOUBLE双精度。

2.DECIMAL[M,D] [UNSIGNED]

- 可以支持更加精确的小数位。M是小数位数（精度）的总数，D是小数点后面的位数（标度）。
- 如果D是0，则值没有小数点或者分数部分，M最大65，D最大30。如果D被省略，默认是0；如果M被省略，默认是10。
- 建议：如果希望小数的精度高，推荐使用decimal。

```
#演示decimal类型、float类型、double使用
```

```
CREATE TABLE t6(
    num1 FLOAT,
    num2 DOUBLE,
    num3 DECIMAL(30,20)
);

INSERT INTO t6 VALUES(88.12345678912345,88.12345678912345,88.12345678912345);
#decimal 会在不够的小数位数后面补0
INSERT INTO t6
VALUES(88.12345678912345,88.12345678912345,88.123456789123451234519);#decimal 也会四舍五入
SELECT * FROM t6;
```

num1	num2	num3
88.1235	12345678912345.12345678912345000000	
88.1235	12345678912345.12345678912345123452	

字符串的基本使用

CHAR(size) 固定长度字符串 最大255字符

VARCHAR(size) 可变长度字符串 最大65532字节 [utf8编码最大21844字符 1-3个字节用于记录大小]

```
#演示字符串类型使用 CHAR VARCHAR
```

```
CREATE TABLE t9(
    `name` CHAR(255)
);

#如果表的编码是 utf8 varchar(size) size = (65535 - 3) / 3 = 21844
#如果表的编码是 gbk varchar(size) size = (65535 - 3) / 2 = 32766
CREATE TABLE t10(
    `name` VARCHAR(21844) CHARSET gbk
);
```

字符串的使用细节

1.CHAR(4) 这个4表示字符数，不是字节数，不管是中文还是字母都是放四个，按字符计算。

VARCHAR(4) 这个4也表示字符数，不管是字母还是中文都以定义好的表的编码来存放数据。

不管是中文还是英文，都是最多存放4个，是按照字符来存放的。

```
#演示字符串类型的使用细节
```

```
#CHAR(4) 和 VARCHAR(4)这个4表示的是字符，而不是字节 不区分汉字和字母
```

```
CREATE TABLE t11(
    `name` CHAR(4)
);

INSERT INTO t11 VALUES('嘉可莉然');
```

```
SELECT * FROM t11;

CREATE TABLE t12(
    `name` VARCHAR(4)
);

INSERT INTO t12 VALUES('刻向晚晴');
INSERT INTO t12 VALUES('海ybb');
SELECT * FROM t12;
```

2.CHAR(4) 是定长(固定的大小), 即使插入少于4个字符, 也会分配四个字符的空间。

VARCHAR(4) 是变长, 按照实际占用内存来分配。(VARCHAR本身还需要1-3个字节来记录存放内容长度)

3.什么时候使用CHAR, 什么时候使用VARCHAR?

1. 如果数据是定长, 推荐使用char, 比如md5的密码, 邮编, 手机号, 身份证号码等。
2. 如果一个字段的长度不确定, 我们使用varchar, 比如留言, 文章。
3. 查询速度: char > varchar

4.在存放文本时, 也可以使用TEXT数据类型。可以将TEXT视为VARCHAR列。注意TEXT不能有默认值, 大小为0-2^16 字节。如果希望存放更多字符, 可以选择MEDIUMTEXT 0-2^24 或者 LONGTEXT 0-2^32。

```
#如果VARCHAR不够用, 可以使用TEXT。

CREATE TABLE t13(
    content TEXT
);
INSERT INTO t13 VALUES('一个人过五一。'); #不同字符所占空间不同
```

日期类型的基本使用

```
#演示时间相关的内容
CREATE TABLE t14(
    birthday DATE,
    job_time DATETIME,
    login_time TIMESTAMP NOT NULL
        DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP
        #如果希望login_time列自动更新, 需要配置
);

INSERT INTO t14(birthday,job_time) VALUES ('2022-11-11','2022-11-11 10:10:10');
#如果我们更新 t14表的某条记录, login_time列会自动的以当前时间进行更新(更新不是指插入新数据)
```

修改表

使用ALTER TABLE 语句追加, 修改, 或删除列的语法。

添加列:

```
ALTER TABLE tablename
ADD
    column datatype [DEFAULT expr]
    [, column datatype]...
;
```

修改列:

```
ALTER TABLE tablename
MODIFY
    column datatype [DEFAULT expr]
    [, column datatype]...
;
```

删除列:

```
ALTER TABLE tablename
DROP column;
```

查看表的结构: desc 表名;

修改表名: rename table 表名 to 新表名;

修改表字符集: alter table 表名 character set 字符集;

修改列名: ALTER TABLE table_name CHANGE 原表名 新表名 datatype;

例:

● 应用实例 **altertab.sql**

- 员工表emp的上增加一个image列, varchar类型(要求在resume后面)。
 - 修改job列, 使其长度为60。
 - 删除sex列。
- 表名改为employee。
 - 修改表的字符集为utf8
 - 列名name修改为user_name

alter table user change column name username varchar(20);

给5min学生练习下

```
#注意不要加括号
#员工表的emp上增加一个image列， varchar类型(要求在resume后面)。
ALTER TABLE emp ADD image VARCHAR(255) NOT NULL DEFAULT '' AFTER `resume`;
#修改job列，使其长度为60。
ALTER TABLE emp MODIFY job VARCHAR(60);
#删除sex列
ALTER TABLE emp DROP sex;
#表名改为employee
RENAME TABLE employee TO emp
#修改表的字符串为utf8
ALTER TABLE employee CHARACTER SET utf8
#列名name 修改为 user_name
ALTER TABLE employee CHANGE `name` `user_name` VARCHAR(64) NOT NULL DEFAULT '';
```

数据库C[create]R[read]U[update]D[delete]语句

1.INSERT 语句(添加数据)

2.UPDATE 语句(更新数据)

3.DELETE语句(删除数据)

4.SELECT语句(查找数据)

INSERT语句

```
INSERT INTO table_name(column1,column2,column3)
VALUES(value1,value2,value3)
```

INSERT语句细节

- 插入的数据应与字段的数据类型相同，否则会报错（如果在整型中插入字符串，mysql会尝试将字符串转化为整形，插入'80'不会报错）
- 数据的长度应该在列的规定范围内。
- 在values中列出的数据位置必须与被加入的列的排列位置相对应。
- 字符和日期型数据应该包含在单引号内。
- 列可以插入空值(前提是该字段允许为空),insert into table value(null)。
- insert into table_name(列名..)values (),(),()形式添加多条记录。
- 如果是给表中所有字段添加数据，可以不写前面的字段名称。
- 默认值的使用，当不给某个字段值时，如果有默认值就会添加，否则就会报错。

如果某个列 没有指定 not null，那么当添加数据时，没有给定值，则会默认给NULL。

如果我们希望指定某个列的默认值，可以在创建表时指定 DEFAULT 默认值。

UPDATE语句

```
UPDATE table_name SET col_name = expr1;
```

例：

```
#将所有员工的薪水修改为5000
UPDATE emp SET salary = 5000;
#将姓名为 小妖怪 的员工薪水修改为 3000
UPDATE emp SET salary = 3000
    WHERE user_name = '小妖怪';
#将姓名为 老妖怪 的薪水在原来的基础上增加1000
UPDATE emp SET salary = salary + 3000
    WHERE user_name = '老妖怪';
```

使用细节：

1. UPDATE语句可以用新值更新原有表行中的各列。
2. SET语句指示要修改那些列和要给予哪些值
3. WHERE子句指定应该更新哪些行。如没有WHERE自己，则更新所有的行。
4. 如果要修改多个字段，可以通过set 字段1 = 值1, 字段2 = 值2.

DELETE语句

使用DELETE语句删除表中数据

```
DELETE FROM table_name WHERE where_definition;
```

例：

```
#删除表中名称为'老妖怪'的记录
DELETE FROM emp
    WHERE user_name = '老妖怪';
#删除表中所有记录
DELETE FROM emp;
```

使用细节：

- 1.如果不使用WHERE语句，将删除表中所有数据。
2. DELETE语句不能删除某一列的值(可使用update 设为 null 或者 "")
3. 使用DELETE语句仅删除记录，不删除表本书。如要删除表，使用DROP TABLE语句。

SELECT语句

```
SELECT [DISTINCT] * | {column1, column2, column3} FROM tablename;
```

使用表达式对查询的列进行运算

```
SELECT [DISTINCT] * | {column1 | expression , column2 | expression} FROM tablename;
```

在SELECT语句中可以使用as语句。

```
SELECT column_name AS 别名 FROM 表名;
```

例：

```

#统计每个学生的总分
SELECT `name`,(chinese+english+math)  FROM student;
#在所有学生总分+10分
SELECT `name`,(chinese+english+math + 10)  FROM student;
#使用别名表示学生分数
SELECT `name`,(chinese+english+math) AS grade FROM student;

```

在WHERE语句中经常使用的运算符

● 在where子句中经常使用的运算符

比较运算符	> < <= >= = <> !=	大于、小于、大于(小于)等于、不等于
	BETWEEN ...AND...	显示在某一区间的值
	IN(set)	显示在in列表中的值, 例: in(100,200)
	LIKE '张pattern' NOT LIKE ''	模糊查询 模糊查询
	IS NULL	判断是否为空
逻辑运算符	and	多个条件同时成立
	or	多个条件任一成立
	not	不成立, 例: where not(salary>100);

'韩%'表示开头是韩就可以。

使用ORDER BY子句查询排序结果。

```

SELECT column,column2,column3..
  FROM table
  ORDER BY column ASC|DESC;

```

1. ORDER BY指定排序的列, 排序的列既可以是表中的列名, 也可以是SELECT语句后指定的列名。
2. ASC升序[默认], DESC降序。
3. ORDER BY语句位于SELECT语句的末尾

合计/统计函数 -count

```

SELECT count(*) | count(列名) from table_name
[WHERE where_definition]

```

COUNT(*) 和 COUNT(列)的区别:

COUNT(*)返回满足条件的记录的行数

COUNT(列)统计满足条件的某列值有多少个, 但是会排除为null的情况。

合计函数 - sum

SUM函数返回满足where条件行的和

```
SELECT SUM(列) FROM table_name  
[WHERE where_definition];
```

SUM仅对数值起作用，否则没意义。

合计函数-avg

```
SELECT AVG(列) FROM table_name  
[WHERE where_definition];
```

合计函数-MAX/MIN

```
SELECT MAX/MIN(列) FROM table_name  
[WHERE where_definition];
```

GROUP BY 子句

使用GROUP BY语句对列进行分组

```
SELECT column1,column2,column3 FROM table  
GROUP BY column;
```

使用HAVING子句对分组后的结果进行过滤

```
SELECT column1,column2,column3 FROM table  
GROUP BY column HAVING ...;
```

字符串相关函数

CHARSET(str)	返回字串字符集
CONCAT (string2 [...])	连接字串
INSTR (string ,substring)	返回 substring 在 string 中出现的位置,没有返回0
UCASE (string2)	转换成大写
LCASE (string2)	转换成小写
LEFT (string2 ,length)	从 string2 中的左边起取 length 个字符
LENGTH (string)	string 长度【按照字节】
REPLACE (str ,search_str ,replace_str)	在 str 中用 replace_str 替换 search_str
STRCMP (string1 ,string2)	逐字符比较两字串大小,
SUBSTRING (str , position [,length])	从 str 的 position 开始【从1开始计算】,取 length 个字符
LTRIM (string2) RTRIM (string2) trim	去除前端空格或后端空格

例：

```
# 以首字母小写的方式显示所有员工emp表的姓名
SELECT REPLACE(`ename`,LEFT(`ename`,1),LCASE(LEFT(`ename`,1))) FROM emp
```

数学相关函数

ABS(num)	绝对值
BIN (decimal_number)	十进制转二进制
CEILING (number2)	向上取整, 得到比 num2 大的最小整数
CONV(number2,from_base,to_base)	进制转换
FLOOR (number2)	向下取整, 得到比 num2 小的最大整数
FORMAT (number,decimal_places)	保留小数位数
HEX (DecimalNumber)	转十六进制
LEAST (number , number2 [...])	求最小值
MOD (numerator ,denominator)	求余
RAND([seed])	RAND([seed]) 其范围为 $0 \leq v \leq 1.0$

进制转换:

```
#10进制的8 转为 2进制的8
#DUAL 是mysql 内置元表 可用于测试
SELECT CONV(8,10,2) FROM DUAL;
```

保留小数位数:

```
#前面写小数 后面写保留位数 自动四舍五入
SELECT FORMAT(78.12345678,2) FROM DUAL;
```

随机数:

```
#如果希望产生的随机数不变化 可以在里面填入种子
SELECT RAND([seed]) FROM DUAL;
```

时间日期相关函数

CURRENT_DATE()	当前日期
CURRENT_TIME()	当前时间
CURRENT_TIMESTAMP()	当前时间戳
DATE(datetime)	返回 datetime 的日期部分
DATE_ADD(date2, INTERVAL d_value d_type)	在 date2 中加上日期或时间
DATE_SUB(date2, INTERVAL d_value d_type)	在 date2 上减去一个时间
DATEDIFF(date1, date2)	两个日期差(结果是天)
TIMEDIFF(date1, date2)	两个时间差(多少小时多少分钟多少秒)
NOW()	当前时间
YEAR Month DATE(datetime) FROM_UNIXTIME()	年月日

例：

```
#查询在十分钟内发布的新闻
SELECT * FROM mes
WHERE DATE_ADD(send_time, INTERVAL 10 MINUTE) >= NOW()
```

1. DATE_ADD()中的 interval 后面可以是 year minute second day 等
2. DATE_SUB() 中的 interval 后面可以是 year minute second day 等
3. DATEDIFF(date1,date2) 得到的是天数，而且是date1-date2的天数，因此可以取负数
4. 这四个函数的日期类型可以是date,datetime 或者 timestamp

UNIX_TIMESTAMP() 返回的是1990-1-1 到现在的秒数

```
SELECT UNIX_TIMESTAMP() FROM DUAL;
```

FROM_UNIXTIME() 可以把一个 UNIX_TIMESTAMP() 秒数，转换为一个指定格式的日期。

在开发中可以用一个整数来保存时间，然后通过FROM_UNIXTIME() 进行转换。

```
#默认是DATETIME格式
SELECT FROM_UNIXTIME(UNIX_TIMESTAMP()) FROM DUAL;
#可以设定格式
SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y-%m-%d') FROM DUAL;
SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y-%m-%d %H:%i:%s') FROM DUAL;
```

加密函数和系统函数

USER()	查询用户
DATABASE()	数据库名称
MD5(str)	为字符串算出一个 MD5 32的字符串，(用户密码)加密
PASSWORD(str) select * from mysql.user \G	从原文密码str 计算并返回密码字符串,通常用于对mysql 数据库的用户密码加密

#演示加密函数和系统函数

#USER() 查询用户

#可以查看登录到MYSQL的有哪些用户，以及登录的IP

SELECT USER() FROM DUAL; #用户@IP地址

#DATABASE() 查询当前使用数据库名称

SELECT DATABASE() FROM DUAL;

#MD5(str) 为字符串算出一个 MD5 32 位的字符串，常用(用户密码)加密

#root密码 -> 加密md5 -> 在数据库存放的是加密的密码

SELECT MD5('CAROL') FROM DUAL;

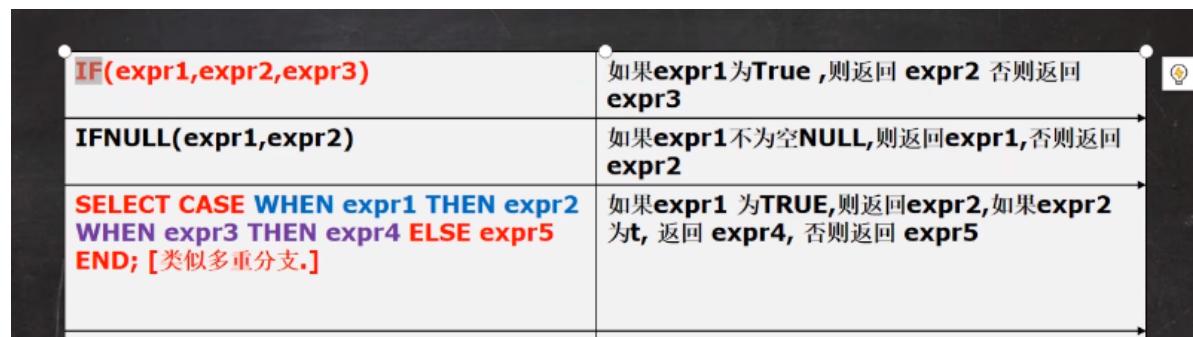
#PASSWORD(str) 另一个加密函数 41位

SELECT PASSWORD('DIANA') FROM DUAL;

#mysql 储存用户的数据库

SELECT * FROM mysql.`user`

流程控制函数



#如果是空 改为0.0

SELECT ename,IF(comm IS NULL,'0.0',comm) FROM emp;

SELECT ename,IFNULL(comm,'0.0') FROM emp;

#判断是否为空 要使用IS NULL / IS NOT NULL

```
SELECT  
  CASE WHEN job = 'MANAGER' THEN '经理'  
        WHEN job = 'CLERK' THEN '职员'  
        WHEN job = 'SALESMAN' THEN '销售人员'  
        ELSE job  
  END  
FROM emp
```

MYSQL表查询--加强

使用WHERE子句：

#查找1992.1.1后入职的员工

SELECT * FROM emp

WHERE TIMEDIFF(hiredate,'1992-1-1') >= 0;

SELECT * FROM emp

WHERE hiredate > '1992-1-1';

使用LIKE操作符：

%: 表示0到多个任意字符。 _: 表示单个任意字符。

```
#显示首字母为S的员工工资和姓名
SELECT * FROM emp
    WHERE ename like 'S%';
#显示第三个字符为大写O的员工工资和姓名
SELECT * FROM emp
    WHERE ename like '__O%';
#显示没有上级的雇员的情况
SELECT * FROM emp
    WHERE mgr IS NULL;
```

查询表结构：

```
#查询表结构
DESC emp;
```

使用ORDER BY子句

```
#按照工资从低到高的顺序，显示雇员的信息。
SELECT * FROM emp
    ORDER BY sal ASC;
#按照部门号升序而雇员的工资降序排序，显示雇员信息
SELECT * FROM emp
    ORDER BY deptno ASC, sal DESC;
```

分页查询：

```
SELECT ..... LIMIT START,ROWS;
```

从START + 1 行开始取，取出ROWS行， START从0开始计算。

```
SELECT * FROM emp
    ORDER BY empno
    limit 0,3;

SELECT * FROM emp
    ORDER BY empno
    LIMIT 每页显示记录数 * (第几页 - 1), 每页显示记录数。
```

使用分组函数和分组子句 GROUP BY

例：

```

#显示每种岗位的雇员总数、平均工资
SELECT job,COUNT(*),AVG(sal) FROM emp
    GROUP BY job
#显示雇员总数, 以及获得补助的雇员数
#COUNT(列), 如果该列的值为NULL, 就不会统计
SELECT COUNT(*),COUNT(comm) FROM emp
SELECT COUNT(*),COUNT(IF(comm IS NULL,NULL,1)) FROM emp
#显示管理者的总人数
SELECT COUNT(DISTINCT mgr) FROM emp
#显示雇员工资的最大差额
SELECT MAX(sal) - MIN(sal) FROM emp

```

数据分析的总结:

如果SELECT语句同时含有GROUP BY,HAVING,LIMIT,ORDER BY 那么他们的顺序是GROUP BY,HAVING,ORDER BY,LIMIT.

```

SELECT column1,column2,column3...FROM table
    GROUP BY column
    HAVING condition
    ORDER BY column
    LIMIT start,rows;

```

例:

```

#统计各个部门的平均工资, 并且是大于1000的, 并且按照平均工资从高到低排序, 取出前两行记录。
SELECT AVG(sal) as avgSal FROM emp
    GROUP BY deptno
    HAVING avgSal > 1000
    ORDER BY avgSal DESC
    LIMIT 0,2

```

MYSQL多表查询

多表查询是指基于两个和两个以上的表的查询。

在默认情况下, 当两个表查询时, 规则:

1. 从第一张表中, 取出一行和第二张表的每一行进行组合, 返回结果[含有两张表的所有列]。
2. 一共返回的记录数 第一张表行数*第二张表行数
3. 这样多表查询默认处理返回的结果, 称为笛卡尔积。
4. 解决多表的关键就是要写出正确的过滤条件。

多表查询的条件不能少于 表的个数-1, 否则会出现笛卡尔积。

自连接

自连接是指在同一张表上的连接查询。

自连接的特点:

1. 把同一张表当作两张表使用。
2. 需要给表起别名 表名 表别名。
3. 列名不明确, 可以指定列的别名。

例:

```
#显示公司员工和他的上级的名字
SELECT emp1.ename,emp2.ename FROM emp emp1,emp emp2
WHERE emp1.mgr = emp2.empno
```

MYSQL表子查询

1. 子查询是指嵌入在其他SQL语句中的SELECT语句，也叫嵌套查询。
2. 单行子查询是指只返回一行数据的子查询语句。
3. 多行子查询指返回多行数据的子查询 使用关键字 IN

```
#不等号 != 或者 <>
SELECT * FROM emp
WHERE job IN (
    SELECT job FROM emp
    WHERE `deptno` = 10
) AND deptno <> 10
```

子查询当作临时表来使用

```
#查询ecshop各个类别中 价格最高的商品

#先得到 各个类别中，价格最高的商品 当作临时表

SELECT * FROM (
    SELECT cat_id,MAX(shop_price) AS max_price
    FROM ecs_goods
    GROUP BY cat_id) temp,ecs_goods
    WHERE temp.cat_id = ecs_goods.cat_id AND temp.max_price =
    ecs_goods.shop_price
```

在多行子查询中使用ALL操作 (可用MAX代替)

```
#显示工资比部门30的所有员工的工资高的员工的姓名、工资和部门号

SELECT ename,sal,deptno FROM emp
WHERE sal > ALL(
    SELECT sal FROM emp
    WHERE deptno = 30
)
```

在多行子查询中使用ANY操作 (可用MIN代替)

```
#显示工资比部门30的其中一个员工的工资高的员工的姓名，工资和部门号

SELECT ename,sal,deptno FROM emp
WHERE sal > any(
    SELECT sal FROM emp
    WHERE deptno = 30
)
```

多列子查询

多列子查询即是查询返回多个列数据的子查询语句

```
#查询与smith的部门和岗位完全相同的所有雇员(并且不含smith本人)
SELECT ename,job,deptno FROM emp
WHERE (job,deptno) = (
    SELECT job,deptno FROM emp
    WHERE ename = 'SMITH'
) AND ename != 'SMITH'
```

子查询练习

```
#查询每个部门工资高于本部门平均工资的人的资料

SELECT ename,sal,emp.deptno FROM emp,(
    SELECT deptno,AVG(sal) AS avg_sal FROM emp
    GROUP BY deptno
) temp
WHERE emp.deptno = temp.deptno AND emp.sal > temp.avg_sal
```

```
#查询每个部门的信息(部门名, 编号, 地址)和人员数量

#tmp.* 表示把一个表的所有内容都显示出来
SELECT dname,dept.deptno,loc,count FROM(
    SELECT deptno,COUNT(*) AS count FROM emp
    GROUP BY emp.deptno) temp,
dept
WHERE temp.deptno = dept.deptno
```

表复制

自我复制数据(蠕虫复制)

有时,为了对某个SQL语句进行效率测试,我们需要海量数据时,可以使用此法为表创建海量数据。

```
-- 表的复制
CREATE TABLE my_tab01(
    id INT,
    `name` VARCHAR(32),
    sal DOUBLE,
    job VARCHAR(32),
    deptno INT
)

SELECT * FROM my_tab01

-- 演示如何自我复制
-- 1.先把 emp 表的记录复制到 my_tab01

INSERT INTO my_tab01 (id, `name`, sal, job, deptno)
    SELECT empno,ename,sal,job,deptno FROM emp

-- 2.自我复制
```

```
INSERT INTO my_tab01 SELECT * FROM my_tab01

-- 3.删除一张表的重复记录
CREATE TABLE my_tab02 LIKE my_tab01; #表示创建一个和my_tab01结构相同的表
INSERT INTO my_tab02 SELECT * FROM my_tab01
CREATE TABLE temp LIKE my_tab02;

INSERT INTO temp SELECT DISTINCT * FROM my_tab02;
DELETE FROM my_tab02;
INSERT INTO my_tab02 SELECT * FROM temp;
```

合并查询

有时在实际应用中，为了合并多个select语句的结果，可以使用集合操作符号UNION,UNION ALL

1.UNION ALL

该操作符可以用于取得两个结果集的并集。当使用该操作符时，不会取消重复行。

2.UNION

将两个查询结果合并，自动去掉结果集中重复行

```
-- 合并查询

SELECT ename,sal,job FROM emp WHERE sal > 2500

SELECT ename,sal,job FROM emp WHERE job = 'MANAGER'

-- UNION ALL 就是将两个查询结果合并，不会去重
SELECT ename,sal,job FROM emp WHERE sal > 2500
UNION ALL
SELECT ename,sal,job FROM emp WHERE job = 'MANAGER'

-- UNION 将两个查询结果合并，自动去掉结果集中重复行
SELECT ename,sal,job FROM emp WHERE sal > 2500
UNION
SELECT ename,sal,job FROM emp WHERE job = 'MANAGER'
```

MYSQL表外连接

问题：前面的查询是利用where子句对两张表或者多张表，形成的笛卡尔积进行筛选，依据**关联条件**，显示所有匹配的记录，匹配不上的，不显示。

例：列出部门名称和这些部门的员工名称和工作，同时要求显示出那些没有员工的部门。

```
SELECT dname,ename,job FROM dept
LEFT JOIN emp ON emp.deptno = dept.deptno
```

外连接：

1. 左外连接（如果左侧的表完全显示我们就是左外连接）
2. 右外连接（如果右侧的表完全显示我们就是右外连接）

在实际的开发中，我们绝大多数使用的还是**内连接**。

MYSQL约束

约束用于确保数据库的数据满足特定的商业规则。

在MYSQL中，约束包括：NOT NULL、UNIQUE、PRIMARY KEY、FOREIGN KEY 和 CHECK 五种。

PRIMARY KEY 的基本使用

字段名 字段类型 PRIMARY KEY

```
CREATE TABLE t18(
    id INT PRIMARY KEY, -- 表示ID列为主键
    `name` VARCHAR(32),
    email VARCHAR(32)
);
-- 主键列的值不能重复
-- 复合主键
CREATE TABLE t20(
    id INT,
    `name` VARCHAR(32),
    email VARCHAR(32),
    PRIMARY KEY(id,email)
);
```

用于唯一标识表行的数据，当定义主键约束后，该列不能重复。

主键使用的细节：

- 1.PRIMARY KEY不能重复而且不能为NULL。
- 2.一张表最多只能有一个主键，但可以是复合主键。
- 3.主键的指定方式有两种：
 - (1)直接在字段名后定义：字段名 PRIMARY KEY
 - (2)在表定义最后写 PRIMARY KEY(列名)
- 4.使用DESC表名，可以看到PRIMARY KEY的情况。

在实际开发中，每个表往往都会设计一个主键。

NOT NULL 和 UNIQUE

NOT NULL (非空)

如果在列上定义了NOT NULL，那么当插入数据时，必须为该列提供数据。

字段名 字段类型 NOT NULL

UNIQUE (唯一)

当定义了唯一约束后，该列值是不能重复的。

字段名 字段类型 UNIQUE

UNIQUE细节

1.如果没有指定NOT NULL，则UNIQUE字段可以有多个NULL。

2.一张表可以有多个UNIQUE字段。

FORRIGN KEY (外键)

用于定义主表和从表之间的关系：外键约束要定义在从表上，主表则必须要有主键约束或是UNIQUE约束，当定义外键约束后，要求外键列数据必须在主表的主键列存在或者为NULL。

FOREIGN KEY (本表字段名) REFERENCES 主表名 (主键名或UNIQUE字段名)



```
CREATE TABLE my_class(
    id INT PRIMARY KEY,
    `name` VARCHAR(32) NOT NULL DEFAULT ''
)

CREATE TABLE my_stu(
    id INT PRIMARY KEY,
    `name` VARCHAR(32),
    class_id INT,
    FOREIGN KEY (class_id) REFERENCES my_class(id)
)
```

FOREIGN KEY (外键) 的细节说明

1. 外键指向的表的字段，要求是PRIMARY KEY或者是UNIQUE。
2. 表的类型是innodb，这样的表才支持外键。
3. 外键字段的类型必须要和主键字段的类型一致（长度可以不同）。
4. 外键字段的值，必须在主键字段中出现过，或者为NULL[前提是外键字段允许为NULL]
5. 一旦建立主外键的关系，数据就不能随意删除了。

CHECK

用于强制行数据必须满足的条件，假定在sal列上定义了check约束，并要求sal列值在1000~2000之间。如果不在1000~2000之间就会提示出错。

oracle和sql server 均支持check，但是mysql5.7目前还不支持check，只做语法校验，但不会生效。

```
CREATE TABLE t23(
    id INT PRIMARY KEY,
    `name` VARCHAR(32),
    sex VARCHAR(6) CHECK (sex IN ('man', 'woman')),
    sal DOUBLE CHECK (sal > 1000 AND sal < 2000)
);
```

```
CREATE TABLE goods(
    goods_id INT PRIMARY KEY,
    goods_name VARCHAR(32),
    unitprice DOUBLE CHECK (unitprice > 1 AND unitprice < 9999.99),
    category VARCHAR(32),
    provider VARCHAR(32)
);

CREATE TABLE customer(
    customer_id INT PRIMARY KEY,
    `name` VARCHAR(32) NOT NULL,
    address VARCHAR(32) UNIQUE,
    #sex ENUM('男', '女'),
    sex VARCHAR(3) CHECK (sex IN ('男', '女')),
    card_id VARCHAR(20)
);

CREATE TABLE purchase(
    order_id INT PRIMARY KEY,
    customer_id INT,
    goods_id INT,
    nums INT,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (goods_id) REFERENCES goods(goods_id)
);
```

自增长

字段名整形 PRIMARY KEY auto_increment

```
CREATE TABLE t24(
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(32) NOT NULL DEFAULT '',
    `name` VARCHAR(32) NOT NULL DEFAULT ''
);

INSERT INTO t24
    VALUES(NULL, '123@qq.com', 'jack')

INSERT INTO t24
```

```
(email, `name`) VALUES('hsp@sohu.com', 'hsp')
```

```
SELECT * FROM t24
DESC t24
```

自增长使用细节

1. 一般来说自增长是和PRIMARY KEY配合使用的。
2. 自增长也可以单独使用[但是需要配合一个UNIQUE]
3. 自增长修饰的字段为整数型的(虽然小数也可以但是非常非常少这样使用)
4. 自增长默认从开始, 你也可以通过如下命令修改: ALTER TABLE 表名 AUTO_INCREMENT = xxx;
5. 如果你添加数据时, 给自增长字段(列)指定的值, 则以指定的值为准。一般来说, 就按照自增长的规则来添加数据。

MySQL索引

提高数据库性能, 索引是最物美价廉的东西了。不用加内存, 不用改程序, 不用调SQL, 查询速度就可能提高百倍千倍。

在没有创建索引前, emp.ibd 文件大小是 524m。

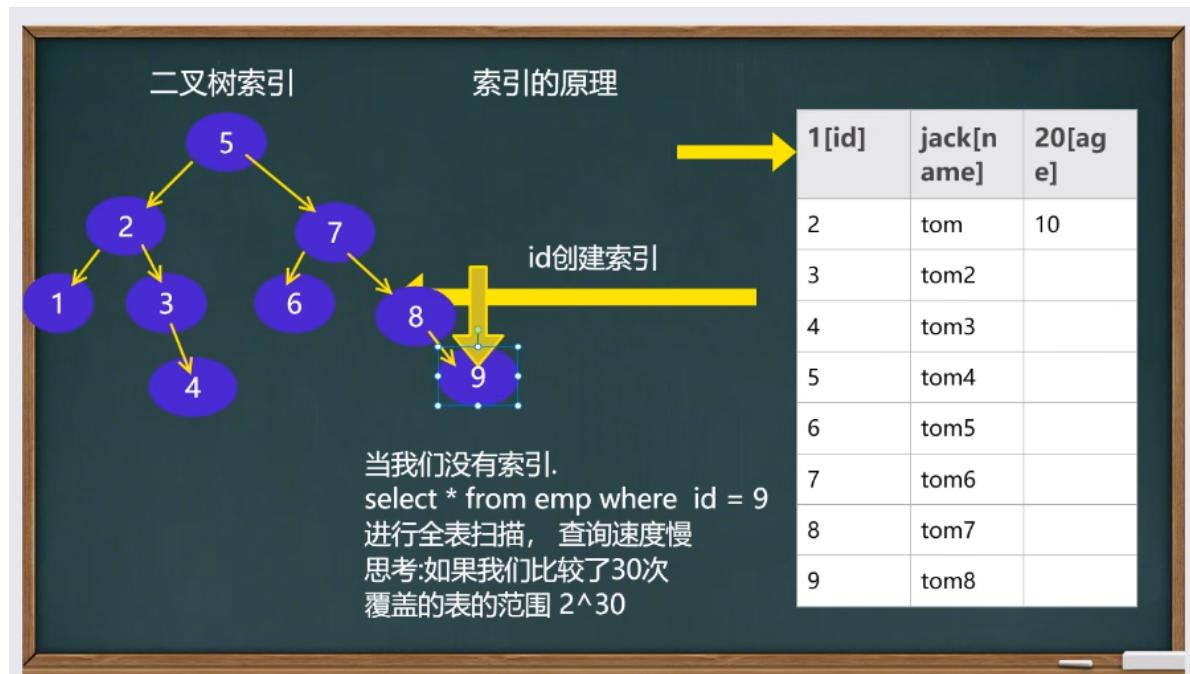
创建索引过后 emp.ibd 文件大小为655m 索引本身也需要占用内存。[索引本身也会占用空间]

```
-- empno_index 索引名称
-- ON emp(empno):表示在 emp 表的 empno列创建索引
CREATE INDEX empno_index ON emp(empno);
```

索引的原理

没有索引为什么会慢? 因为全表扫描。

使用索引为什么会快? 形成一个索引的数据结构, 比如二叉树。



索引的代价:

1、磁盘占用

2.对DML(update delete insert)语句效率的影响。如果对表进行DML操作, 会对索引进行维护。

但是在我们的项目中, select[90%],DML(10%)。 (好像计网的缓存机制hh) 。

索引的类型

1. 主键索引, 主键自动的为主索引 (类型PRIMARY KEY) (主键本身就是一个索引)。
2. 唯一索引 (UNIQUE) 列是唯一的, 同时也是索引。
3. 普通索引 (INDEX)
4. 全文索引 (FULLTEXT) [适用于MyISAM]。一般不使用MYSQL自带的全文索引, 开发中考虑使用: 全文搜索Solar和ElasticSearch(ES)。

索引使用

创建索引

添加索引

```
-- 创建索引
CREATE TABLE t25(
    id INT,
    `name` VARCHAR(32)
);

-- 查询表是否有索引
SHOW INDEXES FROM t25;

-- 添加索引
-- 添加唯一索引
CREATE UNIQUE INDEX id_index ON t25(id);
-- 添加普通索引方式1
CREATE INDEX id_index ON t25(id);
-- 如果某列的值是不会重复的, 则优先考虑UNIQUE索引, 否则使用普通索引。
-- 添加普通索引方式2
ALTER TABLE t25 ADD INDEX id_index(id);
-- 添加主键索引
ALTER TABLE t25 ADD PRIMARY KEY (id);
```

删除索引

```
-- 删除索引
DROP INDEX id_index ON t25;
-- 删除主键索引
ALTER TABLE t25 DROP PRIMARY KEY;
```

修改索引: 即先删除再添加新的索引。

查询索引:

```
SHOW INDEX FROM t25;
SHOW INDEXES FROM t25;
SHOW KEYS FROM t25;
DESC t25; #查出来的信息没有前三种详细
```

创造索引规则

1. 较频繁的作为查询条件字段应该创建索引。

```
SELECT * FROM emp WHERE empno = 1;
```

2. 唯一性太差的字段不适合单独创建索引，即使频繁作为查询条件。

```
SELECT * FROM emp WHERE sex = '男';
```

3. 更新非常频繁的字段不适合创建索引。

```
SELECT * FROM emp WHERE logincount = 1;
```

4. 不会出现在WHERE子句中的字段不该创建索引。

事务

什么是事务：

事物用于保证数据的一致性，它由**一组相关的DML语句构成**，该组的DML语句要么全部成功，要么全部失败。如：转账就要用事物来处理，用于保证数据的一致性。

事物和锁：

当执行事物操作时（dml语句），mysql会在表上加锁，防止其他用户改表的数据，这对用户来说是非常重要的。

• mysql 数据库控制台事务的几个重要操作(**基本操作 transaction.sql**)

1. start transaction -- **开始一个事务**
2. savepoint 保存点名 -- **设置保存点**
3. rollback to 保存点名 -- **回退事务**
4. rollback -- **回退全部事务**
5. commit -- **提交事务, 所有的操作生效, 不能回退**

- 细节：
1. 没有设置保存点
 2. 多个保存点
 3. 存储引擎
 4. 开始事务方式

• 回退事务

在介绍回退事务前，先介绍一下保存点(savepoint).保存点是事务中的点.用于取消部分事务，当结束事务时（commit），会自动的删除该事务所定义的所有保存点.当执行回退事务时，通过指定保存点可以回退到指定的点,这里我们作图说明

• 提交事务

使用commit语句可以提交事务.当执行了commit语句后,会确认事务的变化、结束事务、删除保存点、释放锁，数据生效。当使用commit语句结束事务后，其它会话[其他连接]将可以查看到事务变化后的新数据[所有数据就正式生效.]

```
-- 1. 创建一张测试表
CREATE TABLE t27(
    id INT,
    `name` VARCHAR(32)
);
-- 2. 开始事务
START TRANSACTION;
-- 3. 设置保存点
```

```
SAVEPOINT a;
-- 执行DML语句
INSERT INTO t27 VALUES(100, 'tom');
SELECT * FROM t27;

SAVEPOINT b;
-- 执行DML操作
INSERT INTO t27 VALUES(200, 'jack');

-- 回退到b
ROLLBACK TO b;
-- 继续回退到a
ROLLBACK a
-- 直接回退到事务开始时的状态
ROLLBACK
-- 提交事务
COMMIT
```

事务细节：

1. 如果不开始事务，默认情况下，dml语句是自动提交的，不能回滚。
2. 如果开始一个事务，你没有创建保存点。你可以执行ROLLBACK，默认就是回到事务开始的状态。
3. 可以在事务中（还没有提交时），创建多个保存点。
4. 可以在事务提交前，选择回退到哪个保存点。
5. mysql的事务机制需要innodb的存储引擎，myisam不支持。
6. 开始一个事务**start transaction**，或者写**set autocommit = off**。

事务隔离级别

1.多个连接开启各自事务操作数据库中数据时，数据库系统要负责隔离操作，以保证各个链接在获取数据是的准确性。

2.如果不考虑隔离性，可能会引发如下问题；

- 脏读
- 不可重复读
- 幻读

脏读 (dirty read)：当一个事务读取另一个事务尚未提交的修改时，产生脏读。

不可重复读 (nonrepeatable read)：同一查询在同一事物中多次进行，由于其他提交事务所做的修改或删除，每次返回不同的结果集，此时发生不可重复读。

幻读 (phantom read)：同一查询在同一事物中多次进行，由于其他提交事务所做的插入操作，每次返回不同的结果集，此时发生幻读。

概念：Mysql隔离级别定义了事务与事务之间的隔离程度。

Mysql隔离级别	脏读	不可重复读	幻读	加锁读
读未提交 (Read uncommitted)	V	V	V	不加锁
读已提交 (Read committed)	X	V	V	不加锁
可重复读 (Repeatable read)	X	X	X	不加锁
可串行化 (Serializable) [演示 重开客户端]	X	X	X	加锁

说明: V 可能出现 X 不会出现

```
-- 通过两个控制台演示MYSQL的事务隔离级别
# 隔离级别是和事务相关的 一定要启动事务
-- 1. 打开两个MYSQL的控制台

-- 2. 查看当前MYSQL的隔离级别是什么
SELECT @@tx_isolation;

-- mysql> SELECT @@tx_isolation;
-- +-----+
-- | @@tx_isolation |
-- +-----+
-- | REPEATABLE-READ |
-- +-----+


-- 3. 把其中一个控制台的隔离级别设置 READ UNCOMMITTED
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

-- 4. 创建表
USE tmp;
CREATE TABLE `account`(
    id INT,
    `name` VARCHAR(32),
    money INT
);
```

事务隔离语句

1. 查看当前会话隔离级别

```
SELECT @@tx_isolation;
```

2. 查看系统当前隔离级别

```
SELECT @@global.tx_isolation;
```

3. 设置当前会话隔离级别

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

4. 设置系统当前隔离级别

```
SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

5. mysql默认的事务隔离级别是repeatable read，一般情况下，没有特殊要求，没有必要修改（因为该级别可以满足绝大部分项目要求）。

全局修改，修改mysql.ini配置文件，在最后加上

#可选参数有：READ-UNCOMMITTED,READ-COMMITTED,REPEATABLE-READ,SERIALIZABLE.

```
[mysqld]transaction-isolation = REPEATABLE-READ
```

MySQL事务ACID

事务的ACID特性：

1.原子性 (Atomicity)

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

2.一致性 (Consistency)

事务必须使数据库从一个一致性状态变换到另一个一致性状态。

3.隔离性 (Isolation)

事务的隔离性是多个事务并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事物之间要相互隔离。

4.持久性 (Durability)

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不该对其有任何影响。

MySQL表类型和存储引擎

基本介绍：

1. MySQL的表类型由存储引擎 (Storage Engines) 决定，主要包括MyISMA、InnoDB、Memory等。
2. MySQL数据表主要支持六种类型，分别是：CSV、Memory、ARCHIVE、MRG_MYISAM、MYISAM、InnoDB。
3. 这六种又分为两类，一类是“事务安全型” (transaction-safe)，比如：InnoDB；其余都属于第二类，称为“非事务安全型” (non-transaction-safe) [myisam和memory]。

```
-- 查看所有的存储引擎
```

```
SHOW ENGINES;
```

主要的存储引擎/表类型特点

特点	MyISAM	InnoDB	Memory	Archive
批量插入的速度	高	低	高	非常高
事务安全		支持		
全文索引	支持		表锁	行锁
锁机制	表锁	行锁	有	没有
存储限制	没有	64TB	支持	
B树索引	支持	支持	支持	
哈希索引		支持	支持	
集群索引		支持	支持	
数据缓存		支持	支持	
索引缓存	支持	支持	支持	
数据可压缩	支持			支持
空间使用	低	高	N/A	非常低
内存使用	低	高	中等	低
支持外键		支持		

存储引擎细节：

● 细节说明

我这里重点给大家介绍三种：MyISAM、InnoDB、MEMORY

1. MyISAM不支持事务、也不支持外键，但其访问速度快，对事务完整性没有要求
2. 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。但是比起MyISAM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。
3. MEMORY存储引擎使用存在内存中的内容来创建表。每个MEMORY表只实际对应一个磁盘文件。MEMORY类型的表访问非常得快，因为它的数据是放在内存中的，并且默认使用HASH索引。但是一旦服务关闭，表中的数据就会丢失掉，表的结构还在。

如何选择表的存储引擎：

1. 如果你的应用不需要事务，处理的只是基本的CRUD操作，那么MyISAM是不二选择，速度快
2. 如果需要支持事务，选择InnoDB。
3. Memory 存储引擎就是将数据存储在内存中，由于没有磁盘I/O的等待，速度极快。但由于是内存存储引擎，所做的任何修改在服务器重启后都将消失。（经典用法 **用户的在线状态()**）

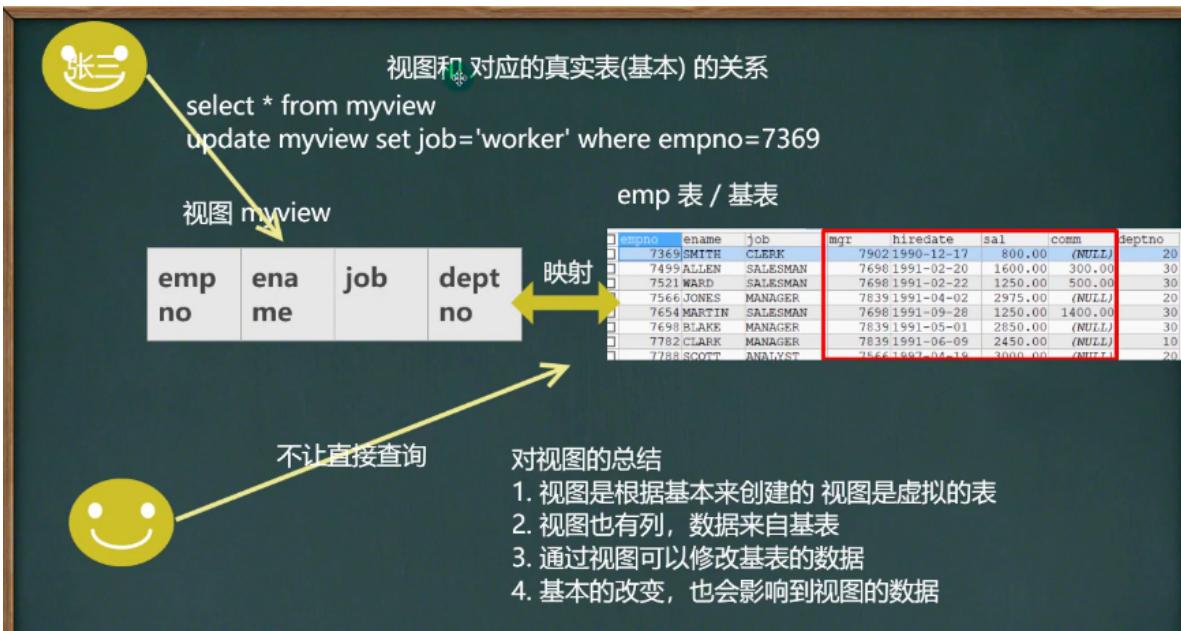
修改存储引擎：

```
ALTER TABLE `表名` ENGINE = 存储引擎;
```

视图

基本概念：

1. 视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含列，其数据来自对应的真实表（基表）。
2. 视图和基表的示意图。



视图的基本使用：

```
-- 视图的使用

-- 创建视图
CREATE VIEW emp_view01
AS
SELECT empno,ename,job,deptno FROM emp;

-- 查看视图
DESC emp_view01;
SELECT * FROM emp_view01;

-- 修改视图
ALTER VIEW emp_view01
AS
SELECT empno,ename,job,deptno FROM emp;

-- 查看创建视图的指令
SHOW CREATE VIEW emp_view01;

-- 删除视图
DROP VIEW emp_view01;
```

视图细节：

1. 创建视图后, 到数据库去看, 对应视图只有一个视图结构文件, 并没有数据文件 (形式: 视图名.frm)
2. 视图的数据变化会影响到基表, 基表的数据变化也会影响到视图。
3. 视图中可以再使用视图, 数据仍然来自基表。
4. 视图可以有多张基表。

视图实践：

1. **安全**。一些数据表有着重要的信息。有些字段是保密的，不能让用户直接看到。这时就可以创建一个视图，在这张视图中只保留一部分字段。这样，用户就可以查询自己需要的字段，不能查看保密的字段。
2. **性能**。关系数据库的数据常常会分表存储，使用外键建立这些表的之间关系。这时，数据库查询通常会用到连接（**JOIN**）。这样做不但麻烦，效率相对也比较低。如果建立一个视图，将相关的表和字段组合在一起，就可以避免使用**JOIN**查询数据。
3. **灵活**。如果系统中有一张旧的表，这张表由于设计的问题，即将被废弃。然而，很多应用都是基于这张表，不易修改。这时就可以建立一张视图，视图中的数据直接映射到**新建**的表。这样，就可以少做很多改动，也达到了升级数据表的目的。

MySQL管理

MySQL用户

mysql中的用户，都储存在系统数据库mysql中user表中

其中user表的重要字段说明：

1. host:允许登录的“位置”，localhost表示该用户只允许本机登录，也可以指定IP地址，比如：
192.168.1.100。
2. user: 用户名。
3. authentication_string:密码，是通过mysql的password()函数加密之后得到的密码。

```
-- 原因：当我们做项目开发时，可以根据不同的开发人员，赋给他相应的MySQL权限  
-- 所以MySQL数据库管理人员(root)，根据需要创建不同的用户，赋给相应的权限，供人员使用
```

```
-- 创建用户  
-- (1) 'diduo'@'localhost' 表示用户的完整信息 'diduo'用户名 'localhost'登录地点  
-- (2) 123456是密码，但是存放到mysql.user表时，是password('123456')加密后的密码。  
-- (3) 不能添加的话，flush privileges; 刷新一下
```

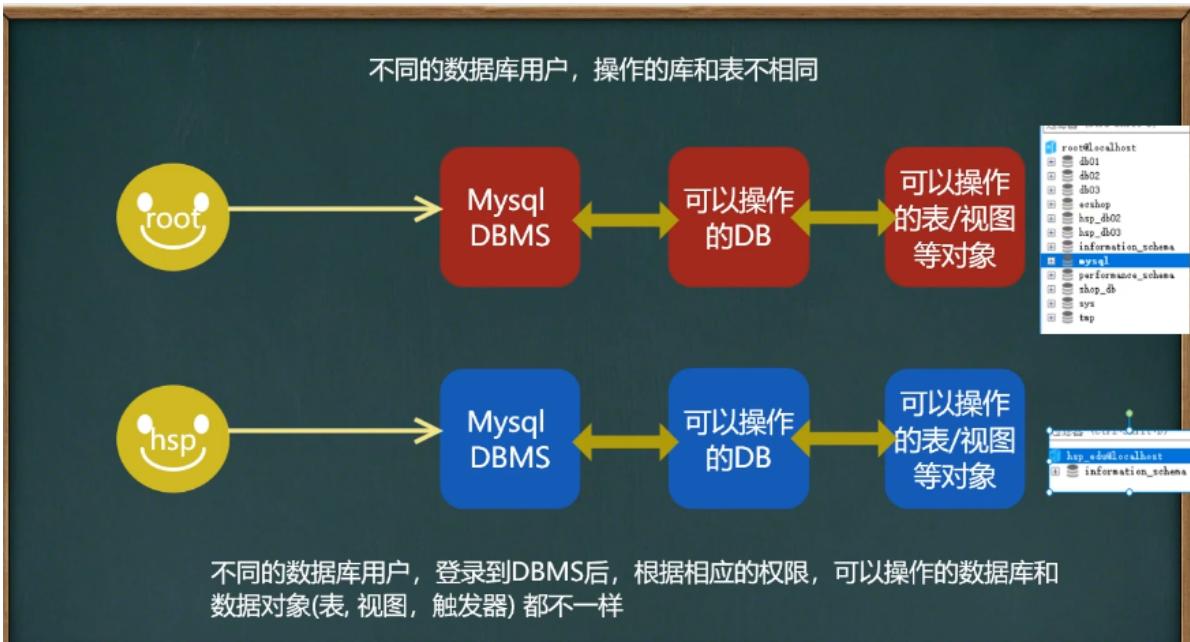
```
CREATE USER 'diduo'@'localhost' IDENTIFIED BY '123456';  
SELECT * FROM mysql.user;
```

```
-- 删除用户  
DROP USER 'diduo'@'localhost';
```

```
-- 登录
```

```
-- 修改密码  
SET PASSWORD = PASSWORD('abcdef');  
-- 修改别人的密码需要权限  
SET PASSWORD FOR 'diduo'@'localhost' = PASSWORD('123456');
```

不同的数据库用户，操作的库和表不相同



MYSQL权限

权限	意义
ALL [PRIVILEGES]	设置除GRANT OPTION之外的所有简单权限
ALTER	允许使用ALTER TABLE
ALTER ROUTINE	更改或取消已存储的子程序
CREATE	允许使用CREATE TABLE
CREATE ROUTINE	创建已存储的子程序
CREATE TEMPORARY TABLES	允许使用CREATE TEMPORARY TABLE
CREATE USER	允许使用CREATE USER, DROP USER, RENAME USER和REVOKE ALL PRIVILEGES。
CREATE VIEW	允许使用CREATE VIEW
DELETE	允许使用DELETE
DROP	允许使用DROP TABLE
EXECUTE	允许用户运行已存储的子程序
FILE	允许使用SELECT... INTO OUTFILE和LOAD DATA INFILE
INDEX	允许使用CREATE INDEX和DROP INDEX
INSERT	允许使用INSERT
LOCK TABLES	允许对您拥有SELECT权限的表使用LOCK TABLES
PROCESS	允许使用SHOW FULL PROCESSLIST
REFERENCES	未被实施
RELOAD	允许使用FLUSH
REPLICATION CLIENT	允许用户询问从属服务器或主服务器的地址
REPLICATION SLAVE	用于复制型从属服务器（从主服务器中读取二进制日志事件）
SELECT	允许使用SELECT
SHOW DATABASES	SHOW DATABASES显示所有数据库
SHOW VIEW	允许使用SHOW CREATE VIEW
SHUTDOWN	允许使用mysqladmin shutdown
SUPER	允许使用CHANGE MASTER, KILL, PURGE MASTER LOGS和SET GLOBAL语句, mysqladmin debug命令; 允许您连接（一次），即使已达到max_connections。
UPDATE	允许使用UPDATE
USAGE	“无权限”的同义词
GRANT OPTION	允许授予权限

给用户授权

● 给用户授权

基本语法:

grant 权限列表 on 库.对象名 to '用户名' @' 登录位置' 【identified by '密码'】

说明:

1, 权限列表, 多个权限用逗号分开

grant select on

grant select, delete, create on

grant all [privileges] on //表示赋予该用户在该对象上的所有权限

2. 特别说明

. : 代表本系统中的所有数据库的所有对象(表, 视图, 存储过程)

库.* : 表示某个数据库中的所有数据对象(表, 视图, 存储过程等)

3, identified by可以省略, 也可以写出.

(1)如果用户存在, 就是修改该用户的密码。

(2)如果该用户不存在, 就是创建该用户!

结束语

● 回收用户授权

基本语法:

revoke 权限列表 on 库.对象名 from '用户名'@"登录位置";

● 权限生效指令

如果权限没有生效, 可以执行下面命令.

基本语法:

FLUSH PRIVILEGES;

用户细节:

1. 在创建用户的时候, 如果不指定Host, 则为%, %表示所有IP都有连接权限
create user xxx;

2. 你也可以这样指定

create user 'xxx'@'192.168.1.%' 表示 xxx用户在 192.168.1.*的ip可以登录mysql

3. 在删除用户的时候, 如果 host 不是 %, 需要明确指定 '用户名'@'host值'