MySQL 数据库

21 文件下载默认端口

80 web程序默认端口

初识

> 简介：MySQL 是最流行的关系型数据库管理系统，在 WEB 应用方面 MySQL 是最好的 RDBMS(Relational Database Management System：关系数据库管理系统)应用软件之一。

数据库

何为数据库

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库。

每个数据库都有一个或多个不同的 API 用于创建，访问，管理，搜索和复制所保存的数据。

我们也可以将数据存储在文件中，但是在文件中读写数据速度相对较慢。

所以，现在我们使用关系型数据库管理系统（RDBMS）来存储和管理大数据量。所谓的关系型数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。

RDBMS 即关系数据库管理系统(Relational Database Management System)的特点：

- 1.数据以表格的形式出现
- 2.每行为各种记录名称
- 3.每列为记录名称所对应的数据域
- 4.许多的行和列组成一张表单
- 5.若干的表单组成database

DB DBMS SQL

DB：DataBase（数据库，在电脑上以硬盘形式存在）

DBMS：DataBase Management System (数据库管理系统)

SQL 结构化查询语言（sql语句编译由DBMS完成）

DBMS -(执行) -> SQL -(操作) -> DB

表包括行和列

1. 行：被称为数据/记录（data）
2. 列：被称为字段（column）

字段应包含字段名、数据类型、相关的约束

SQL语句的分类

1. **DQL**（数据查询语言）：查询语句，select
2. **DML**（数据操作语言）：对表中的数据进行增删改 insert delete update
3. **DDL**（数据定义语言）：对表结构的增删改 create drop alter
4. **TCL**（事务控制语言）：commit提交事务、rollback回滚事务。（T：Transaction）
5. **DCL**（数据控制语言）：grant授权、revoke撤销权限等

数据库操作

1. 进入数据库
    1. `mysql -u root -p;`
    2. `mysql -h 主机名 -u 用户名 -p;`
2. 初始化数据库后修改密码：`alter user user() identified by "密码";`
3. 创建数据库：`create database 数据库名;`
4. 查看数据库：`show databases;`
5. 使用数据库：`use 数据库名;`
6. 查看数据库表：`show tables;`
7. 执行sql脚本：`source sql文件路径;`
8. 查看表结构：`desc 表名;`
9. 结束一条语句：`/c`
10. 退出：`exit`
11. 执行sql脚本：`source sql文件路径`
12. 查看创建表时的语句：`show create table 表名;`

sql脚本

批量执行sql命令可以使用source命令初始化

source sql文件路径

学习所使用练习表

数据库：

表名	描述
dept	部门表
emp	员工表
salgrade	工资等级表

部门表

Field	Type	Null	Key	Default	Extra	描述
DEPTNO	int	NO	PRI	NULL		部门编号
DNAME	varchar(14)	YES		NULL		部门名称
LOC	varchar(13)	YES		NULL		部门位置

员工表

Field	Type	Null	Key	Default	Extra	描述
EMPNO	int	NO	PRI	NULL		员工编号
ENAME	varchar(10)	YES		NULL		员工姓名
JOB	varchar(9)	YES		NULL		工作岗位
MGR	int	YES		NULL		上级领导编号
HIREDATE	date	YES		NULL		入职日期
SAL	double(7,2)	YES		NULL		月薪
COMM	double(7,2)	YES		NULL		补助/津贴
DEPTNO	int	YES		NULL		部门编号

工资等级表

Field	Type	Null	Key	Default	Extra	描述
GRADE	int	YES		NULL		等级
LOSAL	int	YES		NULL		最低薪资
HISAL	int	YES		NULL		最高薪资

查询语句DQL

简单查询语句

```
语法格式：select 字段名1,字段名2,字段名3,...from 表名;
```

sql语句不区分大小写

查询字段可以使用数学运算

```
select ename, sal * 12 from emp;
```

查询字段可以起别名

```
select ename, sal * 12 as yearSal from emp;
```

字符串必须使用单引号括起来（mysql支持双引号，但其他语言不支持），as可以省略

```
select ename, sal * 12 '年薪' from emp;
```

条件查询where

select 字段... from 表名 where 条件;

```
select ename from emp where sal = 5000;
```

找出高于平均工资的员工

```
select ename,sal from emp where sal > (select avg(sal) from emp);
```

查询指定范围的数据

```
select ename from emp where sal between 1100 and 3000
```

```
select ename from emp where sal >= 1100 and sal <= 3000
```

查询字符串

```
select ename from emp where ename between 'A' and 'C' //左闭右开
```

is null

```
select ename,sal,comm from emp where comm is null;
```

is not null

```
select ename,sal,comm from emp where comm is not null;
```

or

```
select ename,job from emp where job = 'clerk' or job = 'manager';
```

and和or的优先级问题

尽量使用小括号去决定运算符优先级

需求：找出薪资多于1000且部门编号是20或30部门的员工

```
select ename,sal,deptno from emp where sal > 1000 and (deptno = 20 or deptno =
30);
```

in

in如同on

```
select ename,job from emp where job in( 'clerk', 'manager');
```

not in

不在这几个值里面

```
select ename,job from emp where job not in( 'clerk', 'manager');
```

模糊查询like

1. %：任意多个字符

2. _：任意一个字符
3. \：转义字符

```
select ename from emp where ename like '%O%';
```

```
'%a'      //以a结尾的数据
'a%'      //以a开头的数据
'%a%'     //含有a的数据
'_a_'     //三位且中间字母是a的
'_a'      //两位且结尾字母是a的
'a_'      //两位且开头字母是a的
```

排序（升序，降序）order by

asc(升序)，desc(降序)

嵌套排序：order by 字段1 排序1, 字段2 排序2...; //越靠前优先级越高

按字段位置排，order by n; //n是第n个字段位置

案例： 对所有员工信息按照薪资降序排

```
select ename, sal from emp order by 2 asc;
```

案例：查询员工信息，先按工资降序排，同个工资的按升序排

```
select ename,sal from emp order by sal desc, ename asc;
```

select * 3
from tableName 1
where 条件 2
order by 排序条件 4

分组函数

分组函数也称多行处理函数

分组函数**自动忽略NULL**

分组函数不能出现在where语句中，▲原因：group by在where后面执行，而分组函数在group by后面执行，所以分组函数不能出现在where后面

1. count
   count(*)表示统计全部数据的数量,
   count(字段)统计该字段**不为null**的数量

2. sum
3. avg
4. max
5. min

分组函数就是对某一组数据进行操作

```
select max(sal) from emp;
```

单行处理函数

1. ifnull() 空处理函数

   语法：ifnull(可能为null的数据,被当作什么数据处理)

案例：计算所有员工的年总收入

```
select ename, (sal + ifnull(comm,0)) * 12 as yearSal from emp;
```

group by

简介：按照某个字段或者某些字段进行分组

找出每个岗位的最高薪资

```
select ename, max(sal) from emp group by job;
```

找出每个部门不同岗位的最高薪资

```
select max(sal),deptno,job from emp group by deptno,job;
```

having

简介：having是对分组之后的数据进行再次过滤

案例：找出每个部门的最高薪资，要求显示薪资大于2500的数据

1. 第一步： 找出每个部门最高薪资：select max(sal), deptno from emp group by deptno;
2. 第二步：显示薪资大于2500的薪资：select max(sal), deptno from emp group by deptno having max(sal) > 2500;

第二种方式，效率更高，先在分组前筛选数据

```
select max(sal) ,deptno from emp where sal > 2500 group by deptno;
```

案例：找出每个部门的平均薪资，要求薪资大于2000（计算过后的数据不能使用where）

```
select avg(sal) , deptno from emp group by deptno having avg(sal) > 2000;
```

查询结果集的去重distinct

```
select distinct job from emp; //distinct
```

distinct只能出现在所有字段的最前方，若有distinct后面有多个字段，就会对多字段联合去重

统计一共有多少个岗位

```
select count(distinct job) from emp;
```

完整的DQL语句

select 5
..
from 1
..
where 2
..
group by 3
..
having 4
..
order by 6
..

连接查询

> 在实际开发中，在一般的业务情况下，是多表联合查询

分类

1. 内连接
    1. 等值连接
    2. 非等值连接
    3. 自连接
2. 外连接

    1. 左外连接（左连接）
    2. 右外连接（右连接）
3. 全连接

笛卡尔积现象（笛卡尔乘积现象）

案例：找出每一个员工的部门名称，要求显示员工名和部门名(92语法)

`select ename, dname from emp, dept;`：会产生笛卡尔积现象，两个表数量的乘积

表的别名（设置别名，防止不同表同名现象）

```
select e.ename, d.dname from emp e, dept d;
```

正确做法(92语法)

```
select e.ename, d.dname from emp e, dept d where e.deptno = d.deptno;
```

等值连接

> 条件为等量关系

案例：找出每一个员工的部门名称，要求显示员工名和部门名(99语法)

inner可省略，代表内连接

```
select e.ename,d.dname from emp e inner join dept d on e.deptno = d.deptno;
```

非等值连接

> 连接条件中的关系是非等量关系

案例： 找出每个员工工资所属等级

```
select e.ename,e.sal,s.grade from emp e join salgrade s on e.sal between s.losal
and s.hisal;
```

自连接

> 最大的特点：一张表看作两张表，自己链接自己
>
> 假设a、b表进行连接，使用内连接的话，凡是a表和b表能够匹配上的记录查询出来，便是内连接，a、b表无主副之分，两表平等

案例：找出每个员工的上级领导

```
select e.ename as '员工姓名',b.ename as '领导者' from emp e join emp b on e.mgr =
b.empno;
```

外连接

> 假设a表和b表进行连接，使用外连接时，a、b表中有一张表是主表，另一张表是副表，主要查询主表中的数据，捎带查询附表，当附表中的数据没有和主表匹配上时，附表自动模拟出null与之匹配

outer可省略，代表外连接

查询所有员工的领导（即使没有也要返回null）

```
select e.ename as '员工姓名' , b.ename as '领导者' from emp e left outer join emp b
on e.mgr = b.empno;
```

左外连接

> 表示左边是主表

```
select e.ename as '员工姓名' , b.ename as '领导者' from emp e left outer join emp b
on e.mgr = b.empno;
```

右外连接

> 表示右边是主表

```
select e.ename as '员工姓名' , b.ename as '领导者' from emp e right join emp b on
e.mgr = b.empno;
```

案例：查询没有员工的部门编号

```
select d.* from emp e right join dept d on e.deptno = d.deptno where e.deptno is
null;
```

三表联合查询

案例：找出所有员工的部门名称和工资等级

```
select e.ename, d.dname, s.grade from emp e join dept d on e.deptno = d.deptno
join salgrade s on e.sal between s.losal and s.hisal;
```

案例：找出所有员工的部门名称、工资等级和上级领导

```
select e.ename as '员工', e1.ename as '领导', d.dname, s.grade from emp e join dept
d on e.deptno = d.deptno join salgrade s on e.sal between s.losal and s.hisal left
join emp e1 on e.mgr = e1.empno;
```

子查询

> select语句嵌套select语句，被嵌套的select语句是子查询

select
  ...(select)
from
  ...(select)
where
  ...(select)

where

案例：找出高于平均薪资的员工信息

1. 找出平均薪资 `select avg(sal) from emp`
2. 找出高于平均薪资的员工信息表 `select ename, sal from emp where sal > 平均薪资`

```
 select ename, sal from emp where sal > (select avg(sal) from emp);
```

from

案例：找出每个部门平均薪水的薪资等级

1. 找出每个部门的平均薪资：`select deptno, avg(sal) as 'avgsal' from emp group by deptno;`
2. 找出第一步所得出的薪资等级：`select t.*, s.grade from (第一步结果) t join salgrade s on t.avgsal between s.losal and s.hisal;`

```
select t.*, s.grade from (select deptno, avg(sal) as 'avgsal' from emp group by
deptno) t join salgrade s on t.avgsal between s.losal and s.hisal;
```

案例：找出每个部门平均的薪水等级

1. 第一步：求出各个员工所属部门的的工资水平 `select e.sal, e.deptno, s.grade from emp e join salgrade s on e.sal between s.losal and s.hisal`
2. 第二步：基于第一步求每个部门的平均薪资

```
select e.deptno, avg(s.grade) from emp e join salgrade s on e.sal between s.losal
and s.hisal group by e.deptno;
```

select

案例：查询员工，并查询员工所属的部门名称

1. 第一步：查询员工的基本信息：`select e.ename, e.deptno from emp e;`
2. 第二步：查询部门编号所属部门名称：`select d.dname from dept d where e.deptno = d.deptno`

```
select e.ename, (select d.dname from dept d where e.deptno = d.deptno) as '部门名'
from emp e;
```

union

> 将查询结果进行相加

案例：找出工作岗位是saleman和manager的员工

1. `select ename, job from emp where job = 'salesman' or job = 'manager';`
2. `select ename, job from emp where job in ('salesman', 'manager');`

union用法

```
select ename, job from emp where job = 'salesman' union select ename, job from emp
where job = 'manager';
```

limit

> limit是mysql有的，常应用于分页查询，可以取结果数据集的部分数据

语法：`limit startIndex, length`

1. startIndex表示其实位置
2. length表示取的数量

limit是sql语句最后执行的一个环节

案例：找出工资前五的员工

```
select ename, sal from emp order by sal desc limit 0, 5;
```

通用标准分页sql：`select * from tableName limit (pageNo - 1) * pageSize, pageSize;`

表操作

创建表

建表语法格式：

```
create table 表名 (
字段名1 数据类型(数据长度),
字段名2 数据类型(数据长度),
字段名3 数据类型(数据长度),
)
```

数据类型

name	Type
int	整数型(java int)
bigint	长整型（java long）
float	浮点型(float double)
char	定长字符串(String)
varchar	可变长字符串(StringBuffer/StringBuilder)
date	日期类型（java Date）
BLOB	二进制大对象（存储图片、视频等流媒体信息）binary large object
CLOB	字符大对象（存储较大文本，比如可以存储4G字符串）Character Large Object

案例：创建一个学生表t_student

```
create table t_student(
    no bigint,
    name varchar(255),
    classno varchar(255),
    sex char(1),
    birth date
);
```

插入数据

> 插入数据

语法格式：insert into 表名（字段名1，字段名2，字段名3...）values（值1，值2，值3，...）；

1. 插入单条数据

```
insert into t_student (name, classno, birth, sex) values('zhangsan',501,
'2000-01-01', 1);
```

2. 插入多条数据

```
insert into t_student (no, name, classno, sex, birth) values (1,'wangwu',
502, 1, '1999-01-01'), (2, 'xiaoli',503, 0 , '2001-01-01');
```

3. 插入全部数据可以省略字段名

```
insert into t_student values (1,'wangwu', 502, 1, '1999-01-01');
```

表复制

1. 将查询结果创建新表

```
create table 表名 as select 语句;
```

2. 将查询结果插入指定表

```
insert into 表名 select 语句;
```

修改数据

1. 更新单条记录

```
update 表名 set 字段名1=值1，字段名2=值2... where 条件；
```

2. 更新所有记录

```
update 表名 set 字段名1=值1，字段名2=值2... ；
```

删除数据

语法格式：`delete from 表名 where 条件;`

没有条件全部删除

删除大容量数据的表 `truncate table 表名` //表被截断，不可回滚，永久丢失

删除表：drop table if exists 表名

表结构修改

表结构的修改一般使用工具进行

但对于表数据的增删改查，则需要掌握sql语句，因为实际开发是在java进行的

约束（Constraint）

1. 非空 not null
2. 唯一 unique
3. 主键 primary key
4. 外键 foreign key
5. 检查 check

```
create table t_user(
    no bigint,
    name varchar(255) not null,
    sex char(1),
    birth date
);
```

唯一性 unique

> 唯一约束修饰的字段具有唯一性，不能重复，但可以为null

单字段添加约束（**列级约束**）

```
create table 表名（
    字段名 字段类型 unique，
）
```

多个字段联合添加约束（**表级约束**）

```
create table 表名 （
    字段名1 字段类型1，
    字段名2 字段类型2，
    unique(字段名1， 字段名2)
）
```

主键 primary key

> 主键约束 primary key， 主键即不能为空也必须唯一，一张表的主键约束只能有一个

作用：表的设计三范式的要求，第一范式要求所有表都必须有主键

主键分类

1. 单一主键

```
create table 表名(
    id int primary key,
    username varchar(255)
);
```

2. 复合主键（多个字段联合形成的主键，不推荐使用，不符合三范式）

```
create table 表名(
    id int,
    username varchar(255),
    primary key(id, username)
);
```

主键性质划分

1. 自然主键：和业务无任何关系的自然数
2. 业务主键：主键值和业务进行挂钩，如身份证号、银行卡号（但不建议使用，因为可能因业务改变而引起主键跟着变）

主键值自增：auto_increment

```
create table 表名(
    id int primary key auto_increment,
    username varchar(255)
)
```

外键约束 foreign key

> 外键约束：用来**关联另一张表**的字段约束，外键**可以为null**，引用的其他表字段**非必须**为主键，但**必须为一个具有唯一性的字段**

删除表格时，应先删除子表，再删除父表

创建表时，先创建父表再创建子表

添加数据，应先添加父表，再添加子表

```
create table t_class(
    cname varchar(255),
    id int primary key
)

create table t_student(
    sname varchar(255),
    id int primary key,
    classno int,
    foreign key(classno) references t_class(id)
)

insert into t_class values("高一一班",1);
insert into t_class values("高一二班",2);
insert into t_class values("高一三班",3);

insert into t_student values("张三", 1, 1);
insert into t_student values("李四", 2, 2);
insert into t_student values("王五", 3, 3);
insert into t_student values("赵六", 4, 3);
insert into t_student values("你好", 5, 1);

select * from t_class;
select * from t_student;
```

存储引擎(待补充)

> 数据库中的各表均被指定的存储引擎处理
>
> 服务器可用的引擎依赖于mysql版本等

> mysql支持很多种存储引擎，不同种存储引擎对于不同的存储方式，各有优缺点

emp完整的建表语句

```
| emp   | CREATE TABLE `emp` (
  `EMPNO` int NOT NULL,
  `ENAME` varchar(10) DEFAULT NULL,
  `JOB` varchar(9) DEFAULT NULL,
  `MGR` int DEFAULT NULL,
  `HIREDATE` date DEFAULT NULL,
  `SAL` double(7,2) DEFAULT NULL,
  `COMM` double(7,2) DEFAULT NULL,
  `DEPTNO` int DEFAULT NULL,
  PRIMARY KEY (`EMPNO`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 |
```

其中：`ENGINE=InnoDB DEFAULT` 是存储引擎版本

`CHARSET=utf8mb3` 是字符集

查看当前mysql版本支持的存储引擎：`show engines \G`

MyISAM存储引擎

innoDB存储引擎

> 该存储引擎数据的安全可以得到保证，支持事务、行级锁、外键等

表的数据结构存储在xxx.frm文件中

数据存储在tablespace这样的表空间中（逻辑概念），无法被压缩，无法转换为只读

提供MySQL数据库奔溃后的自动恢复

支持级联删除和级联更新

MEMORY存储引擎

> 不支持事务，数据易丢失（数据和索引都存储于内存中），查询速度极快

事务

> 一个事务是一个完整的业务逻辑单元，不可再分

> 要想保证两条DML语句同时成功或同时失败，不能出现两条结果不一的情况，则需要使用数据库的"事务机制"

比如，a账户向b账户转账1000，则在数据库层面必会产生两条update语句

```
update account set balance = balance - 100 where actno = 'a';
update account set balance = balance + 100 where actno = 'b';
```

事务的存在为了保证数据的安全性和完整性

DML语句：

1. insert
2. delete
3. update

事务的特性ACDI

1. A：原子性，事务是最小的工作单元，不可再分
2. C：一致性：事务必须保证多条dml语句同时成功或者同时失败
3. I：隔离性：事务A和事务B之间具有隔离
4. D：持久性：持久性说的是最终必须持久化到硬盘文件中，事务才算成功的结束

MySQL相关操作

1. 提交事务 `commit`
2. 回滚事务 `rollback`

隔离性

事务隔离性存在隔离级别，理论上隔离级别包括4个:

1. 第一级别:读未提交(read uncommitted)
   对方事务还没有提交，我们当前事务可以读取到对方未提交的数据。
   读未提交存在脏读(Dirty Read)现象:表示读到了脏的数据。
2. 第二级别:读已提交(read comnitted)
   对方事务提交之后的数据我方可以读取到。
   这种隔离级别解决了:脏读现象没有了。
   读已提交存在的问题是:不可重复读。
3. 第三级别:可重复读(repeatable read)
   这种隔离级别解决了:不可重复读问题。
   这种隔离级别存在的问题是:读取到的数据是幻象。
4. 第四级别:序列化读/串行化读
   解决了所有问题。
   效率低。需要事务排队

oracle数据库默认的隔离级别是:读已提交，mysql数据库默认的隔离级别是:可重复读。

MySQL事务默认下是自动提交的，关闭自动提交： `start transaction;`

test

默认情况下的回滚

```
mysql> insert into t_user values(2,'lisi');
Query OK, 1 row affected (0.01 sec)

mysql> insert into t_user values(3,'wanwu');
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
+----+----------+
3 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
+----+----------+
3 rows in set (0.00 sec)
```

关闭自动提交后的回滚与提交

回滚

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t_user values(4,'zhaoliu');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_user values(5,'hahhaa');
Query OK, 1 row affected (0.00 sec)

mysql> insert into t_user values(6,'ttt');
Query OK, 1 row affected (0.00 sec)

mysql> select * from t_user;
```

```
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
|  4 | zhaoliu  |
|  5 | hahhaa   |
|  6 | ttt      |
+----+----------+
6 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from t_user;
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
+----+----------+
3 rows in set (0.00 sec)
```

提交

```
mysql> insert into t_user values(4,'zhaoliu');
Query OK, 1 row affected (0.01 sec)

mysql> insert into t_user values(5,'hahhaa');
Query OK, 1 row affected (0.01 sec)

mysql> insert into t_user values(6,'ttt');
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_user;
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
|  4 | zhaoliu  |
|  5 | hahhaa   |
|  6 | ttt      |
+----+----------+
6 rows in set (0.00 sec)

mysql> commit;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
|  4 | zhaoliu  |
|  5 | hahhaa   |
|  6 | ttt      |
+----+----------+
6 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_user;
+----+----------+
| id | username |
+----+----------+
|  1 | zhangsan |
|  2 | lisi     |
|  3 | wanwu    |
|  4 | zhaoliu  |
|  5 | hahhaa   |
|  6 | ttt      |
+----+----------+
6 rows in set (0.00 sec)
```

查看当前事务隔离级别：`select @@transaction_isolation;`

查看全局事务隔离级别：`select @@global.transaction_isolation;`

设置事务全局隔离级别：`set global transaction isolation level  等级;`

1. read uncommitted

```
mysql> select @@global.tx_isolation;
ERROR 1193 (HY000): Unknown system variable 'tx_isolation'
mysql> Unknown system variable 'tx_isolation'^C
mysql> select @@global.transaction_isolation;
+--------------------------------+
| @@global.transaction_isolation |
+--------------------------------+
| READ-UNCOMMITTED               |
+--------------------------------+
```

2. read committed

```
mysql> set global transaction isolation level read committed;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@global.transaction_isolation;
+--------------------------------+
| @@global.transaction_isolation |
+--------------------------------+
| READ-COMMITTED                 |
+--------------------------------+
1 row in set (0.00 sec)
```

3. repeatable read

```
mysql> set global transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@global.transaction_isolation;
+--------------------------------+
| @@global.transaction_isolation |
+--------------------------------+
| REPEATABLE-READ                |
+--------------------------------+
1 row in set (0.00 sec)
```

4. serializable

```
mysql> set global transaction isolation level serializable;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@global.transaction_isolation;
+--------------------------------+
| @@global.transaction_isolation |
+--------------------------------+
| SERIALIZABLE                   |
+--------------------------------+
1 row in set (0.00 sec)
```

索引

> 索引相当于书的目录，可以通过目录快速查找对应的资源。使用索引可快速访问数据库表中的特定信息。索引是对数据库表中一列或多列的值进行排序的一种结构。

数据库中查询一张表有两种检索方式

1. 全表扫描
2. 根据索引检索（效率高）

索引原理：缩小扫描范围

缺点：需要维护，如定义索引的字段被修改，则该索引也需要对应的修改维护

适用范围

1. 数据量庞大（根据客户需求，根据线上情况）
2. 该字段很少DML操作（因为字段进行修改操作，索引也需要维护）
3. 该字段经常出现在where子句中

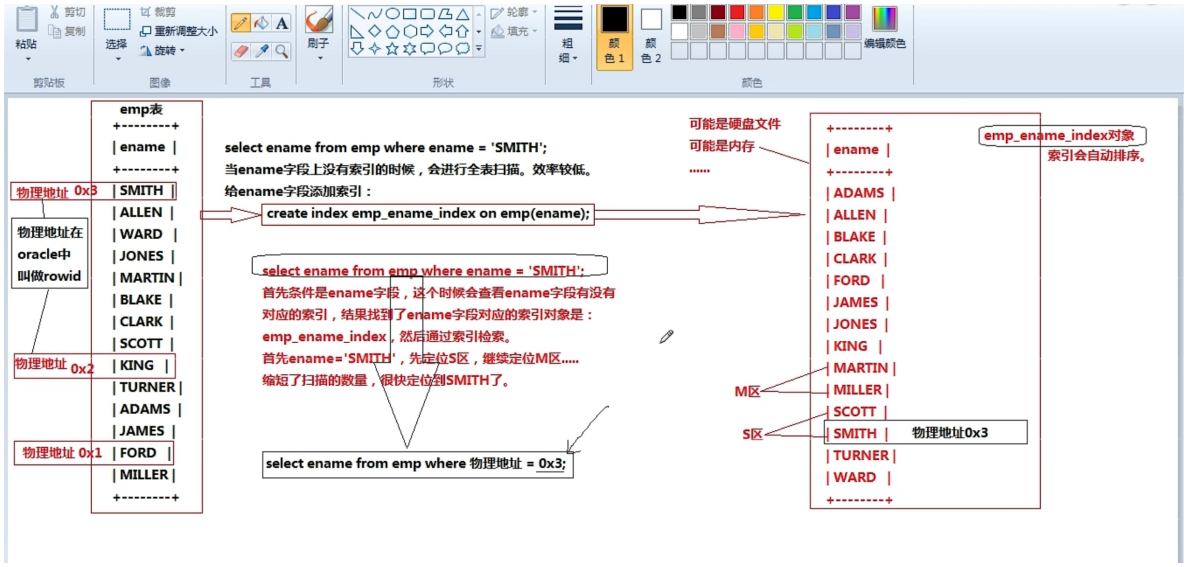
查看sql语句的执行计划 `explain`

```
explain select ename ,sal from emp where sal = 5000;
```

添加索引：`create index 索引名 on 表名(字段名);`

删除索引：`drop index 索引名 on 表名;`

原理



底层采用的数据结构：B + Tree

通过B TREE 缩小扫描范围， 底层索引进行排序分区，索引会携带数据在表中的”物理地址“，最终用过索引检索到数据后，获取关联的物理地址，通过物理地址定位表中的数据

如`select ename from emp where ename = 'smith';`

通过索引转化为`select ename from emp where 物理地址 = '0x3...';`

分类

1. 单一索引：给单个字段添加索引
2. 符合索引：多个字段联合起来添加一个索引
3. 主键索引：主键上自动添加索引
4. 唯一索引：有unique约束的字段会自动添加索引

索引失效地方：模糊查询的时候，当第一个通配符使用的是%，这时候索引是失效的 `select ename from emp where ename like "%A%";`

视图view

> 位于不同角度去看待数据，同一张表的数据，通过不同角度看待

创建视图： `create view 视图名 as select语句;`

删除视图： `drop view 视图名;`

作用：视图可以隐藏表的实现细节， 保密级别较高的系统，数据库只对外提供相关视图，程序员只对视图对象进行crud

DBA命令

数据库的导入导出

导出： `mysqldump 表名>导出路径 -uroot -p密码` （在DOC命令窗口，不是在mysql中进行）

可以具体到某个表： `mysqldump 表 具体表>导出路径 -uroot -p密码`

导入： `source sql文件路径`

数据库设计三范式

> 设计范式：设计表的依据，按照该三范式设计的表不会出现数据冗余

第一范式：任何一张表都有主键，并且每一个字段原子性不可再分

第二范式：建立在第一范式的基础之上，所有非主键字段完全依赖主键，不能产生部分依赖

多对多关系：三张表，关系表两外键

第三范式：建立在第二范式基础之上，所有非主键字段直接依赖主键，不能产生传递依赖

一对多关系：两张表，多的表加外键

一对一关系：主键共享或外键唯一

在实际开发中：根据业务需求可能需要用冗余换执行速度

练习

1. 取得每个部门最高薪水的人员名称

```
select e.ename, t.maxsal, t.deptno from (select max(sal) as maxsal, deptno
from emp group by deptno) t join emp e on t.deptno = e.deptno and t.maxsal =
e.sal;
```