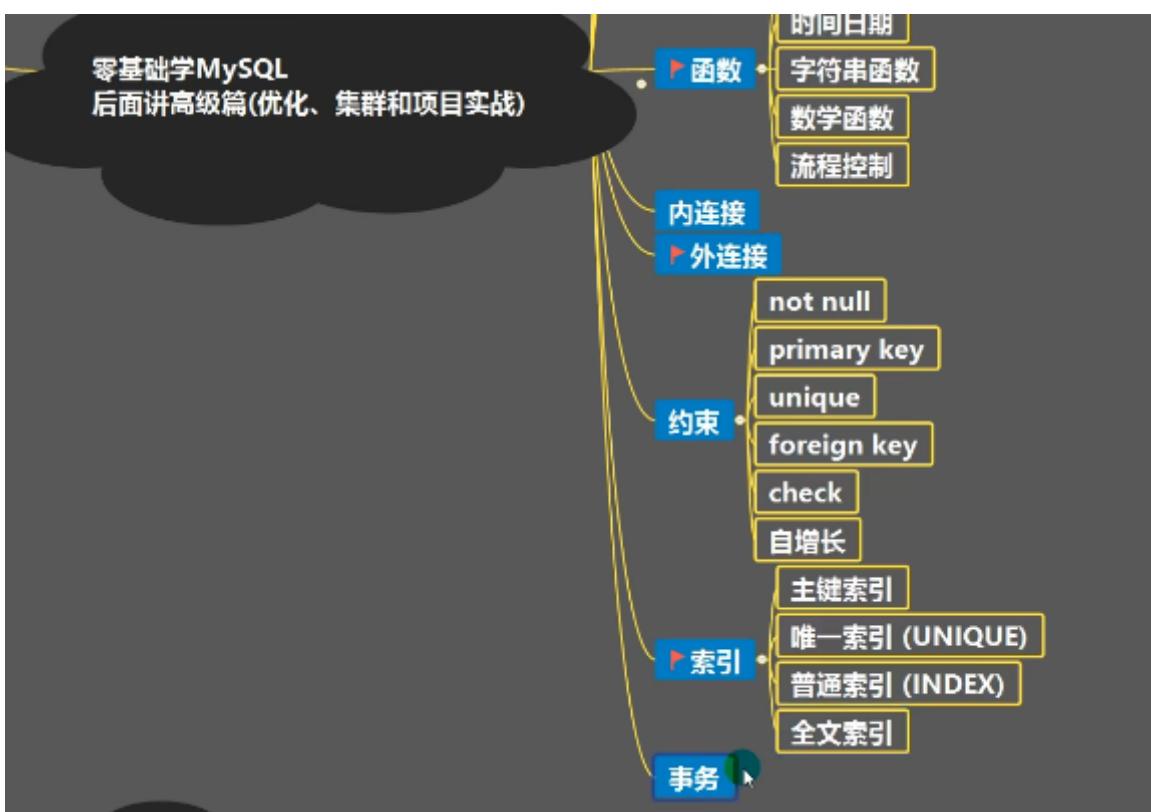
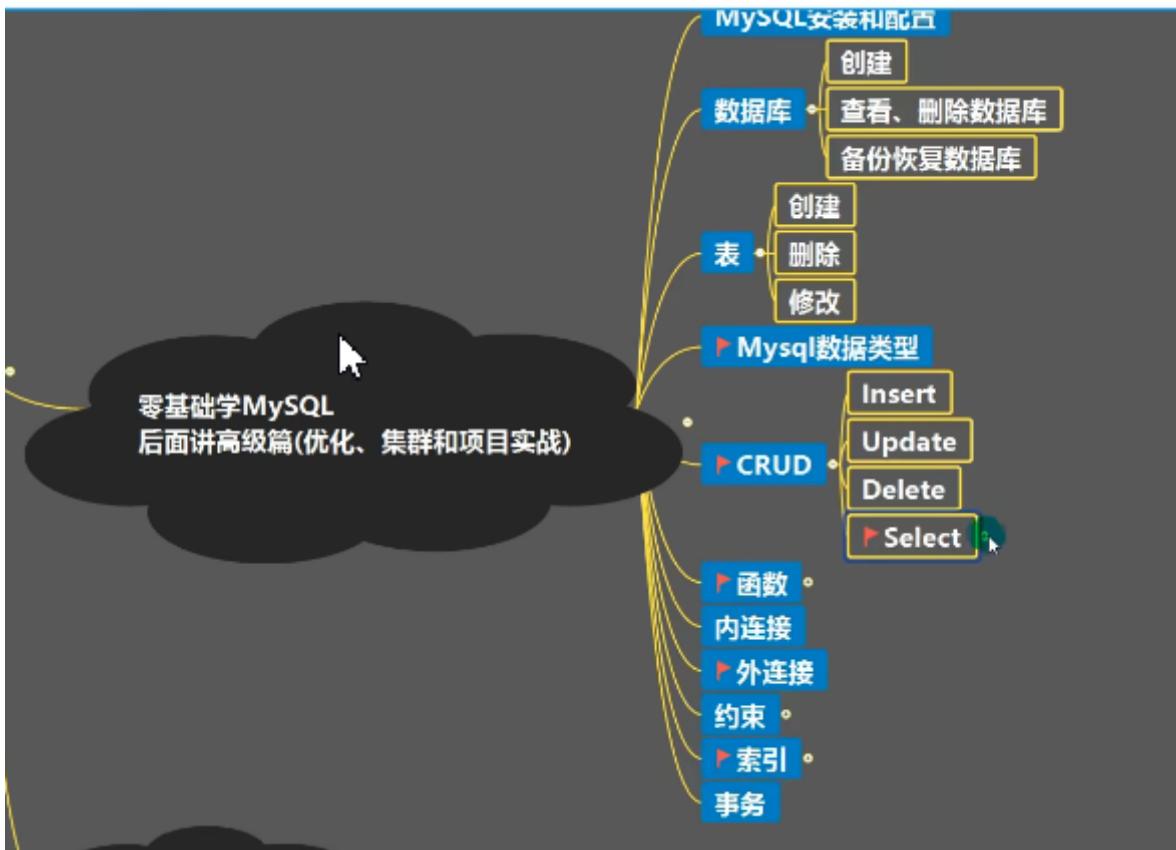
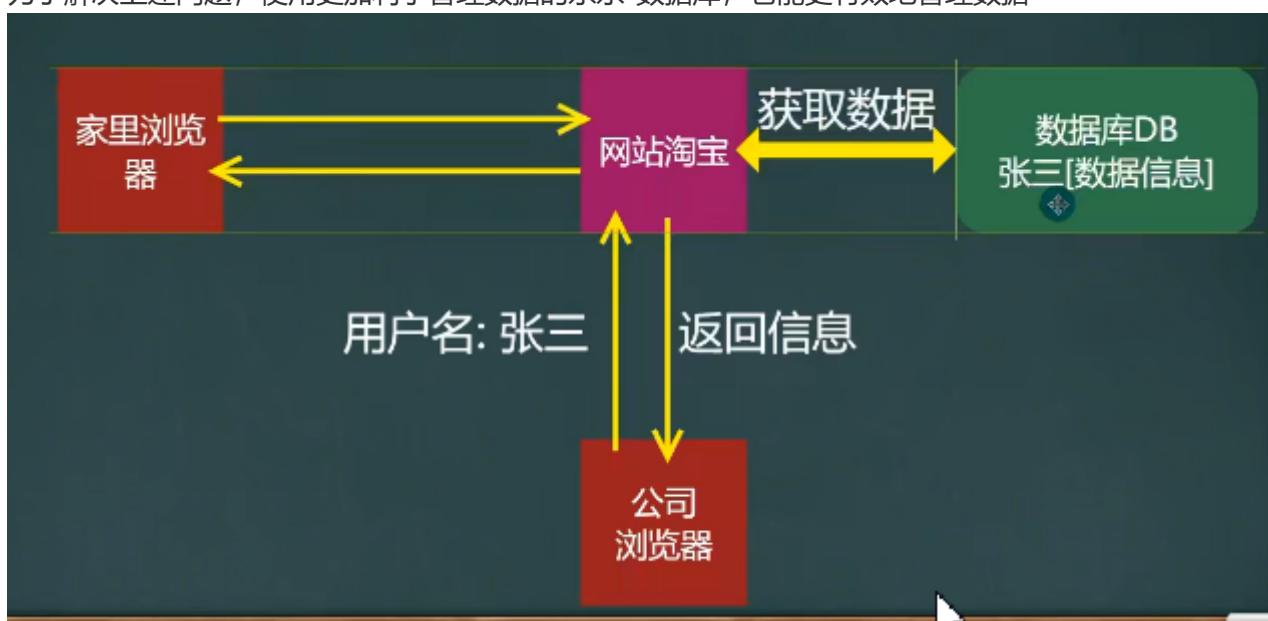


# chapter24-MySQL



引出

- 一个问题：淘宝网、京东、微信、抖音都有各自的功能，那么当我们退出系统的时候，下次再访问时，为什么信息还存在？
- 解决之道-文件、数据库  
为了解决上述问题，使用更加利于管理数据的东东-数据库，它能更有效地管理数据



## MySQL数据库的安装和配置

### 安装和配置

1. MySQL5.7.19解压版下载地址：  
<https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.19-winx64.zip>
2. 解压缩，不要带中文以及空格路径
3. 在解压目录下新建 my.ini

```
[client]
port=3306
default-character-set=utf8

[mysqld]
# 设置为自己MySQL的安装目录
basedir=C:\Users\Zhangyu\Documents\mysql-5.7.19-winx64\
# 设置为MySQL的数据目录，这个目录是系统创建
datadir=C:\Users\Zhangyu\Documents\mysql-5.7.19-winx64\data\
port=3306
character_set_server=utf8
# 跳过安全检查，后面设置完密码之后注释掉
skip-grant-tables
```

3. 添加 bin 文件夹到环境变量中 Path 中
4. 使用管理员权限打开 cmd
5. 使用 cd 指令进入到 bin 目录中
6. 安装数据库：mysqld install

7. 初始化数据库: mysqld --initialize-insecure --user=mysql (会在安装目录下生成一个data目录)
8. 启动 mysql 服务: net start mysql
9. 停止 mysql 服务: net stop mysql
10. 进入 mysql 管理终端: mysql -u root -p  
(-u 是用户名, -p 是密码, 这里为空)
11. 修改 root 用户密码:  
use mysql;  
update user set authentication\_string=password('hsp') where user='root' and Host='localhost';  
修改 root 用户密码为 hsp  
flush privileges; 刷新权限  
quit; 退出
12. 修改 my.ini, 再次进入就会进行权限验证了
13. 重新启动 mysql:  
net stop mysql  
net start mysql

## 命令行连接到MySQL

- 连接到 Mysql 服务(Mysql数据库)的指令:  
**mysql -h 主机IP -P 端口 -u 用户名 -p密码**
  - (1) -p密码不要有空格
  - (2) -p后面没有写密码, 回车会要求输入密码
  - (3) 如果没有写-h 主机, 默认就是本机
  - (4) 如果没有写-P 端口, 默认就是3306
  - (5) 在实际工作中, 3306一般会修改

登录前, 保证服务的启动

## Navicat 安装和使用

- 介绍: 图形化MySQL管理软件
- 下载&安装&使用
- 演示: 使用Navicat创建一个数据库 db01, 在db01创建一张表users, 保存3个用户

The screenshot shows the Navicat for MySQL interface. On the left, the database tree shows '本机MySQL' expanded, with 'db01' selected. Under 'db01', there are tables, views, functions, queries, backups, and system databases like 'information\_schema', 'mysql', 'performance\_schema', and 'sys'. On the right, the main area displays the 'users' table with three rows of data:

id	name	address
1	张三	北京海淀区
2	李四	北京昌平区
*	王五	天津XX小区

## SQLyog 安装和使用

- 介绍：图形化MySQL管理软件
- 下载&安装&使用
- 演示：使用Navicat创建一个数据库 db01，在db01创建一张表users，保存3个用户

## 数据库三层架构-破除MySQL神秘

1. 所谓安装 Mysql 数据库，就是在主机安装了一个数据库管理系统（DBMS），这个管理程序可以管理多个数据库。DBMS (database manage system)
2. 一个数据库中可以创建多个表，以保存数据（信息）。
3. 数据库管理系统（DBMS）、数据库和表的关系如图所示：示意图





分析：MySQL中有三层结构：DBMS、DB、表

- DBMS对应 mysqld.exe程序 该程序监听端口3306
- 数据库对应 data 文件夹
- 表对应文件
- MySQL 数据库-普通表的本质仍然是文件
- 操作过程：SELECT \* FROM users 就是将指令发送到3306端口，DBMS 接收后执行

## 数据库的存储结构



表的一行称之为一条记录 -> 在java程序中，一行记录往往使用对象表示

## 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]

# 数据库

## 创建数据库

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] 注意默认是 utf8\_general\_ci) [举例说明 database.sql 文件]

```
# 使用指令创建数据库
CREATE DATABASE hsp_db01;
# 删除数据库
DROP DATABASE hsp_db01;

# 创建一个使用utf8字符集的hsp_db02数据库`hsp_db02`
CREATE DATABASE hsp_db02 CHARACTER SET utf8
# 创建一个使用utf8字符集, 并带校对规则的 hsp_db03
CREATE DATABASE hsp_db03 CHARACTER SET utf8 COLLATE utf8_bin
# 校对规则 utf8_bin 区分大小写 默认 utf8_general_ci 不区分大小写

# 下面`t1`是一条查询的sql, select 查询 * 表示所有字段 FROM 从哪个表
# WHERE 从哪个字段 NAME = 'tom' 查询名字是 tom
SELECT *
FROM t1
WHERE NAME = 'Tom'
```

## 查看、删除数据库

- 显示数据库语句  
SHOW DATABASE
- 显示数据库创建语句  
SHOW CREATE DATABASE db\_name
- 数据库删除语句【慎用】  
DROP DATABASE [IF EXISTS] db\_name

```
#查看当前数据库服务器中的所有数据库
SHOW DATABASES
#查看前面创建的hsp_db01数据库的定义信息
SHOW CREATE DATABASE hsp_db01
#老师说明 在创建数据库, 表的时候, 为了规避关键字, 可以使用反引号解决
CREATE DATABASE `CREATE`
#删除前面创建的hsp_db01数据库
```

```
DROP DATABASE hsp_db02  
#切换数据库  
use database_name
```

## 备份和恢复数据库

- 备份数据库（注意：在DOS执行）命令行

```
mysqldump -u 用户名 -p -B 数据库1 数据库2 数据库n > 文件名.sql
```

- 恢复数据库（注意：进入Mysql命令行再执行）

```
source 文件名.sql
```

DOS就是cmd终端

```
#练习：database03.sql 备份hsp_db02 和 hsp_db03 库中的数据，并恢复
```

```
#备份，要在Dos下执行mysqldump指令其实在mysql安装目录\bin
```

```
#这个备份的文件，就是对应的sql语句
```

```
mysqldump -u root -p -B hsp_db02 hsp_db03 > c:\\ioTest\\bak.sql  
> -B表示备份数据库
```

```
DROP DATABASE hsp_db02
```

```
#恢复数据库（注意：进入Mysql命令行再执行）
```

```
source c:\\ioTest\\bak.sql
```

```
#第二个恢复方法，直接将bak.sql的内容放到查询编辑器中，执行
```

- 备份数据库的表

```
mysqldump -u 用户名 -p密码 数据库 表1 表2 表n > c:\\ioTest\\文件名.sql
```

## 表

### 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:指定列明 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（这个涉及内容较多，后面单独讲解）
```

名称用下划线\_间隔

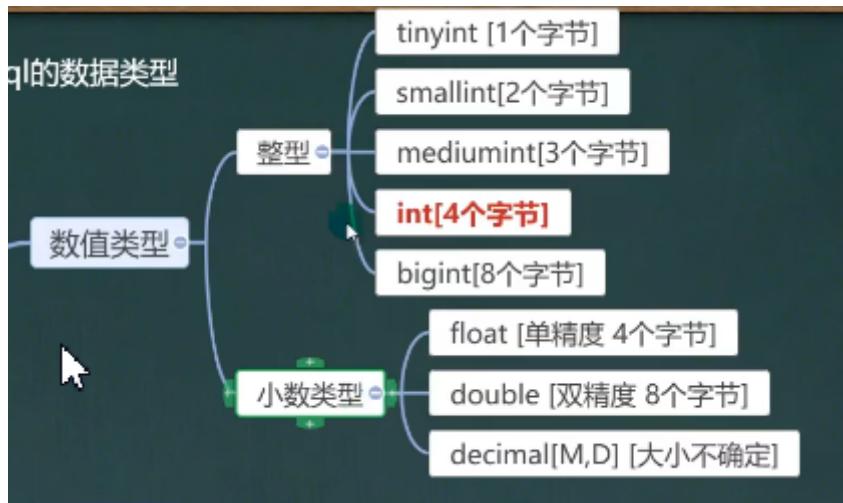
```
#指令创建表
#id 整形
#name 字符串
#password 字符串
#birthday 日期
CREATE TABLE `user` (
    id INT,
    `name` VARCHAR(255),
    `password` VARCHAR(255),
    `birthday` DATE)
CHARACTER SET utf8 COLLATE utf8_bin ENGINE INNODB;
```

- name, password 是 mysql 的关键字，需要用反引号`标出
- 删除表的指令：DROP TABLE table\_name;

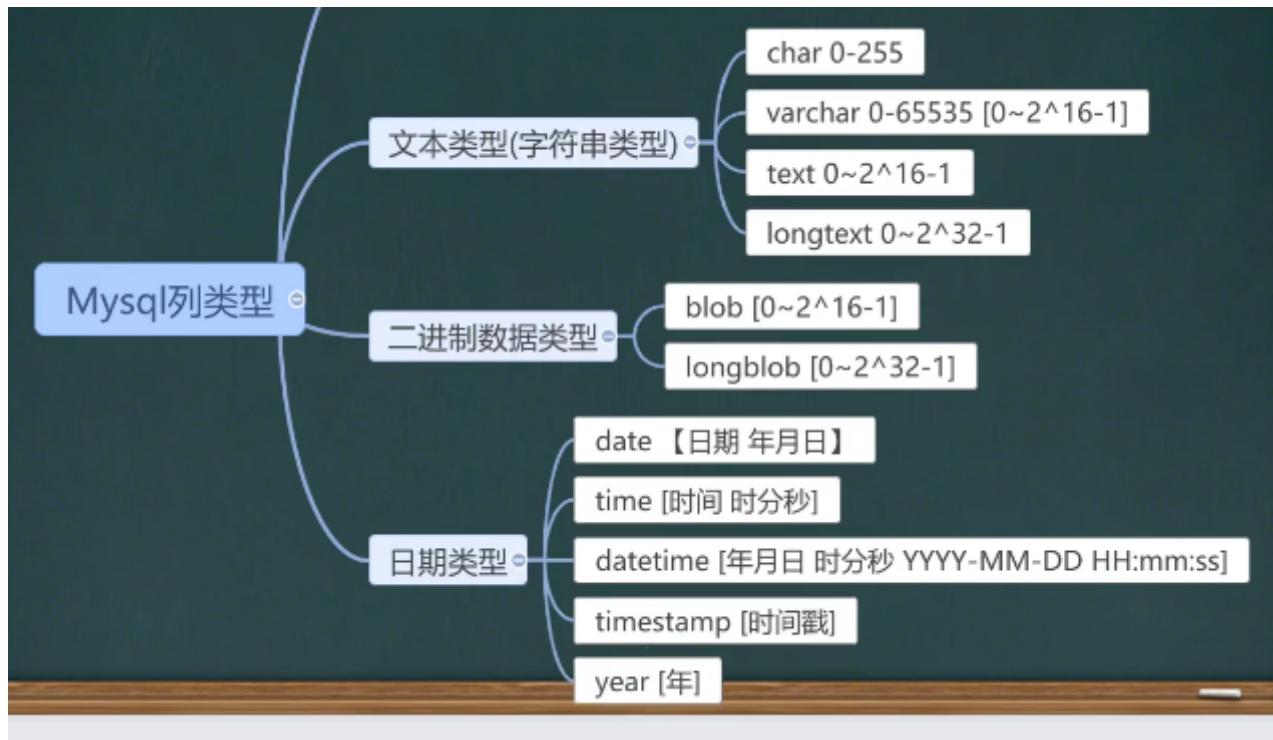
## Mysql常用数据类型（列类型）

分为四大类型：

1. 数值类型
2. 文本
3. 二进制数据类型
4. 时间日期



数值类型：int[4个字节]用得最多



## 数值型（整数）的基本使用

1. 说明，使用规范：在能够满足需求的情况下，尽量选择占用空间小的类型

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

```

#演示整型
#老韩使用 tinyint 来演示范围 有符号 -128~127 如果没有符号 0~255
#说明：表的字符集，校验规则，存储引擎，老师使用默认
#1. 如果没有指定 unsigned，则 TINYINT 就是有符号
#2. 如果指定unsigned，则TINYINT就是无符号 0~255
CREATE TABLE t3 (
    id TINYINT;

CREATE TABLE t4 (
    id TINYINT UNSIGNED);

INSERT INTO t3 VALUES(127); #这是非常简单的添加语句，输入-129、128就报错

SELECT * FROM t3

INSERT INTO t4 VALUES(255); #输入-1, 256就报错
SELECT * FROM t4

```

## bit类型

### 1. 基本使用

```
mysql> create table t05 (num bit(8));
mysql> insert into t05 (1, 3);
mysql> insert into t05 values(2, 65);
```

### 2. 细节说明 bit.sql

- bit字段显示时，按照位的方式显示
- 查询的时候仍然可以使用添加的数值
- 如果一个值只有0,1 可以考虑使用bit(1),可以节约空间
- 位类型。M指定位数，默认值1，范围1-64
- 使用不多

```
#演示bit的使用
#说明
#1. bit(m) m 在1-64
#2. 添加数据 范围 按照你给定的为数来确定，比如 m = 8 表示一个字节 0~255
#3. 显示按照bit
#4. 查询时，仍然可以按照数来查询
CREATE TABLE t05(num BIT(8));
INSERT INTO t05 VALUES(255);
SELECT * FROM t05;
SELECT * FROM t05 WHERE num = 1;
```

## 数值型（小数类型）的使用

### 1. FLOAT/DOUBLE [UNSIGNED]

Float 单精度精度， Double双精度

### 2. DECIMAL[M,D] [UNSIGNED]

- 可以支持更加精确的小数位。M是包括位数（精度）的总数，D是小数点（标度）后面的位数。
- 如果D是0，则值没有小数点或分数部分。M最大65、D最大是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
- 一般使用Double即可

```
#演示decimal类型、float、double使用
#创建表
CREATE TABLE t06(
    num1 FLOAT,
    num2 DOUBLE,
    num3 DECIMAL(30,0));

#添加数据
INSERT INTO t06 VALUES(88.12345678912345, 88.12345678912345, 88.12345678912345);
SELECT * FROM t06;
```

```

#decimal可以存放很大的数
CREATE TABLE t07 (
    num DECIMAL(65));

#decimal能存进去
INSERT INTO t07 VALUES(899999999999999999999998984784945784948574893);
SELECT * FROM t07;

CREATE TABLE t08(
    num BIGINT UNSIGNED);

#bigint类型存不进去
INSERT INTO t08 VALUES(899999999999999999999998984784945784948574893);
SELECT * FROM t08;

```

## 字符串

- CHAR(size)  
固定长度字符串 最大255 字符
- VARCHAR(size) 0~65535  
可变成都字符串 最大65532字节 【utf8编码最大21844字符 1-3个字节用于记录大小】
  - 注意区分CHAR最大长度是字符， VARCHAR最大长度是字节，而括号中的size填写的是字符。
  - 因此在创建VARCHAR(size)类型时需要将字节转换成->字符，如UTF-8编码， size应该填写size =  $(65535-3) / 3 = 21844$
  - 其中有3个字节是记录字段大小额，因此上面的计算需要先减

```

# 演示字符串类型使用char\varchar
#注释的快捷键 shift+ctrl+c, 注释取消 shift+ctrl+r

-- char(size)
-- 固定长度字符串 最大255字符
-- varchar(size) 0~65535
-- 可变成都字符串 最大65532字节 [utf8编码最大21844字符 1-3个字节用于记录大小]
-- 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 t09(
    `name` CHAR(255));

CREATE TABLE t10(
    `name` VARCHAR(32766)) CHARSET gbk;

DROP TABLE t10;

```

## 字符串的使用细节

### 细节1

- char(4) //这个4表示字符数（最大255），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表示字符数，不管是字母还是中文都以定义好的表编码来存放数据

## 细节2

- char(4)是定长 (固定的大小) , 就是说, 即使你插入 'aa', 也会占用分配的4个字符的空间
- varchar(4)是变长 (变化的大小) , 就是说, 如果你插入了'aa', 实际占用空间大小并不是4个字符, 而是按照实际占用空间来分配 (老韩说明: varchar本身还需要占用1-3个字节来存放内容长度)  $L(\text{实际数据大小}) + (1-3)\text{字节}$

## 细节3

- 什么时候使用 char, 什么时候使用 varchar?
    - 1) 如果数据是固定长度, 推荐使用char, 比如md5的密码, 邮编, 手机号, 身份证号码等. char(32)
    - 2) 如果一个字段的长度是不确定, 我们使用varchar, 比如留言, 文章
- 查询速度: char > varchar

## 细节4

- 在存放文本时, 也可以使用TEXT数据类型。可以将TEXT列视为VARCHAR列, 注意Text 不能有默认值, 大小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- $2^{32}$

## 日期类型

- 日期类型的基本使用

```
CREATE TABLE birthday6(
    t1 DATE, -- 年月日
    t2 DATETIME, -- 记录年月日 时分秒
    t3 TIMESTAMP
    NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP);
```

```
mysql> INSERT INTO birthday(t1,t2) VALUES('2022-11-11','2022-11-11 10:10:10');
```

该语句用于自动更新 timestamp 列:

```
NOT NULL DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP
```

```
#演示时间相关的类型
#创建一张表, date, datetime, timestamp
CREATE TABLE t14(
    birthday DATE, -- 生日
    job_time DATETIME, -- 记录年月日 时分秒
    login_time TIMESTAMP
    NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP); -- 登陆时间, 如果希望 login_time 列自动更新, 需要配置
```

```
DROP TABLE t14
SELECT * FROM t14;
INSERT INTO t14(birthday, job_time) -- 只更新birthday 和 job_time列
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列会自动的以当期那时间进行更新

- 日期类型的细节说明  
TimeStamp 在 Insert 和 Update时，自动更新

课堂练习：

```
#创建表的课堂练习
CREATE TABLE `emp` (
    Id INT,
    `name` VARCHAR(32),
    sex CHAR(1),
    birthday DATE,
    entry_date DATE,
    job VARCHAR(32),
    salary DOUBLE,
    `resume` TEXT) CHARSET utf8 COLLATE utf8_bin ENGINE INNODB

DROP TABLE t15
-- 添加一条
INSERT INTO `emp`
VALUES(100, '小妖怪', '男', '2000-11-11', '2010-11-10 11:11:11',
'巡山的', 3000, '大王叫我来巡山');

SELECT * FROM `emp`;
```

## 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 table_name
ADD (column datatype [DEFAULT expr]
[, column datatype]...);
```

- 修改列：

```
ALTER TABLE table_name
MODIFY (column datatype [DEFAULT expr]
[, column datatype]...);
```

- 删除列

```
ALTER TABLE table_name
DROP (column);
```

- 查看表的结构：desc 表名； -- 可以看表的所有的列

- 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；

```
#修改表的操作练习
-- 在 emp 上新增 image 列, varchar 类型 (要求在resume后面)
ALTER TABLE emp
ADD image VARCHAR(32) NOT NULL DEFAULT '' 
AFTER RESUME
DESC emp -- 显示表结构, 可以查看表的所有列

-- 修改 job 列, 使其长度为60
ALTER TABLE emp
MODIFY job VARCHAR(60) NOT NULL DEFAULT '' -- 非空, 默认为''

-- 删除 sex 列
ALTER TABLE emp
DROP sex

-- 修改表名为 employee
RENAME TABLE emp TO employee

-- 修改表的字符集为 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 ''
DESC employee
```

## CRUD语句

含义: C(create)R(read)U(update)D(delete)

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基本使用 (添加数据)

```
INSERT INTO table_name [(column [, column...])]
VALUES (value [, value...]);
```

```
#练习insert语句
CREATE TABLE `goods` (
    id INT,
    goods_name VARCHAR(10),
    price DOUBLE);
-- 添加数据
INSERT INTO `goods` (id, goods_name, price)
VALUES(10, '华为手机', 2000);
INSERT INTO `goods` (id, goods_name, price)
```

```
VALUES(20, '苹果手机', 3000);
SELECT * FROM `goods`;
```

- 细节说明

1. 插入的数据应与字段的数据类型相同
2. 数据的长度应在列的规定范围内
3. 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
4. 字符和日期类型数据应包含在单引号中
5. 列可以插入空值[前提是该字段允许为空]
6. insert into tab\_name (列名..) values (),(),() 形式添加多条记录
7. 如果是给表中的所有字段添加数据，可以不写前面的字段名称
8. 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，否则报错

```
#说明insert 语句的细节
#1. 插入的数据应与字段的数据类型相同
# 比如 把 'abc' 添加到 int 类型会错误
INSERT INTO `goods` (id, goods_name, price)
VALUES('韩顺平', '华为手机', 2000);
#2. 数据的长度应在列的规定范围内，例如：不能将一个长度为80的字符串加入到长度为40的列中
INSERT INTO `goods`(id, goods_name, price)
VALUES(40, 'vovo手机vovo手机vovo手机vovo手机vovo手机vovo手机', 3000);
#3. 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。
INSERT INTO `goods`(id, goods_name, price)
VALUES('vovo手机', 40, 2000);
#4. 字符和日期类型数据应包含在单引号中。
INSERT INTO `goods`(id, goods_name, price)
VALUES(40, vovo手机, 3000); -- 错误的 vovo手机 应该 'vovo手机'
#5. 列可以插入空值[前提是该字段允许为空]，insert into table value(null)
INSERT INTO `goods` (id, goods_name, price)
VALUES(40, 'vovo手机', NULL);
#6. insert into tab_name (列名..) values (),(),() 形式添加多条记录
INSERT INTO `goods` (id, goods_name, price)
VALUES(50, '三星手机', 2300), (60, '海尔手机', 1800);
#7. 如果是给表中的所有字段添加数据，可以不写前面的字段名称
INSERT INTO `goods`
VALUES(70, 'IBM手机', 5000);
#8. 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
INSERT INTO `goods` (id, goods_name)
VALUES(80, '格力手机');
SELECT * FROM goods;

INSERT INTO `goods2` (id, goods_name)
VALUES(80, '顺平手机');
SELECT * FROM goods2
```

## update语句 (更新数据)

```
UPDATE tb1_name
SET col_name = expr1 [, col_name=expr2 ...]
```

[WHERE where\_definition]

中括号[]括起来的语句为备选的意思，可写可不写

- 使用细节

1. UPDATE 语法可以用新值更新原有表行中的各列。
2. SET 子句指示要修改哪些列和要给与哪些值。
3. WHERE 子句指定应更新哪些行。如没有 WHERE 子句，则更新所有的行（记录），因此老师提醒一定要小心。
4. 如果需要修改多个字段，可以通过 set 字段1=值1，字段2=值2...

```
#演示update语句
```

```
#要求：在上面创建的employee表中修改表中的记录
```

```
#1. 将所有员工薪水修改为5000元。[如果没有带where 条件，会修改所有的记录，因此要小心]
```

```
UPDATE employee SET salary = 5000
```

```
SELECT * FROM employee
```

```
#2. 将姓名为 小妖怪 的员工薪水修改为3000元
```

```
UPDATE employee
```

```
SET salary = 3000
```

```
WHERE user_name = '小妖怪'
```

```
#3. 将 老妖怪 的薪水在原有基础上增加1000元
```

```
INSERT INTO employee
```

```
VALUES(200, '老妖怪', '1990-11-11', '2000-11-11', '捶背的', 5000, '给大王捶背的', 'c:\\ioTest\\a.jpg');
```

```
UPDATE employee
```

```
SET salary = salary + 1000
```

```
WHERE user_name = '老妖怪';
```

```
-- 可以修改多个列
```

```
UPDATE employee
```

```
SET salary = salary + 1000, job = '出主意的'
```

```
WHERE user_name = '老妖怪'
```

## delete语句（删除数据）

使用细节：

1. 如果不使用 where 子句，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，不删除表本身。如要删除表，使用 drop table 语句。drop table 表名

```
-- delete 语句演示
```

```
-- 删除表中名称为'老妖怪'的记录
```

```
DELETE FROM employee
```

```
WHERE user_name = '老妖怪';
```

```
-- 删除表中所有记录，老师提醒，一定要小心
```

```
DELETE FROM employee;
```

```
-- Delete语句不能删除某一列的值（可使用update 设为 null 或者 ''）
```

```
UPDATE employee SET job = '' WHERE user_name = '老妖怪'
```

```
SELECT * FROM employee
```

-- 要删除这个表

```
DROP TABLE employee;
```

## select语句（查找数据）**面试重点**

### 基本语法

```
SELECT [DISTINCT] * | {column1, column2, column3..}  
FROM table_name
```

- 注意事项

1. Select 指定查询哪些列的数据
2. column 指定列名
3. \*号代表查询所有列
4. From 指定查询那张表
5. DISTINCT 可选，指显示结果时，是否去掉重复数据

```
-- select 语句【重点、难点】
```

```
CREATE TABLE student(  
    id INT NOT NULL DEFAULT 1,  
    NAME VARCHAR(20) NOT NULL DEFAULT '',  
    chinese FLOAT NOT NULL DEFAULT 0.0,  
    english FLOAT NOT NULL DEFAULT 0.0,  
    math FLOAT NOT NULL DEFAULT 0.0);
```

```
INSERT INTO student (id, NAME, chinese, english, math) VALUES(1, '韩顺平', 89, 78, 56);  
INSERT INTO student (id, NAME, chinese, english, math) VALUES(2, '张飞', 67, 98, 56);  
INSERT INTO student (id, NAME, chinese, english, math) VALUES(3, '宋江', 87, 78, 77);  
INSERT INTO student (id, NAME, chinese, english, math) VALUES(4, '关羽', 88, 98, 90);  
INSERT INTO student (id, NAME, chinese, english, math) VALUES(5, '赵云', 82, 84, 67);  
INSERT INTO student (id, NAME, chinese, english, math) VALUES(6, '欧阳峰', 55, 85, 45);  
INSERT INTO student (id, NAME, chinese, english, math) VALUES(7, '黄蓉', 75, 65, 30);
```

```
SELECT * FROM student;
```

-- 查询表中所有学生的信息

```
SELECT * FROM student;
```

-- 查询表中所有学生的姓名和对应的英语成绩

```
SELECT `name`, english FROM student;
```

-- 过滤表中重复数据 distinct

```
SELECT DISTINCT english FROM student;
```

-- 要查询的记录，每个字段都相同，才会去重

```
SELECT DISTINCT `name`, english FROM student;
```

DISTINCT需要查询出来的记录，所有字段都相同了，才会去重

- 使用表达式对查询的列进行运算

```
SELECT *{column1 | expression, column2 | expression, ..}
```

```
FROM tablename;
```

- 在select语句中可使用 as 语句

```
SELECT column_name as 别名 from 表名;
```

运算之后以运算表达式作为列名会比较乱，因此使用 as 语句取一个列名

```
-- select 语句的使用
-- 统计每个学生的总分
SELECT `name`, (chinese + english + math) FROM student;
-- 在所有学生总分加10分的情况
SELECT `name`, (chinese + english + math + 10) FROM student;
-- 使用别名表示学生分数
SELECT `name` AS '名字', (chinese + english + math + 10) AS total_score
FROM student;
```

## 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'	模糊查询
	NOT LIKE ''	模糊查询
逻辑运算符	IS NULL	判断是否为空
	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);

课堂练习：

- 查询姓名为赵云的学生成绩
- 查询英语成绩大于90分的同学
- 查询总分大于200分的所有同学
- 查询math大于60，并且id大于4的学生成绩
- 查询英语成绩大于语文成绩的同学
- 查询总分大于200分，并且数学成绩小于语文成绩的姓赵的学生

```
-- select 语句
-- 查询姓名为赵云的学生成绩
SELECT * FROM student
WHERE `name` = '赵云'
-- 查询英语成绩大于 90 分的同学
SELECT `name` FROM student
WHERE english > 90;
```

```
-- 查询总分大于200分的所有同学
SELECT * FROM student
WHERE (chinese + english + math) > 200
-- 查询math大于60，并且id大于4的学生成绩
SELECT * FROM student
WHERE math > 60 AND id > 4
-- 查询英语成绩大于语文成绩的同学
SELECT * FROM student
WHERE english > math;
-- 查询总分大于200分 并且 数学成绩小于语文成绩 的姓赵的学生
-- 赵% 表示 名字以韩开头的就可以
SELECT * FROM student
WHERE (chinese + english + math) > 200 AND
      math < chinese AND `name` LIKE '赵%'

```

练习2：

```
-- 查询英语分数在80-90之间的同学
SELECT * FROM student
WHERE english >= 80 AND english <= 90;
SELECT * FROM student
WHERE english BETWEEN 80 AND 90; -- between . and .. 是闭区间
-- 查询数学分数为89、90、91的同学
SELECT * FROM student
WHERE math = 89 OR math = 90 OR math = 91
SELECT * FROM student
WHERE math IN (89, 90, 91);
-- 查询所有姓韩的学生成绩
SELECT * FROM student
WHERE `name` LIKE '韩%'
-- 查询数学分 >80， 语文分 >80 的同学
SELECT * FROM student
WHERE math > 80 AND chinese > 80

-- 1. 查询语文分数在 70-80 之间的同学
SELECT * FROM student
WHERE chinese BETWEEN 70 AND 80;
-- 2. 查询总分为185,190,191的同学
SELECT * FROM student
WHERE (chinese + math + english ) IN (185,190,191);
-- 3. 查询所有姓李 或者 姓宋 的学生成绩
SELECT * FROM student
WHERE `name` LIKE '李%' OR `name` LIKE '宋%';
-- 4. 查询数学比语文多30分的同学
SELECT * FROM student
WHERE (chinese - math) > 30
```

## 使用 order by 子句排序查询结果

数据库相关内容复习的时候可以直接看例句

```
SELECT column1, 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 子句应位于 SELECT 语句的结尾

课堂演示：

- 对数学成绩排序后输出牌[升序]
- 对总分按从高到低的顺序输出
- 对姓李的学生成绩排序输出 (升序)

```
-- 演示order by使用
-- 对数学成绩排序后输出[升序]
SELECT * FROM student
ORDER BY math;
-- 对总分按从高到低的顺序输出[降序]
SELECT `name` , (chinese + english + math) AS total_score FROM student
ORDER BY total_score DESC;
-- 对姓李的学生成绩[总分]排序输出 (升序) where + order by
SELECT `name` , (chinese + english + math) AS total_score FROM student
WHERE `name` LIKE '韩%'
ORDER BY total_score;
```

## 函数

### 合计/统计函数

- count

Count 返回行的总数

Select count(\*) | count(列名) from tablename  
[WHERE where\_definition]

- Sum

Sum 函数返回满足where条件的行的和- 一般使用在数值列

Select sum(列名) {,sum(列名)...} from tablename  
[WHERE where\_definition]

- AVG

AVG函数返回满足here条件的一列的平均值

Select avg(列名) {,avg(列名)...} from tablename  
[WHERE where\_definition]

- Max/Min

Max/Min函数返回满足where条件的一列的最大/最小值

Select max(列名) from tablename  
[WHERE where\_definition]

```
-- 演示mysql的统计函数的使用
-- 统计一个班级的人数有多少学生
SELECT COUNT(*) FROM student;
-- 统计数学成绩大于90的学生有多少个?
SELECT COUNT(*) FROM student
WHERE math > 90;
-- 统计总分大于250的人数有多少?
SELECT COUNT(*) FROM student
WHERE (math + english + chinese) > 250
-- count(*) 和 count(列) 的区别
-- 解释: count (*) 返回满足条件的记录的行数
-- count(列) : 统计满足条件的某列有多少个, 但是会排除为null
CREATE TABLE t15(
    name VARCHAR(20));
INSERT INTO t15 VALUES('tom');
INSERT INTO t15 VALUES('jack');
INSERT INTO t15 VALUES('mary');
INSERT INTO t15 VALUES(NULL);
SELECT * FROM t15;

SELECT COUNT(*) FROM t15; -- 4
SELECT COUNT(`name`) FROM t15; -- 3
```

```
-- 演示sum函数的使用
-- 统计一个班级数学总成绩?
SELECT SUM(math) FROM student;
-- 统计一个班级语文、英语、数学各科的总成绩
SELECT SUM(math) AS math_total_score, SUM(english), SUM(chinese) FROM student;
-- 统计一个班级语文、英语、数学的成绩总和
SELECT SUM(math + english + chinese) FROM student;
-- 统计一个班级语文成绩平均分
SELECT SUM(chinese)/COUNT(*) FROM student;
SELECT SUM(`name`) FROM student;

-- 演示avg的使用
-- 练习:
-- 求一个班级数学平均分
SELECT AVG(math) FROM student;
-- 求一个班级总分平均分
SELECT AVG(math + chinese + english) FROM student;

-- 演示Max/min的使用
-- 求班级最高分和最低分 (数值范围在统计中特别有用)
SELECT MAX(math + english + chinese), MIN(math + english + chinese)
FROM student;

-- 求出班级数学最高分和最低分
SELECT MAX(math) AS math_high_score, MIN(math) AS math_low_score
FROM student;
```

- 使用 group by 子句对列进行分组  
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

```

# 演示group by + having
# group by 用于对查询的结果分组统计
# having 子句用于限制分组显示结果.
# 如何显示每个部门的平均工资和最高工资
-- 老韩分析: avg(sal) max(sal)
-- 按照部门来分组查询
SELECT AVG(sal), MAX(sal), deptno
FROM emp GROUP BY deptno;

-- 显示每个部门的每种岗位的平均工资和最低工资
-- 老师分析 1. 显示每个部门的平均工资和最低工资
--          2. 显示每个部门每种岗位的平均工资和最低工资
SELECT AVG(sal), MIN(sal), deptno, job
FROM emp GROUP BY deptno, job;

-- 显示平均工资低于2000的部门号和它的平均工资 // 别名
-- 老师分析: [写sql语句的思路是化繁为简, 各个击破]
-- 1. 显示各个部门的平均工资和部门号
-- 2. 在1的结果基础上, 进行过滤, 保留 AVG
-- 3. 使用别名进行过滤

SELECT AVG(sal), deptno
FROM emp GROUP BY deptno
HAVING AVG(sal) < 2000;
-- 使用别名
SELECT AVG(sal) AS avg_sal, deptno
FROM emp GROUP BY deptno
HAVING avg_sal < 2000;

```

如何理解分组?

按照部门号进行分组, 然后进行处理

## 字符串相关函数

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

演示代码:

```
-- 演示字符串相关函数的使用
-- CHARSET (str) 返回字符串字符集
SELECT CHARSET(ename) FROM emp

-- CONCAT(string2 [,...]) 连接字符串,将多个列拼接成一列
SELECT CONCAT(ename, ' 工作是 ', job) FROM emp;

-- INSTR(string, substring) 返回 substring 在string中出现的位置, 没有返回0
-- DUAL 亚元表, 系统表 可以作为测试表使用
SELECT INSTR('hanshunping', 'ping') FROM DUAL;

-- UCASE(string2) 转换成大写
SELECT UCASE(ename) FROM emp;
-- LCASE (string) 转换成小写
SELECT LCASE(ename) FROM emp;

-- LEFT(string2, length) 从string2中的左边起取length个字符
-- RIGHT(string2, length) 从string2中的右边起取length个字符
SELECT LEFT(ename, 2), ename FROM emp;
SELECT RIGHT(ename, 2) , ename FROM emp;

-- LENGTH (string) string长度[按照字节]
SELECT LENGTH(ename) ,ename FROM emp;

-- REPLACE(str, search_str, replace_str)
-- 在str中用 replace_str 替换 search_str
-- 如果是 manager 就替换成 经理
SELECT job, REPLACE(job, 'MANAGER', '经理') FROM emp;

-- STRCMP(string1, string2) 逐字符比较两字符串大小
SELECT STRCMP('hsp', 'jsp') FROM DUAL;

-- SUBSTRING(str, position [,length])
-- 从 str 的 position 开始[从1开始计算], 取length个字符
-- 从ename 列的第一个位置开始取出2个字符
SELECT SUBSTRING(ename, 1, 2) FROM emp

-- LTRIM (string2) RTRIM (string2) TRIM (string)
-- 去除前端空格、后端空格、前端后端空格
SELECT LTRIM(' 韩顺平教育 ') FROM DUAL;
SELECT RTRIM(' 韩顺平教育 ') FROM DUAL;
SELECT TRIM(' 韩顺平教育 ') FROM DUAL;

练习: 以首字母小写的方式显示所有员工emp表的姓名
-- 使用两种方式
-- 方式一, 使用 concat 拼接 substring
SELECT CONCAT(LCASE(SUBSTRING(ename, 1,1)), SUBSTRING(ename,2)) FROM emp;
-- 方式二, 使用 concat 拼接 left 和 substring
SELECT CONCAT(LCASE(LEFT(ename, 1)), SUBSTRING(ename, 2)) FROM emp;

-- 老师讲解
-- 方法1:
-- 思路先取出ename 的第一个字符, 转成小写的
-- 把他和后面的字符串进行拼接输出即可
-- 方法2:
-- 使用LEFT替换substring

SELECT CONCAT(LCASE(SUBSTRING(ename,1,1)), SUBSTRING(ename,2)) AS new_name
```

```

FROM emp;

SELECT CONCAT(LCASE(LEFT(ename,1,1)), SUBSTRING(ename,2)) AS new_name
FROM emp;

```

## 数学相关函数

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

-- 演示数学相关函数

-- ABS(num) 绝对值  
`SELECT ABS(-10) FROM DUAL;`

-- BIN(decimal\_number) 十进制转二进制  
`SELECT BIN(10) FROM DUAL;`

-- CEILING(number2) 向上取整, 得到比num2 大的最小整数  
`SELECT CEILING(-1.1) FROM DUAL;`

-- CONV(number2, from\_base, to\_base) 进制转换  
-- 下面的含义是 8 是十进制的 8, 转成 2 进制输出  
`SELECT CONV(8, 10, 2) FROM DUAL;`

-- 下面的含义是 'C' 是16进制的 'C', 转成 10 进制输出  
`SELECT CONV('C', 16, 10) FROM DUAL;`

-- FLOOR(number2) 向下取整, 得到比 num2 小的最大整数  
`SELECT FLOOR(-1.1) FROM DUAL;`

-- FORMAT(number, decimal\_places) 保留小数位数 (四舍五入)  
`SELECT FORMAT(78.125458, 2) FROM DUAL;`

-- HEX(DecimalNumber ) 转十六进制  
`SELECT HEX(12) FROM DUAL`

-- LEAST(number, number2 [,...]) 求最小值  
`SELECT LEAST(0, 1, -10, 4) FROM DUAL;`

```
-- MOD (numerator, denominator) 求余
SELECT MOD(10, 3) FROM DUAL;

-- RAND([seed]) RAND([seed]) 返回随机数 其范围为 0 <= v <= 1.0
-- 老韩说明
-- 1. 如果使用 rand() 每次返回不同的随机数, 在 0 <= v <= 1.0
-- 2. 如果使用 rand(seed) 返回随机数, 范围 0 <= v <= 1.0, 如果seed不变,
-- 该随机数也不变化
SELECT RAND(6) FROM DUAL;
```

## 时间日期相关函数

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

<b>DATE(datetime)</b>	返回 <b>datetime</b> 的日期部分
<b>DATE_ADD(date2, INTERVAL d_value d_type)</b>	在 <b>date2</b> 中加上日期或时间
<b>DATE_SUB(date2, INTERVAL d_value d_type)</b>	在 <b>date2</b> 上减去一个时间
<b>DATEDIFF(date1, date2)</b>	两个日期差(结果是天)

上面函数的细节说明：

1. DATE\_ADD()中 interval 后面可以是 year minute second day等
2. DATE\_SUB()中 interval 后面可以是 year minute second hour 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

```
-- 日期时间相关函数

-- CURRENT_DATE() 当前日期
SELECT CURRENT_DATE() FROM DUAL;
-- CURRENT_TIME() 当前时间
SELECT CURRENT_TIME() FROM DUAL;
-- CURRENT_TIMESTAMP() 当前时间戳
```

```

SELECT CURRENT_TIMESTAMP() FROM DUAL;

-- 创建测试表 信息表
CREATE TABLE mes(
    id INT,
    content VARCHAR(30),
    send_time DATETIME);

-- 添加一条记录
INSERT INTO mes
VALUES(1, '北京新闻', CURRENT_TIMESTAMP())

INSERT INTO mes VALUES(2, '上海新闻', NOW())
INSERT INTO mes VALUES(3, '广州新闻', '2020-11-11')

SELECT * FROM mes;
SELECT NOW() FROM DUAL;

-- 上应用实例
-- 显示所有留言信息，发布日期只显示 日期，不用显示时间
SELECT id, content, DATE(send_time) FROM mes;

-- 请查询在10分钟内发布的新闻
SELECT *
FROM mes
WHERE DATE_ADD(send_time, INTERVAL 10 MINUTE) >= NOW()
SELECT *
FROM mes
WHERE send_time >= DATE_SUB(NOW(), INTERVAL 30 MINUTE)

-- 请在mysql 的sql语句中求出 2011-11-11 和 1990-1-1 相差多少天
SELECT DATEDIFF('2011-11-11', '1990-01-01')/365 FROM DUAL

-- 请用mysql 的sql语句求出你活了多少天? [练习] 1986-11-11 出生
SELECT DATEDIFF(NOW(), '1986-11-11') FROM DUAL;

-- 如果你能活80岁，求出你还能活多少天[练习] 1986-11-11 出生
-- 先求出活到80岁，是什么日期 x
-- 然后再使用 datediff(x, now());
-- INTERVAL 80 YEAR: YEAR 可以是 年月日，时分秒
-- '1986-11-11' 可以是 date, datetime, timestamp
SELECT DATEDIFF(DATE_ADD('1986-11-11', INTERVAL 80 YEAR), NOW())
FROM DUAL;

SELECT TIMEDIFF('10:11:11', '06:10:10') FROM DUAL;

```

<b>TIMEDIFF(date1,date2)</b>	两个时间差(多少小时多少分钟多少秒)
<b>NOW()</b>	当前时间
<b>YEAR Month DAY  DATE (datetime ) FROM_UNIXTIME() unix_timestamp();</b>	年月日

这里用的比较多的是 now()

```
-- YEAR|Month|DAY|DATE(datetime)
SELECT YEAR(NOW()) FROM DUAL;
SELECT MONTH(NOW()) FROM DUAL;
SELECT DAY(NOW()) FROM DUAL;
SELECT YEAR('2013-10-10') FROM DUAL;

-- unix_timestamp() 返回的是1970-1-1 到现在的秒数
SELECT UNIX_TIMESTAMP() FROM DUAL

-- FROM_UNIXTIME(): 可以把一个unix_timestamp 秒数[时间戳], 转成指定格式的日期
-- %Y-%m-%d 格式是规定好的, 表示年月日
-- 意义: 在开发中, 可以存放一个整数int, 然后表示时间, 通过FROM_UNIXTIME转换
SELECT FROM_UNIXTIME(1618483484, '%Y-%m-%d') FROM DUAL;
SELECT FROM_UNIXTIME(1618483484, '%Y-%m-%d %H:%i:%s') FROM DUAL;
```

在实际开发中，我们也经常使用int来保存一个unix时间戳，然后使用 from\_unixtime()进行转换，还是非常有实用价值的

## 加密和系统函数

<b>USER()</b>	查询用户
<b>DATABASE()</b>	数据库名称
<b>MD5(str)</b>	为字符串算出一个 MD5 32的字符串, (用户密码)加密
<b>PASSWORD(str)</b> 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

在开发的过程中，存储密码的时候，一定要记得使用 md5 或者 PASSWORD 函数进行加密

```
-- 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();

-- MD5(str) 为字符串算出一个 MD5 32的字符串。常用 (用户密码) 加密
-- root 密码是 hsp -> 加密md5 -> 在数据库中存放的是加密后的密码
SELECT MD5('hsp') FROM DUAL;
SELECT LENGTH(MD5('hsp')) FROM DUAL;

-- 演示用户表, 存放密码时, 是md5
CREATE TABLE hsp_user
(id INT,
`name` VARCHAR(32) NOT NULL DEFAULT '',
pwd CHAR(32) NOT NULL DEFAULT '');

INSERT INTO hsp_user
VALUES(100, '韩顺平', MD5('hsp'));
SELECT * FROM hsp_user;
```

```
SELECT * FROM hsp_user -- SQL注入问题
WHERE `name`='韩顺平' AND pwd = MD5('hsp')

-- PASSWORD(str) -- 加密函数, MySQL数据库的用户密码就是 PASSWORD函数加密

SELECT PASSWORD('hsp') FROM DUAL; -- 数据库的 *81220D972A52D4C51BB1C37518A2613706220DAC

-- 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
-- mysql.user 表示 数据库.表
```

## 流程控制函数

```
# 演示流程控制语句

# IF(expr1, expr2,expr3) 如果 expr1 为True, 则返回 expr2 否则返回 expr3
SELECT IF(TRUE, '北京', '上海') FROM DUAL;

# IFNULL(expr1, expr2) 如果expr1不为空NULL, 则返回 expr1, 否则返回expr2
SELECT IFNULL('jack', '韩顺平教育') FROM DUAL;
#

-- 类似java的case语句
SELECT CASE
WHEN TRUE THEN 'jack' -- jack
WHEN FALSE THEN 'tom' --
ELSE 'mary' END

-- 1. 查询emp表, 如果 comm 是null, 则显示0.0
-- 老师说明, 判断是否为空, 要使用is null, 判断不为空 使用is not null
SELECT ename, IF(comm IS NULL, 0.0, comm)
FROM emp;

SELECT ename, IFNULL(comm, 0.0)
FROM emp

-- 2. 如果emp 表的 job 是 CLERK 则显示 职员, 如果是 MANAGER 则显示经理
-- 如果是 SALESMAN 则显示销售人员, 其它正常显示

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

## mysql表查询--加强

### where、like、order by

- 使用 where 子句  
? 如何查找1992.1.1后入职的员工
- 如何使用 like 操作符  
%: 表示单个或任意个字符, \_:表示单个字符
- 如何使用order by子句

```
-- 查询加强
-- 使用where子句
-- ?如何查找1992.1.1后入职的员工
SELECT * FROM emp
WHERE hiredate <= '1992-01-01'
-- 如何使用like操作符 (模糊)
-- %: 表示0到多个任意字符 _: 表示单个任意字符
-- ?如何显示首字符为s的员工姓名和工资
SELECT ename, sal FROM emp
WHERE ename LIKE 'S%'
-- ? 如何显示第三个字符为大写O的所有员工的姓名和工资
SELECT ename, sal FROM
WHERE ename LIKE '__O%'

-- 如何显示没有上级的员工信息
SELECT * FROM emp
WHERE mgr IS NULL;
-- 查询表的结构
DESC emp

-- 使用order by子句(注意不是group by)
-- ? 如何按照工资的从低到高的顺序[升序], 显示雇员的信息
SELECT * FROM emp
ORDER BY sal

SELECT * FROM emp
ORDER BY deptno ASC, sal DESC
```

## 分页查询 (重要)

```
-- 推导一个公式
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
```

```
# 分页查询
-- 1. 按雇员的id号升序取出, 每页显示3条, 请分别显示 第1页, 第2页, 第3页

-- 第1页
SELECT * FROM emp
ORDER BY empno
LIMIT 0, 3 -- 从第1行开始取 (0+1), 取3行

-- 第2页
SELECT * FROM emp
```

```
ORDER BY empno  
LIMIT 3, 3 -- 从第4行开始取 (3+1) , 取3行
```

-- 第3页

```
SELECT * FROM emp  
ORDER BY empno  
LIMIT 6, 3 -- 从第7行开始取 (6+1) , 取3行
```

-- 推导一个公式

```
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
```

课堂练习：

-- 按照雇员的empno号降序取出，每页显示5条记录。请分别显示 第3页，第5页 对应的sql语句

```
SELECT * FROM emp  
ORDER BY empno DESC  
LIMIT 10, 5
```

```
SELECT * FROM emp  
ORDER BY empno DESC  
LIMIT 20, 5
```

## group by 加强

-- 增强group by的使用

-- (1)显示每种岗位的雇员总数、平均工资

```
SELECT COUNT(*), AVG(sal), job  
FROM emp  
GROUP BY job;
```

-- (2) 显示雇员总数, 以及获得补助的雇员数

-- 思路：获得补助的雇员数 就是 comm 列为非null, 就是count (列) , 如果该列的

-- 值为null, 是不会统计,SQL 非常灵活, 需要我们动脑筋

```
SELECT COUNT(*), COUNT(comm)  
FROM emp
```

-- 老师的扩展要求：统计没有获得补助的雇员数

```
SELECT COUNT(*), COUNT(IF(comm IS NULL, 1, NULL))  
FROM emp
```

-- 上面的IF(comm IS NULL, 1, NULL)返回的就是一列

-- select IF(comm IS NULL, 1, NULL) from emp

```
SELECT COUNT(*), COUNT(*) - COUNT(comm)  
FROM emp
```

-- (3) 显示管理者的总人数, 去重使用distinct

```
SELECT COUNT(DISTINCT mgr)  
FROM emp;
```

-- (4) 显示雇员工资的最大差额

-- 思路: max(sal) - min(sal)

```
SELECT MAX(sal) - MIN(sal)  
FROM emp;
```

小技巧：尝试写->修改->尝试[慢慢就可以找到正确的写法]

# 数据分组总结

如果select语句同事包含有group by, having, limit, order by  
name他们的顺序是group by, having, order by, limit

```
SELECT deptno, AVG(sal) AS avg_sal
  FROM emp
 GROUP BY deptno
 HAVING avg_sal > 1000
 ORDER BY avg_sal DESC
 LIMIT 0, 2
```

# mysql多表查询

- 问题的引出 (**重点, 难点**)  
商品和评论放在一起
- 说明  
多表查询是指基于两个和两个以上的表查询。在实际应用中，查询单个表可能不能满足你的需求

## 多表查询-默认情况

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，第一张表的行数 \* 第二张表的行数**
3. 这样多表查询默认处理返回的结果，称为笛卡尔集
4. 解决这个多表的关键就是要写出正确的过滤条件 where，需要程序员进行分析

```
-- 多表查询
-- ? 显示雇员名, 雇员工资及所在部门的名字【笛卡尔集】
/*
老韩分析:
1. 雇员名、雇员工资 来自emp表
2. 部门的名字 来自 dept表
3. 需求对 emp 和 dept 查询 ename, sal, dname, deptno
4. 当我们需要指定显示某个表的列时, 需要 表名.列名
*/
SELECT ename, sal, dname, emp.deptno
  FROM emp, dept
 WHERE emp.deptno = dept.`deptno`
SELECT * FROM emp
SELECT * FROM dept
SELECT * FROM salgrade
-- 老韩小技巧

SELECT ename, sal, dname, emp.deptno
  FROM emp, dept
 WHERE emp.deptno = dept.`deptno` AND emp.deptno = 10
```

```
-- 思路: 姓名, 工资 来自 emp表
-- 工资级别 salgrade
-- 写sql, 先写一个简单的, 然后加入过滤条件...
SELECT ename, sal, grade
FROM emp, salgrade
WHERE sal BETWEEN losal AND hisal;

-- 课堂练习: 显示雇员名, 雇员工资以及所在部门的名字, 并按部门排序[降序排列]
SELECT ename, sal, dept.`deptno`
FROM emp, dept
WHERE emp.`deptno` = dept.`deptno`
ORDER BY emp.`deptno` DESC
```

## 自连接

自连接是指在同一张表的连接查询【将同一张表看做两张表】

```
-- 思考题: 显示公司员工和他的上级的名字
-- 老韩分析: 员工名字 在emp, 上级的名字的名字 emp
-- 员工和上级是通过 emp 表的 mgr 列关联
-- 这边老师小结:
-- 自连接的特点 1. 把同一张表当做两张表使用
--           2. 需要给表取别名: 表名 表别名
--           3. 列名不明确, 可以指定列的别名 列名 AS 列的别名
SELECT worker.ename AS '职员名' , boss.`ename` AS '上级名'
FROM emp worker, emp boss -- 169
WHERE worker.mgr = boss.empno
```

这里最好自己写一写案例

## mysql表子查询

- 什么是子查询?  
子查询是指嵌入在其它sql语句中的select语句, 也叫嵌套查询
- 单行子查询  
单行子查询是指只返回一行数据的子查询语句
- 多行子查询  
多行子查询指返回多行数据的子查询 使用关键字 in

```
-- 子查询的演示
-- 请思考: 如何显示与SMITH同一部门的所有员工?
/*
1. 先查询到 SMITH的部门号得到
2. 把上面的select 语句当做一个子查询来使用
```

```
/*
SELECT deptno
FROM emp
WHERE ename = 'SMITH'
-- 下面的答案:
SELECT *
FROM emp
WHERE deptno = (
SELECT deptno
FROM emp
WHERE ename = 'SMITH'
)
```

-- 多行子查询  
-- 课堂练习：如何查询和部门10的工作相同的雇员的  
-- 名字、岗位、工资、部门号，但是不含10号部门自己的雇员

```
/*
1. 查询到10号部门有哪些工作岗位
2. 把上面查询的结果当做子查询使用
*/
SELECT DISTINCT job
FROM emp
WHERE deptno = 10;
```

-- 下面语句完整

```
SELECT ename, job, sal, deptno
FROM emp
WHERE job IN (
SELECT DISTINCT job
FROM emp
WHERE deptno = 10
) AND deptno != 10
```

## 子查询当做临时表使用

-- 查询ecshop中各个类别中，价格最高的商品  
-- 查询 商品表  
-- 先得到 各个类别中，价格最高的商品 max + group by cat\_id，当做临时表  
-- 可以将子查询当做一张临时表使用

```
SELECT goods_id, cat_id, goods_name, shop_price
FROM ecs_goods
GROUP BY cat_id

SELECT goods_id, cat_id, goods_name, shop_price
FROM ( -- temp表取到各商品分类的最高价格
SELECT cat_id, MAX(shop_price) AS max_price
FROM ecs_goods
GROUP BY cat_id
) temp, ecs_goods -- 这里是多表查询，得到笛卡尔表
WHERE temp.cat_id = ecs_goods.cat_id
AND temp.max_price = ecs_goods.shop_price -- 添加条件筛选出最高的价格
```

## 在多行子查询中使用all操作符

-- all 和 any的使用

-- 请思考：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
)
```

-- 可以这样写

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

-- 请思考：如何显示工资比部门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
)
```

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

## 多列子查询

多列子查询则是查询返回多个列数据的子查询语句

-- 多列子查询

-- 请思考如何查询与smith的部门和岗位完全相同的所有雇员（并且不含smith本人）

-- (字段1, 字段2 ...) = (select 字段1, 字段2 from...)

-- 分析：1. 得到smith的部门和岗位

```
SELECT deptno, job
FROM emp
WHERE `ename` = 'SMITH'
```

```
-- 分析: 2. 把上面的查询当做子查询来使用, 并且使用多列子查询的语法进行匹配
SELECT *
FROM emp
WHERE (deptno, job) = ( -- 语法
SELECT deptno, job
FROM emp
WHERE `ename` = 'SMITH'
) AND ename != 'SMITH'
```

-- 练习: 请查询和宋江数学, 语文, 英语成绩完全相同的学生

```
SELECT *
FROM student
WHERE (chinese, english, math) = ( -- 语法
SELECT chinese, english, math
FROM student
WHERE `name` = '宋江'
)
```

## 子查询练习

-- 子查询练习

-- 请思考: 查找每个部门工资高于本部门平均工资的人的资料  
-- 这里要用到数据查询的小技巧, 把一个子查询当做一个临时表使用

-- 1. 先得到每个部门的 部门号和 对应的平均工资

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
```

-- 2. 把上面的结果当做子查询。和 emp 进行多表查询

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

-- 查找每个部门工资最高的人的详细资料

-- 1. 先查询每个部门的最高工资  
-- 2. 把上面得到的表当做一个临时表进行子查询

```
SELECT deptno, MAX(sal)
FROM emp
GROUP BY deptno
```

```
SELECT *
FROM emp, (SELECT deptno, MAX(sal) AS max_sal
FROM emp
GROUP BY deptno
) temp
WHERE emp.`sal` = temp.max_sal
AND emp.`deptno` = temp.deptno
```

```
-- 查询每个部门的信息(部门名, 编号, 地址) 和人员数量
```

```
-- 1. 部门名, 来自 dept表
```

```
-- 2. 各个部门的人员数量 -> 构建一个零时表
```

```
SELECT *
```

```
FROM emp, dept
```

```
WHERE emp.`deptno` = dept.`deptno`
```

```
-- 人员数量
```

```
SELECT COUNT(*), deptno
```

```
FROM emp
```

```
GROUP BY deptno
```

```
SELECT dname, dept.deptno, loc, per_num AS '人数'
```

```
FROM dept, (
```

```
SELECT COUNT(*) AS per_num, deptno
```

```
FROM emp
```

```
GROUP BY deptno
```

```
) tmp
```

```
WHERE tmp.deptno = dept.`deptno`
```

```
-- 还有一种方法, 表.* 表示将该表所有列都显示出来
```

```
SELECT tmp.*, dname, loc
```

```
FROM dept, (
```

```
SELECT COUNT(*) AS per_num, deptno
```

```
FROM emp
```

```
GROUP BY deptno
```

```
) tmp
```

```
WHERE tmp.deptno = dept.`deptno`
```

## 表复制和去重

```
-- 表的复制
```

```
-- 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
```

```
CREATE TABLE my_tab01
```

```
( id INT,  
`name` VARCHAR(32),  
sal DOUBLE,  
job VARCHAR(32),  
deptno INT);
```

```
DESC my_tab01
```

```
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;
```

```
SELECT COUNT(*) FROM my_tab01
```

```

-- 如何删掉一张表的重复记录
-- 1. 先创建一张表 my_tab01
-- 2. 让 my_tab02 有重复的记录

CREATE TABLE my_tab02 LIKE emp; -- 这个语句 把emp表的结构(列), 复制到my_tab02

DESC my_tab02;

INSERT INTO my_tab02
SELECT * FROM emp

SELECT * FROM my_tab02
-- 3. 考虑如何去重
/*
思路
(1) 先创建一张临时表 my_temp, 该表的结构和 my_tab02一致
(2) 把my_temp 的记录 通过 distinct 关键字 处理后 把记录复制到 my_temp 表
(3) 清除掉 my_tab02 记录
(4) 把my_temp 表的记录复制到 my_tab02
(5) drop 掉临时表
*/
-- (1) 先创建一张临时表 my_temp, 该表的结构和my_tab01一样
CREATE TABLE my_temp LIKE my_tab02
-- (2) 把my_tab02的记录 distinct 处理后复制到 my_temp
INSERT INTO my_temp
SELECT DISTINCT * FROM my_tab02
SELECT * FROM my_temp
-- (3) 将my_teb02的数据全部删除
DELETE FROM my_tab02
-- (4) 将my_temp 表的记录复制到my_tab02
INSERT INTO my_tab02
SELECT * FROM my_temp
-- (5) 将my_temp临时表删除
DROP TABLE my_temp

SELECT * FROM my_tab02

```

## 合并查询

- 将两个查询结果合并

```

-- 合并查询

SELECT ename, sal, job FROM emp WHERE sal>2500 -- 5

SELECT ename, sal, job FROM emp WHERE job='MANAGER' -- 3

-- 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' -- 8

-- union 就是将两个查询结果合并, 会去重
SELECT ename, sal, job FROM emp WHERE sal>2500

```

```
UNION
SELECT ename, sal, job FROM emp WHERE job='MANAGER' -- 6
```

# 外连接

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

语法: select .. from 表1 left join 表2 on 条件

```
-- 外连接
-- 创建 stu
CREATE TABLE stu (
    id INT,
    `name` VARCHAR(32));
```

```
INSERT INTO stu VALUES(1, 'jack'),(2, 'tom'),(3, 'kity'),(4, 'nono');
SELECT * FROM stu
```

```
-- 创建exam
CREATE TABLE exam(
    id INT,
    grade INT);
INSERT INTO exam VALUES(1, 56),(2,76),(11, 8)
```

-- 使用左连接 (显示所有人的成绩, 如果没有成绩, 也要显示该人的姓名和id, 成绩显示为空)

```
SELECT `name`, stu.`id`, grade
FROM stu, exam
WHERE stu.`id` = exam.`id`
```

```
-- 改成左外连接
SELECT `name`, stu.`id`, grade
FROM stu LEFT JOIN exam
ON stu.`id` = exam.`id`
```

-- 使用右外连接 (显示所有成绩, 如果没有名字匹配, 显示空)

-- 即, 右边的表 (exam) 和左表没有匹配的记录, 也会把右表的记录显示出来

```
SELECT `name`, stu.`id`, grade
FROM stu RIGHT JOIN exam
ON stu.`id` = exam.`id`
```

```
-- 课堂练习
-- 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名
-- 使用左外连接:
SELECT dept.dname, ename, job
FROM dept LEFT JOIN emp
ON dept.`deptno` = emp.`deptno`
```

```
-- 使用右外连接
SELECT dept.dname, ename, job
FROM emp RIGHT JOIN dept
ON dept.`deptno` = emp.`deptno`
```

```
-- 总结:  
-- 在实际的开发中，我们绝大多数情况使用的是内连接  
-- 外连接用得比较少
```

## 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

## 主键

primary key(主键)-细节说明

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. 直接在字段后指定：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
5. 在实际开发中，每个表往往都会有一个主键

```
-- 主键使用
```

```
-- id name email
```

```
CREATE TABLE t17  
(id INT PRIMARY KEY,  
 `name` VARCHAR(32),  
 email VARCHAR(32));
```

```
-- 主键列的值是不可以重复
```

```
INSERT INTO t17  
VALUES(1, 'jack', 'jack@sohu.com');
```

```
INSERT INTO t17  
VALUES(2, 'tom', 'jack@sohu.com');
```

```
INSERT INTO t17  
VALUES(2, 'hsp', 'hsp@sohu.com');
```

```
SELECT * FROM t17;
```

```
-- 主键使用的细节讨论
```

```
-- primary key 不能重复而且不能为null
```

```
INSERT INTO t17  
VALUES(NULL, 'hsp', 'hsp@sohu.com');
```

```
-- 一张表最多只能有一个主键，但可以是复合主键
```

```
CREATE TABLE t18  
(id INT PRIMARY KEY, -- 表示id是主键  
 `name` VARCHAR(32) PRIMARY KEY, -- 错误的  
 email VARCHAR(32));
```

```
-- 演示复合主键 (id 和 name 做成复合主键)
```

```
CREATE TABLE t18
```

```

(id INT ,
`name` VARCHAR(32),
email VARCHAR(32),
PRIMARY KEY(id, `name`) -- 这里就是复合主键
);
INSERT INTO t18
VALUES(1, 'tom', 'tom@sohu.com');
INSERT INTO t18
VALUES(1, 'jack', 'jack@sohu.com');
INSERT INTO t18
VALUES(1, 'tom', 'xx@sohu.com'); -- 这里就违反了复合主键
SELECT * FROM t18

-- 主键的指定方式 有两种
-- 1. 直接在字段后指定: 字段名 primary key
-- 2. 在表定义最后写 primary key(列名) ;
CREATE TABLE t19
(id INT,
`name` VARCHAR(32) PRIMARY KEY,
email VARCHAR(32)
);

CREATE TABLE t20
(id INT,
`name` VARCHAR(32),
email VARCHAR(32),
PRIMARY KEY(`name`) -- 在表定义最后写 primary key
);

-- 使用desc 表名, 可以看到primary key的情况

DESC t20 -- 查看 t20表的结果, 显示约束的情况
DESC t18

```

## not null (非空)

语法: 字段名 字段类型 not null

## unique的使用

```

-- unique的使用

CREATE TABLE t21
(id INT UNIQUE, -- 表示 id 列是不可以重复的
`name` VARCHAR(32),
email VARCHAR(32)
);

INSERT INTO t21
VALUES(1, 'jack', 'jack@sohu.com');

INSERT INTO t21
VALUES(1, 'tom', 'tom@sohu.com');

-- unique使用细节

```

```

-- 1. 如果没有指定 not null, 则 unique 字段可以有多个null
-- 如果一个列(字段), 是 unique not null 使用效果类似 primary key
INSERT INTO t21
VALUES(NULL, 'tom', 'tom@sohu.com');
SELECT * FROM t21

-- 2. 一张表可以有多个 unique 字段
CREATE TABLE t22
(id INT UNIQUE, -- 表示 id 列是不可以重复的
`name` VARCHAR(32) UNIQUE, -- 表示 name 列是不可以重复的
email VARCHAR(32)
);

INSERT INTO t21
VALUES(NULL, 'tom', 'tom@sohu.com');

```

## 外键约束



分析：班级表中的id作为学生表 class\_id 的外键  
这时候学生表中插入 class\_id 不在班级表的 id 中  
则会导致插入失败

- **foreign 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，主表则必须具有主键约束或 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是为null (学生/班级 图示)

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

```
-- 外键演示

-- 创建 主表 my_class 班级表
CREATE TABLE my_class(
    id INT PRIMARY KEY, -- 班级编号
    `name` VARCHAR(32) NOT NULL DEFAULT '';

-- 创建 从表 my_stu
CREATE TABLE my_stu
    (id INT PRIMARY KEY, -- 学生编号
    `name` VARCHAR(32) NOT NULL DEFAULT '',
    class_id INT, -- 学生所在班级的编号
    -- 下面指定外键关系
    FOREIGN KEY (class_id) REFERENCES my_class(id));

-- 测试数据
INSERT INTO my_class
VALUES(100, 'java'), (200, 'web');
INSERT INTO my_class
VALUES(300, 'php');

SELECT * FROM my_class;

INSERT INTO my_stu
VALUES(1, 'tom', 100);
INSERT INTO my_stu
VALUES(2, 'jack', 200);
INSERT INTO my_stu
VALUES(4, 'mary', 400); -- 这里会失败...因为400班级不存在
SELECT * FROM my_stu;
```

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
5. 一旦建立主外键关系，数据不能随意删除了  
如果有指向主表的外键约束，则该条数据就不能被随意删除

## 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

```
-- 演示check的使用
-- mysql5.7目前还不支持 check，只做语法校验，但不会生效

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

```
);

-- 添加数据
INSERT INTO t23
VALUES(1, 'jack', 'mid', 1);
SELECT * FROM t23
```

## 自增长

1. 一般自增长是和primary key组合使用
2. 自增长也可以单独使用[但是需要配合unique使用]
3. 自增长一般为整型int
4. 自增长默认从1开始，也可以通过以下的命令修改ALTER TABLE t25 AUTO\_INCREMENT = 100
5. 如果添加数据的时候，指定了自增长字段（列）的值，则以指定值为准。**如果指定了自增长，一般来说，就按照自增长的规则来添加数据。**

```
-- 演示自增长的使用
-- 创建表
CREATE TABLE t24
(id INT PRIMARY KEY AUTO_INCREMENT,
email VARCHAR(32)NOT NULL DEFAULT '',
`name` VARCHAR(32)NOT NULL DEFAULT '');

DESC t24
-- 测试自增长的使用
INSERT INTO t24
VALUES(NULL, 'tom@qq.com', 'jack');

INSERT INTO t24
(email, `name`) VALUES('hsp@sohu.com', 'hsp');

SELECT * FROM t24

-- 修改默认的自增长开始值

CREATE TABLE t25
(id INT PRIMARY KEY AUTO_INCREMENT,
email VARCHAR(32)NOT NULL DEFAULT '',
`name` VARCHAR(32)NOT NULL DEFAULT '');
ALTER TABLE t25 AUTO_INCREMENT = 100
INSERT INTO t25
VALUES(NULL, 'tom@qq.com', 'tom');
INSERT INTO t25
VALUES(666, 'hsp@qq.com', 'hsp');
SELECT * FROM t25
```

## mysql索引

说起提高数据库的性能，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。

```
-- 创建测试数据库 temp  
-- 创建(列)索引，能够提高数据库的性能，加快查询速度  
-- 在创建索引前，文件大小是 524m，创建索引后为655m，占用的空间不小  
CREATE INDEX empno_index ON emp(empno)
```

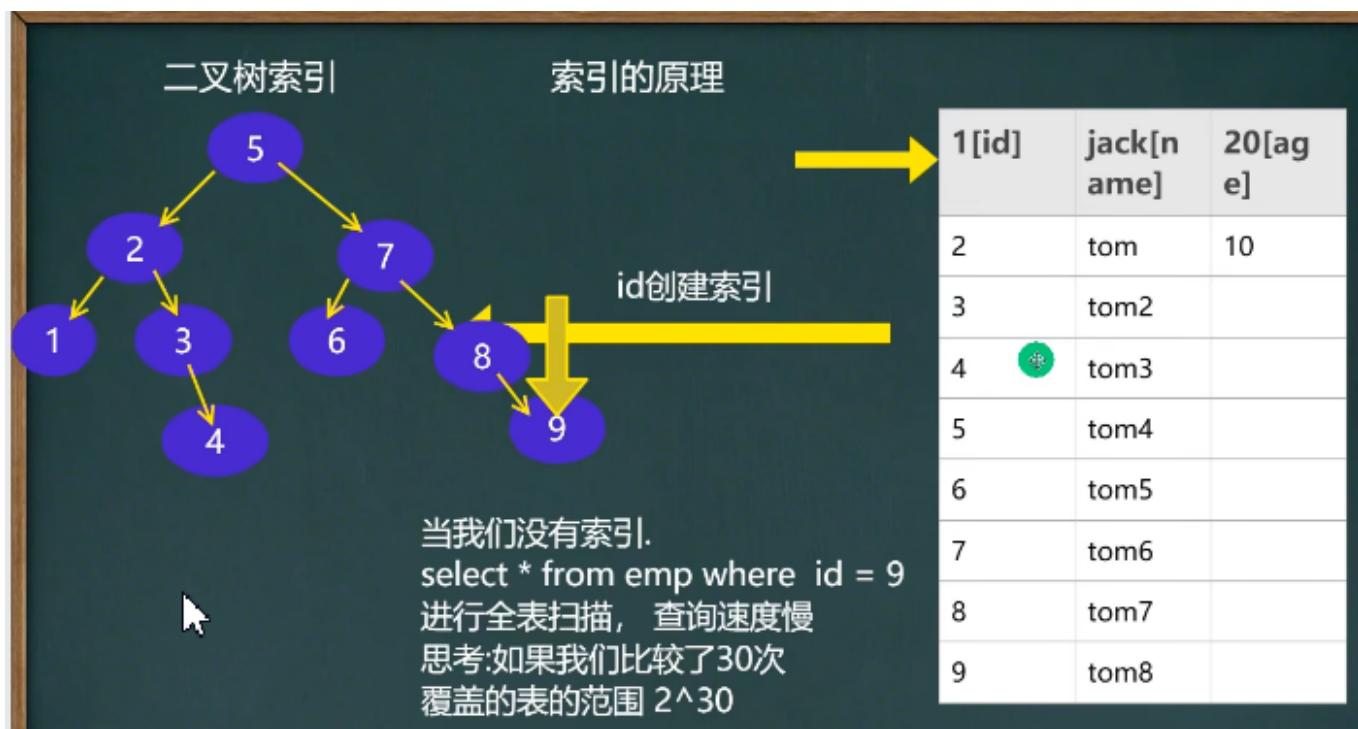
```
SELECT *  
FROM emp  
WHERE empno =1234578 -- 0.003s 原来是4.5s
```

-- 创建索引后，只对创建了索引的列有效

```
SELECT *  
FROM emp  
WHERE ename = 'axJxCo' -- 没有在ename创建索引时，时间4.7s
```

```
CREATE INDEX ename_index ON emp(ename) -- 创建索引后，速度变快
```

## 索引的原理



- 没有索引为什么会慢?  
因为需要全盘扫描
- 使用索引为什么会快?  
因为会形成一个索引的数据结构，比如二叉树
- 思考，如果我们比较了30次，能够覆盖多少条数据?  
能够覆盖 $2^{30}$ 次方条数据
- 如果对表进行dml (update delete insert)，会对索引进行维护，对速度有影响。  
在我们的项目中，select[90%] 多还是update, delete, insert[10%]操作多？

## mysql索引

- 索引的类型

1. 主键索引，主键自动的为主索引 (类型primary key)
  2. 唯一索引 (UNIQUE)
  3. 普通索引 (INDEX)
  4. 全文索引 (FULLTEXT) 一般不使用
- 实际开发中使用全文搜索 Solr 和 ElasticSearch (ES)

```
-- 演示mysql的索引的使用
-- 创建索引
DROP TABLE t25
CREATE TABLE t25 (
    id INT,
    `name` VARCHAR(32));

-- 查询表是否有索引
SHOW INDEXES FROM t25

-- 添加唯一索引
CREATE UNIQUE INDEX id_index ON t25(id);
-- 添加普通索引
CREATE INDEX id_index ON t25(id);
-- 如何选择
-- 1. 如果某列的值，是不会重复的，则优先考虑使用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
CREATE TABLE t26 (
    id INT,
    `name` VARCHAR(32));

ALTER TABLE t26 ADD PRIMARY KEY(id)

SHOW INDEX FROM t26

-- 删除索引
DROP INDEX id_index ON t25

-- 删除主键索引
ALTER TABLE t26 DROP PRIMARY KEY

-- 修改索引，先删除，在添加新的索引

-- 查询索引
-- 1. 方式
SHOW INDEXES FROM t25
-- 2. 方式
SHOW INDEXES FROM t25
-- 3. 方式
SHOW KEYS FROM t25
-- 4. 方式
DESC t25 -- Key上写着MUL的代表有索引，不推荐使用该种方式，因为信息少
```

分析：添加索引、删除索引、查询索引

# 哪些列上适合使用索引

1. 较频繁的作为查询条件字段应该创建索引
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  
如性别 = 男/女
3. 更新非常频繁的字段不适合创建索引
4. 不会出现在WHERE子句中字段不该创建索引

## mysql事务

- 什么是事务

事务用于保证数据的一致性。将多个dml语句当成一个整体，要么全部成功，要么全部失败。例如：转账就需要用到事务来处理，用以保证数据的一致性。

dml操作：数据操作语言(Data Manipulation Language)

当执行事务操作时（dml语句），mysql会在表上加锁，防止其它用户该表的数据。这对用户来讲是非常重要的

- mysql 数据库控制台事务的几个重要操作

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

-- 事务操作的示意图

```
-- 1. 创建一张表
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操作
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 a
ROLLBACK TO a
-- 如果这样，表示直接回退到事务开始的状态
ROLLBACK
```

- 回退事务  
在结束事务之前 (commit) , 通过保存点可以回退到指定的点。
- 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。当使用commit语句结束事务后, 其它会话[其他连接]将可以看到事务变化后的新数据[所有数据就正式生效.]

## 事务细节

1. 如果不开始事务, 默认情况下, dml操作时自动提交的, 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  
-- 默认就是回退到你事务开始的状态
3. 你也可以在这个事务中 (没有提交前) , 创建多个保存点
4. 你可以在事务没有提交前, 选择回退到哪个保存点
5. InnoDB 存储引擎支持事务, MyISAM 不支持
6. 开始一个事物 start transaction 或 set autocommit=off

## 事务隔离级别介绍 (重要)

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, 以保证各个连接在获取数据时的准确性
  2. 如果不考虑隔离性, 可能会引发如下问题
    - 1) 脏读
    - 2) 不可重复读
    - 3) 幻读
- 脏读 (dirty read): 当一个事务读取另一个事物尚未提交的修改时, 产生脏读
  - 不可重复读 (nonrepeatable read) : 同一查询在同一事务中多次进行, 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) : 同一查询在同一事务中多次进行, 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

## 事务隔离级别

## Mysql隔离级别定义了事物与事物之间的隔离程度

Mysql隔离级别 (4种)	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

分析：其实这些隔离级别可以从字面意思理解

1. 读未提交：即一个事务能够读取到另一个事务的还没有提交 (commit) 的修改
2. 读已提交：即一个事务能够读取到另一个事物已经提交 (commit) 的修改
3. 可重复读：一个事务能够重复读取到一样的数据，其它事务的dml操作，提交的以及不提交的事务都不会影响到该结果。
4. 可串行化：通过加锁的方式，当一个事务在操作的时候，还未commit，另一个事务进行操作时，会阻塞。只有当事务commit了之后，别的事务才能够进行操作。

```
-- 查看当前会话隔离级别
SELECT @@tx_isolation

-- 查看当前系统的隔离级别
SELECT @@global.tx_isolation

-- 设置当前会话的隔离级别
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- 设置默认设置的隔离级别
SET GLOBAL TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- 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需求）
```

- 全局修改，修改mysql中my.ini配置文件，在最后加上  
transaction-isolation = REPEATABLE-READ  
可选参数：READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, SERIALIZABLE

## 事务的ACID (知道就好)

1. 原子性 (Atomicity)  
原子性是指事务时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外一个一致性状态

### 3. 隔离性 (Isolation)

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

### 4. 持久性 (Durability)

持久性是指一个事物一旦被提交，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应该对其有任何影响

## mysql表类型和存储引擎

- 基本介绍

1. MySQL的表类型有存储引擎 (Storage Engines) 决定，主要包括MyISAM、InnoDB、Memory等
2. MySQL数据表主要支持六种类型，分别是：CSV、Memory、ARCHIVE、MRG\_MYISAM、MYSIAM、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“非事务安全性型” (non-transaction-safe)

● 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				
锁机制	表锁	行锁	表锁	行锁	
存储限制	没有	64TB	有	没有	
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			

4. 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
5. 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
6. 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类型的表访问非常的快，因为他的数据是放在内存中的，并且默认使用HASH索引。但是一旦MySQL服务关闭，表中的数据就会丢失掉，表的结构还在

## 三种存储引擎表的使用案例

- 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

```
-- 表类型和存储引擎

-- 查看所有的存储引擎
SHOW ENGINES

-- innodb 存储引擎，是前面使用过
-- 1. 支持事务 2. 支持外键 3. 支持行级锁

-- myisam 存储引擎
CREATE TABLE t28 (
    id INT,
    `name` VARCHAR(32)) ENGINE MYISAM;
-- 1. 添加速度快 2. 不支持外键和事务 3. 支持表级锁

START TRANSACTION;
SAVEPOINT t1;
INSERT INTO t28 VALUES(1, 'jack');
SELECT * FROM t28;
ROLLBACK TO t1; -- 没有回滚成功，jack这条记录还在

-- memory 存储引擎
-- 1. 数据存储在内存中[关闭了mysql服务，数据丢失，但是表结构还在]
-- 2. 执行速度很快（没有IO读写） 3. 默认支持索引

CREATE TABLE t29 (
    id INT,
    `name` VARCHAR(32)) ENGINE MEMORY

INSERT INTO t29
VALUES(1, 'tom'), (2, 'jack'), (3, 'hsp');

SELECT * FROM t29
DESC t29

-- 指令修改存储引擎
ALTER TABLE t29 ENGINE = INNODB
```

## 视图 (View)

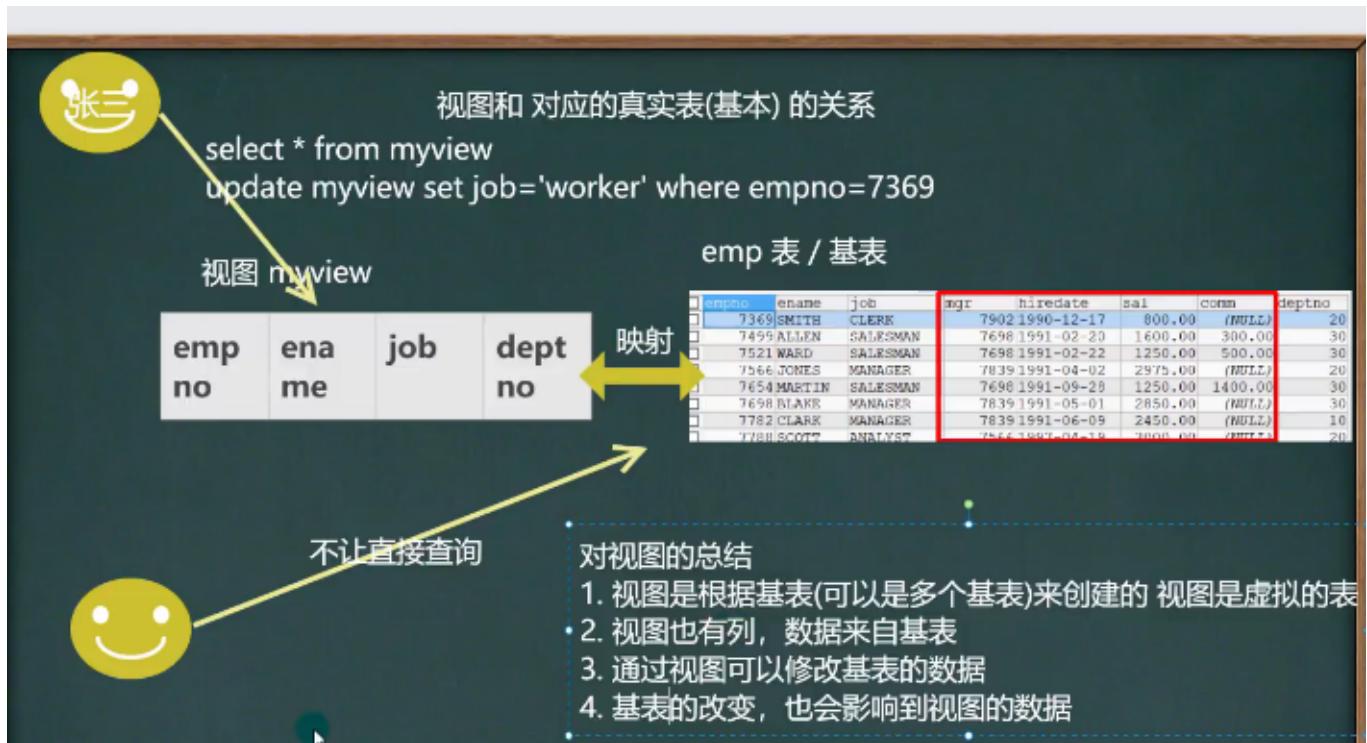


## 看一个需求，引出视图

emp表的列信息很多，有些信息是个人重要信息（比如sal, comm, mgr, hiredate），如果我们希望某个用户只能查询emp表的（empno、ename、job和deptno）信息，有什么办法？ => 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，试图包含列，其数据来自对应的真实表（基表）
2. 视图和基表关系的示意图



- 对视图的总结

1. 视图是根据基表来创建的，视图是虚拟的表
2. 视图也有列，数据来自基表
3. 通过视图可以修改基表的数据
4. 基表的改变，也会影响到视图的数据

- 视图的基本使用

1. create view 视图名 as select 语句
2. alter view 视图名 as select 语句
3. SHOW CREATE VIEW 视图名
4. drop view 视图名1, 视图名2

- 视图细节讨论

1. 创建视图后, 数据库保存的数据只有一个视图结构文件 (不会保存真正的数据)
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

- 视图最佳实践

1. 安全
2. 性能
3. 灵活

```
-- 视图的课堂练习
-- 针对 emp, dept 和 salgrade 三张表, 创建一个视图 emp_view03
-- 可以显示雇员编号, 雇员名, 雇员部门名称和薪水级别
```

```
/*
分析:
1. 使用三表联合查询, 得到结果
2. 将得到的结果, 构建成视图
*/
CREATE VIEW emp_view03
AS
SELECT empno, ename, dname, grade
FROM emp, dept, salgrade
WHERE emp.`deptno` = dept.`deptno` AND
(sal BETWEEN losal AND hisal);

DESC emp_view03
SELECT * FROM emp_view03
```

## Mysql管理

1. 创建新用户
2. 删除用户
3. 用户修改密码

```
-- Mysql用户的管理
-- 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操作权限
-- 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, 供开发人员使用

-- 1. 创建新的用户
```

```
-- 解读 (1) 'hsp_edu'@'localhost' 表示用户的完整信息 'hsp_edu' 用户名 'localhost'  
-- (2) 12345 密码, 但是注意 存放到mysql.user表时, 是password('123456')加密后的密码  
CREATE USER 'hsp_edu'@'localhost' IDENTIFIED BY '123456'  
  
SELECT PASSWORD('123456');  
  
SELECT `host`, `user`, authentication_string  
FROM mysql.`user`;  
  
-- 2. 删除用户: 用户名+登录地址  
DROP USER 'hsp_edu'@'localhost'  
  
-- 3. 登录
```

#### 4. 修改密码

```
1. 修改密码  
-- 修改自己的密码, 没问题  
  
SET PASSWORD = PASSWORD('abcdef');  
  
-- 修改其他人的密码, 需要权限  
  
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('123456');  
  
-- 2. 删除用户  
DROP USER 'hsp_edu'@'localhost'  
  
-- 3. 登录  
  
-- root 用户修改 hsp_edu@localhost 密码, 是可以成功的  
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('123456');
```

分析：登录到不同的用户看到的数据库是不一样的

## 权限管理

- 给用户授权

```
-- 演示 用户权限管理  
CREATE USER 'shunping'@'localhost' IDENTIFIED BY '123'  
  
CREATE DATABASE testdb  
CREATE TABLE news(  
    id INT,  
    content VARCHAR(32));  
  
-- 添加一条测试数据  
INSERT INTO news VALUES(100, '北京新闻');  
SELECT * FROM news;  
  
-- 给 shunping 分配查看 news 表和添加 news 的权限
```

```
GRANT SELECT, INSERT  
ON testdb.news  
TO 'shunping'@'localhost'  
  
SET PASSWORD FOR 'shunping'@'localhost' = PASSWORD('abc');  
  
-- 回收 shunping 用户在 testdb.news 表的所有权限  
REVOKE SELECT, UPDATE, INSERT ON testdb.news FROM 'shunping'@'localhost';  
REVOKE ALL ON testdb.news FROM 'shunping'@'localhost';  
  
-- 删除 shunping  
DROP USER 'shunping'@'localhost'
```

## 细节说明

1. 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可以登陆mysql
3. 在删除用户的时候，如果 host 不是 %，则需要明确指定 '用户'@'host值'

```
-- 直接create user jack 代表 host 为 %, 也就是所有的ip地址都能够连接上来  
CREATE USER jack  
  
SELECT `host`, `user` FROM mysql.user  
  
-- 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  
  
CREATE USER 'smith'@'192.168.1.%'  
  
-- 在删除用户的时候，如果 host 不是 %， 需要明确指定 '用户'@' host值'  
  
DROP USER jack -- 默认就是 DROP USER 'jack'@'%'  
  
DROP USER 'smith'@'192.168.1.%'
```

## homework

(1).以下哪条语句是错误的? []

A. SELECT empno,ename name,sal salary FROM emp;  
B. SELECT empno,ename name,sal AS salary FROM emp;  
C. SELECT ename,sal\*12 AS "Annual Salary" FROM emp;  
D. SELECT ename,sal\*12 |Annual Salary FROM emp;

分析: A答案中“ename name” ename后面跟着别名可以不加AS

D答案因为别名 Annual Salary中间有一个空格，因此错误

- 第二题:

```
SELECT *
FROM emp
WHERE (hiredate > '1990-02-01' AND hiredate < '1995-02-01')
ORDER BY hiredate
```

分析：日期要使用单引号"括起来

- 第三题：

-- 找出员工表中佣金大于薪金的人员名单

```
SELECT ename, comm, sal
FROM emp
WHERE IFNULL(comm, 0) > sal
```

分析：佣金comm可能为空，因此需要加上ifnull来将null化为0. ifnull (comm, 0) 如果comm不为空返回comm, 否则返回0

- 第四题：

-- 9. 找出每个月倒数第3天受雇的所有员工

-- 老韩提示：LAST\_DAY(日期)，可以返回该日期所在月份的最后一天

```
SELECT LAST_DAY('2011-11-11')
```

```
SELECT *
FROM emp
WHERE LAST_DAY(hiredate) - hiredate = 3
```

分析：使用 last\_day(datetime) 返回该月的最后一天

第五题：

-- 13 显示不带R的员工姓名

```
SELECT * FROM
emp
WHERE ename NOT LIKE '%R%'
```

分析：一个字符串是否存在哪个字符可以使用 like 来分析

第六题：

-- 将员工名称中的大写字母 A 替换为小写字母 a

```
SELECT REPLACE(ename, 'A', 'a')
FROM emp
```

分析：使用 replace 语句来替换字符串中的字符

第七题：

```
-- 查找入职时间大于10年的老员工
SELECT ename, hiredate FROM
    emp
WHERE DATE_ADD(hiredate, INTERVAL 10 YEAR) <= NOW()
```

分析：这里使用DATE\_ADD加上10年之后，需要小于现在的时间now ()，前面自己做的时候写成了10.

第八题：

```
-- 显示所有员工的姓名、工作和薪金，按照工作降序排列，如果工作相同，按照薪金排列
SELECT ename, job, sal
FROM emp
ORDER BY job DESC, sal
```

分析：这里排序使用工作和薪金，中间用逗号隔开，job后面填写DESC代表降序，如果不填写则按照默认的升序排列，order by job DESC, sal

第九题：

```
-- 25 以年月日的方式显示所有员工的服务年限
-- (这里年月日不是总和，也就是说不是总计有几个月，而是在算出年后，还剩下几个月)
SELECT FLOOR(DATEDIFF(NOW(), hiredate) / 365) AS '工作年',
       FLOOR(DATEDIFF(NOW(), hiredate) % 365 / 31) AS '工作月',
       FLOOR(DATEDIFF(NOW(), hiredate) % 31) AS 'day'
FROM emp
```

第十题：

```
-- 2. 查询薪水 sal 比 smith 多的员工
-- 思路：
-- (1) 先查 smith 的 sal 作为子查询
-- (2) 然后其它员工的 sal 大于 smith
SELECT *
FROM emp
WHERE sal > (
    SELECT sal
    FROM emp
    WHERE ename = 'SMITH')
```

分析：使用子查询，先查询出smith的薪水，再查询别人薪水大于smith的薪水

第十一题：

-- 11.列出在每个部门工作的员工数量、平均工资和平均服务期限

```
SELECT AVG(DATEDIFF(NOW(), hiredate))/365 FROM emp
```

```
SELECT COUNT(*), AVG(sal),
       FORMAT(AVG(DATEDIFF(NOW(), hiredate))/365, 2)
  FROM emp
 GROUP BY deptno
```

分析：可以使用format修改小数点后面的位数