- 数据库
- SQL：结构化查询语言
  - 关于SQL语句的分类
- MySQL常用命令
- 数据库表
- 查询：select
  - 简单查询
  - 条件查询
  - 分组查询
- 排序：order by
  - 分组函数（多行处理函数）
  - distinct
- 连接查询
  - 内连接：等值连接
  - 内连接：非等值连接
  - 内连接：自连接
  - 外连接
- 子查询
- union合并
- limit
- 表的创建
- 插入数据insert
- 修改update
- 删除数据 delete
- 约束
  - 非空约束：not null
  - 唯一性约束: unique
  - 主键约束: primary key
  - 外键约束：foreign key
- 存储引擎
  - MyISAM存储引擎
  - InnoDB存储引擎
  - MEMORY存储引擎
- 事务
  - 提交事务、回滚事务
  - 事务四个特性
  - 事务的隔离性
- 索引（index）
  - 实现原理
  - 添加索引的条件
  - 索引的创建和删除
  - 是否使用索引进行检索
  - 索引失效的情况
  - 索引分类
- 视图
  - 视图创建和删除

    - 视图的用途
- DBA常用命令
- 数据库设计三范式
- 总结表的设计
- 嘱咐

数据库

> 什么是数据库？什么是数据库管理系统？什么是SQL？他们之间的关系是什么？

```
数据库：
    英文单词DataBase，简称DB。按照一定格式存储数据的一些文件的组合。
    顾名思义：存储数据的仓库，实际上就是一堆文件。这些文件中存储了具有特定格式的数据。
数据库管理系统：
        DataBaseManagement，简称DBMS。
        数据库管理系统是专门用来管理数据库中数据的，数据库管理系统可以对数据库当中的数据进行增删改查。
常见的数据库管理系统：
        MySQL、Oracle、MS SqlServer、DB2、sybase等....
```

SQL：结构化查询语言

```
    程序员需要学习SQL语句，程序员通过编写SQL语句，然后DBMS负责执行SQL语句，最终来完成数据库中数据的增删改查操作。
    SQL是一套标准，程序员主要学习的就是SQL语句，这个SQL在mysql中可以使用，
    同时在Oracle中也可以使用，在DB2中也可以使用。

    三者之间的关系？
        DBMS--执行--> SQL --操作--> DB

    先安装数据库管理系统MySQL，然后学习SQL语句怎么写，编写SQL语句之后，DBMS
    对SQL语句进行执行，最终来完成数据库的数据管理。
```

关于SQL语句的分类

```
SQL语句有很多，最好进行分门别类，这样更容易记忆。
    分为：
    DQL：
        数据查询语言（凡是带有select关键字的都是查询语句）
        select...

    DML：
        数据操作语言（凡是对表当中的数据进行增删改的都是DML）
        insert delete update
```

```
        insert 增
        delete 删
        update 改
        这个主要是操作表中的数据data。

    DDL:
        数据定义语言
        凡是带有create、drop、alter的都是DDL。
        DDL主要操作的是表的结构。不是表中的数据。
        create:新建,等同于增
        drop:删除
        alter:修改
        这个增删改和DML不同,这个主要是对表结构进行操作。

    TCL:
        是事务控制语言
        包括:
        事务提交:commit;
        事务回滚:rollback;

    DCL:
        是数据控制语言。
        例如:授权grant、撤销权限revoke....
```

MySQL常用命令

```
退出mysql : exit;

查看mysql中有哪些数据库 : show databases;  #注意:以分号结尾,分号是英文的分号。
mysql> show databases;
    +--------------------+
    | Database           |
    +--------------------+
    | information_schema |
    | mysql              |
    | performance_schema |
    | test               |
    +--------------------+
    mysql默认自带了4个数据库。

选择使用某个数据库:
mysql> use test;
    Database changed
    表示正在使用一个名字叫做test的数据库。

创建数据库:
mysql> create database bjpowernode;
    Query OK, 1 row affected (0.00 sec)
mysql> show databases;
        +--------------------+
```

```
        | Database           |
        +--------------------+
        | information_schema |
        | bjpowernode        |
        | mysql              |
        | performance_schema |
        | test               |
        +--------------------+
```

查看某个数据库下有哪些表：

```
mysql> show tables;
```

查看mysql数据库的版本号：

```
mysql> select version();
        +-----------+
        | version() |
        +-----------+
        | 5.5.36    |
        +-----------+
```

查看当前使用的是哪个数据库：

```
mysql> select database();
    +-------------+
    | database()  |
    +-------------+
    | bjpowernode |
    +-------------+
```

导入一下提前准备好的数据：bjpowernode.sql 这个文件中是练习准备的数据库表。
怎么将sql文件中的数据导入

```
mysql> source D:\course\03-MySQL\document\bjpowernode.sql #注意：路径中不要有中文！！！！
```

查看表的结构：

```
mysql> desc dept; # describe缩写为：desc
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| DEPTNO | int(2)      | NO   | PRI | NULL    |       |部门编号
| DNAME  | varchar(14) | YES  |     | NULL    |       |部门名字
| LOC    | varchar(13) | YES  |     | NULL    |       |地理位置
+--------+-------------+------+-----+---------+-------+
```

数据库表

数据库当中最基本的单元是表：table

什么是表table？为什么用表来存储数据呢？

```
        姓名     性别     年龄(列：字段)
        ---------------------------
        张三     男          20           ------->行（记录）
        李四     女          21           ------->行（记录）
        王五     男          22           ------->行（记录）
```

数据库当中是以表格的形式表示数据的。因为表比较直观。

任何一张表都有行和列：
行（row）：被称为数据/记录。
列（column）：被称为字段。

姓名字段、性别字段、年龄字段。

了解一下：
每一个字段都有：字段名、数据类型、约束等属性。字段名可以理解，是一个普通的名字，见名知意就行。
数据类型：字符串，数字，日期等，后期讲。
约束：约束也有很多，其中一个叫做唯一性约束，这种约束添加之后，该字段中的数据不能重复。

查询：select

简单查询

查询一个字段?
```
select 字段名 from 表名;
# 其中要注意：select和from都是关键字。
# 字段名和表名都是标识符。
```

强调：对于SQL语句来说，是通用的，所有的SQL语句以“;”结尾。
另外SQL语句不区分大小写，都行。

查询两个字段，或者多个字段使用逗号隔开“,”
查询部门编号和部门名
```
select deptno,dname from dept;
```
```
    +--------+------------+
    | deptno | dname      |
    +--------+------------+
    |     10 | ACCOUNTING |
    |     20 | RESEARCH   |
    |     30 | SALES      |
    |     40 | OPERATIONS |
    +--------+------------+
```

查询所有字段
第一种方式：可以把每个字段都写上
```
    select a,b,c,d,e,f... from tablename;
```

第二种方式：可以使用*

```
select * from dept;
    +--------+------------+----------+
    | DEPTNO | DNAME      | LOC      |
    +--------+------------+----------+
    |     10 | ACCOUNTING | NEW YORK |
    |     20 | RESEARCH   | DALLAS   |
    |     30 | SALES      | CHICAGO  |
    |     40 | OPERATIONS | BOSTON   |
    +--------+------------+----------+
```

这种方式的缺点：
1、效率低
2、可读性差。
在实际开发中不建议，可以自己玩没问题。
你可以在DOS命令窗口中想快速的看一看全表数据可以采用这种方式。

给查询的列起别名：使用as关键字起别名。

```
mysql> select deptno,dname as deptname from dept;
    +--------+------------+
    | deptno | deptname   |
    +--------+------------+
    |     10 | ACCOUNTING |
    |     20 | RESEARCH   |
    |     30 | SALES      |
    |     40 | OPERATIONS |
    +--------+------------+
```

注意：只是将显示的查询结果列名显示为deptname，原表列名还是叫：dname
记住：select语句是永远都不会进行修改操作的。（因为只负责查询）

as关键字可以省略吗？可以的

```
mysql> select deptno,dname deptname from dept;
```

假设起别名的时候，别名里面有空格，怎么办？

```
select deptno,dname 'dept name' from dept; //加单引号
select deptno,dname "dept name" from dept; //加双引号
    +--------+------------+
    | deptno | dept name  |
    +--------+------------+
    |     10 | ACCOUNTING |
    |     20 | RESEARCH   |
    |     30 | SALES      |
    |     40 | OPERATIONS |
    +--------+------------+
```

注意：在所有的数据库当中，字符串统一使用单引号括起来，单引号是标准，双引号在oracle数据库中用不了。但是在mysql中可以使用。

条件查询

条件查询：不是将表中所有数据都查出来。是查询出来符合条件的。

查询语法格式:

```
select
  字段1,字段2,字段3....
from
  表名
where
  条件;
```

都有哪些条件?

= 等于

查询薪资等于800的员工姓名和编号?

```
select empno,ename from emp where sal = 800;
```

查询SMITH的编号和薪资?

```
select empno,sal from emp where ename = 'SMITH'; //字符串使用单引号
```

<> 或!= 不等于

查询薪资不等于800的员工姓名和编号?

```
select empno,ename from emp where sal != 800;
select empno,ename from emp where sal <> 800; // 小于号和大于号组成的不等号
```

< 小于

查询薪资小于2000的员工姓名和编号?

```
mysql> select empno,ename,sal from emp where sal < 2000;
+-------+--------+---------+
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7499 | ALLEN  | 1600.00 |
|  7521 | WARD   | 1250.00 |
+-------+--------+---------+
```

<= 小于等于

查询薪资小于等于3000的员工姓名和编号?

```
select empno,ename,sal from emp where sal <= 3000;
```

> 大于

查询薪资大于3000的员工姓名和编号?

```
select empno,ename,sal from emp where sal > 3000;
```

>= 大于等于

查询薪资大于等于3000的员工姓名和编号?

```
select empno,ename,sal from emp where sal >= 3000;
```

between … and …. 两个值之间, 等同于 >= and <=

查询薪资在2450和3000之间的员工信息? 包括2450和3000

第一种方式: >= and <= (and是并且的意思。)

```
select empno,ename,sal from emp where sal >= 2450 and sal <= 3000;
+-------+-------+---------+
| empno | ename | sal     |
+-------+-------+---------+
|  7566 | JONES | 2975.00 |
```

```
    |  7698 | BLAKE | 2850.00 |
    |  7782 | CLARK | 2450.00 |
    |  7788 | SCOTT | 3000.00 |
    |  7902 | FORD  | 3000.00 |
    +-------+-------+---------+
  第二种方式：between … and …
    select
      empno,ename,sal
    from
      emp
    where
      sal between 2450 and 3000;

    注意：
      使用between and的时候，必须遵循左小右大。
      between and是闭区间，包括两端的值。

is null 为 null (is not null 不为空)
注意：在数据库当中null不能使用等号进行衡量。需要使用is null因为数据库中的null代表什么也没有，它不是一个值，所以不能使用等号衡量。

and 并且

or 或者
select empno,ename,job from emp where job='MANAGER' or job='SALESMAN';

and和or同时出现的话，有优先级问题吗?
and和or同时出现，and优先级较高。如果想让or先执行，需要加“小括号”.以后在开发中，如果不确定优先级，就加小括号就行了。

in 包含，相当于多个 or （not in 不在这个范围中）
select ename,sal from emp where sal in(800,5000);

not 可以取非，主要用在 is 或 in 中
select ename,sal from emp where sal not in(800,5000,3000);

like 称为模糊查询，支持%或下划线匹配
# %匹配任意多个字符
# 下划线：任意一个字符。
# (%是一个特殊的符号，_ 也是一个特殊符号)

找出名字以T结尾的?
  select ename from emp where ename like '%T';

找出名字以K开始的?
  select ename from emp where ename like 'K%';

找出第二个字每是A的?
  select ename from emp where ename like '_A%';

找出第三个字母是R的?
  select ename from emp where ename like '__R%';
```

分组查询

> 什么是分组查询?

- 在实际的应用中，可能有这样的需求，需要先进行分组，然后对每一组的数据进行操作，这个时候我们需要使用分组查询。

```
语法格式:
    select
      ...
    from
      ...
    group by
      ...

注意:
select ename,job,sum(sal) from emp group by job;
    +-------+-----------+----------+
    | ename | job       | sum(sal) |
    +-------+-----------+----------+
    | SCOTT | ANALYST   |  6000.00 |
    | SMITH | CLERK     |  4150.00 |
    | JONES | MANAGER   |  8275.00 |
    | KING  | PRESIDENT |  5000.00 |
    | ALLEN | SALESMAN  |  5600.00 |
    +-------+-----------+----------+
以上语句在mysql中可以执行，但select后面添加ename字段没有意义。
以上语句在oracle中执行报错。
oracle的语法比mysql的语法严格。（mysql的语法相对来说松散一些！）

重点结论：在一条select语句当中，如果有group by语句的话，select后面只能跟：参加分组的字段，以及分组函数。其它的一律不能跟。

找出“每个部门，不同工作岗位”的最高薪资?
select
  deptno, job, max(sal)
from
  emp
group by
  deptno, job;
技巧：两个字段联合成1个字段看。（两个字段联合分组）

使用having可以对分完组之后的数据进一步过滤。
having不能单独使用，having不能代替where，having必须和group by联合使用。

where和having，优先选择where，where实在完成不了了，再选择having。
```

> 补充一点

```
将之前的关键字全部组合在一起，来看一下他们的执行顺序?
    select
        ...
    from
        ...
    where
        ...
    group by
         ...
    having
        ...
    order by
        ...
以上关键字的顺序不能颠倒，需要记忆。
执行顺序是什么?
  1. from
  2. where
  3. group by
  4. having
  5. select
  6. order by
```

排序：order by

```
语法格式:
    select
        ename,sal
    from
        emp
    order by
        sal; // 默认是升序!!
```

指定降序： desc 指定升序： asc

以两个字段排序吗？或者说按照多个字段排序？ 查询员工名字和薪资，要求按照薪资升序，如果薪资一样的话，再按照名字升序排列。 select ename,sal from emp order by sal asc, ename asc; // sal在前，起主导，只有sal相等的时候，才会考虑启用ename排序。

```
# 综合案例
找出工资在1250到3000之间员工薪资降序排列;
```mysql
语法格式:
    select
        ename,sal
```
```

```
    from
        emp
    where
        sal between 1250 and 3000
    order by
        sal desc; // 默认是升序!!
指定降序: desc
指定升序: asc
select * from emp where sal>=1250 and sal<=3000 order by sal desc;
# 数据处理函数

* 数据处理函数又被称为单行处理函数

### 单行处理函数

```mysql
单行处理函数的特点:一个输入对应一个输出。
单行处理函数常见:
lower 转换小写
    mysql> select lower(ename) as ename from emp;

upper 转换大写
    mysql> select upper(name) as name from t_student;

substr 取子串 (substr(被截取的字符串, 起始下标,截取的长度))
    mysql> select substr(ename, 1, 1) as ename from emp;

concat函数进行字符串的拼接
    mysql> select concat(empno,ename) from emp;
length 取长度
    mysql> select length(ename) enamelength from emp;

trim 去空格
    mysql>  select * from emp where ename = trim('   KING');

str_to_date 将字符串转换成日期

date_format 格式化日期

format 设置千分位

# 字面量字面值
select 后面可以跟某个表的字段名(可以看成等同于变量名),也可以跟字面量或字面值(数据)

mysql> select 'KING' as '1' from emp;
+------+
| 1    |
+------+
| KING |
| KING |
| KING |
| KING |
| KING |
| KING |
```
```

```
| KING |
| KING |
| KING |
| KING |
| KING |
| KING |
| KING |
| KING |
+------+
14 rows in set (0.00 sec)

mysql> select 100  as '1' from emp;
+-----+
| 1   |
+-----+
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
| 100 |
+-----+
14 rows in set (0.00 sec)
```

case..when..then..when..then..else..end
当员工的工作岗位是MANAGER的时候，工资上调10%，当工作岗位是SALESMAN的时候，工资上调50%,其它正常。 （注意：不修改数据库，只是将查询结果显示为工资上调）

```
select
  ename,job, sal as oldsal,
  (case job when 'MANAGER' then sal*1.1 when 'SALESMAN' then sal*1.5 else sal end)
as newsal
from
  emp;

mysql> select ename,job,sal,(case job when 'MANAGER' then sal*1.1 when 'SALESMAN'
then sal*1.5 else sal end) as newsal from emp;
+--------+-----------+---------+---------+
| ename  | job       | sal     | newsal  |
+--------+-----------+---------+---------+
| SMITH  | CLERK     |  800.00 |  800.00 |
| ALLEN  | SALESMAN  | 1600.00 | 2400.00 |
| WARD   | SALESMAN  | 1250.00 | 1875.00 |
| JONES  | MANAGER   | 2975.00 | 3272.50 |
| MARTIN | SALESMAN  | 1250.00 | 1875.00 |
| BLAKE  | MANAGER   | 2850.00 | 3135.00 |
| CLARK  | MANAGER   | 2450.00 | 2695.00 |
```

```
| SCOTT  | ANALYST   | 3000.00 | 3000.00 |
| KING   | PRESIDENT | 5000.00 | 5000.00 |
| TURNER | SALESMAN  | 1500.00 | 2250.00 |
| ADAMS  | CLERK     | 1100.00 | 1100.00 |
| JAMES  | CLERK     |  950.00 |  950.00 |
| FORD   | ANALYST   | 3000.00 | 3000.00 |
| MILLER | CLERK     | 1300.00 | 1300.00 |
+--------+-----------+---------+---------+
14 rows in set (0.00 sec)
```

round 四舍五入

```
mysql> select round(1236.567, 0) as result from emp; //保留整数位。
mysql> select round(1236.567, 1) as result from emp; //保留1个小数
mysql> select round(1236.567, 2) as result from emp; //保留2个小数
mysql> select round(1236.567, -1) as result from emp; // 保留到十位。
```

rand() 生成随机数

```
  mysql> select round(rand()*100,0) from emp; // 100以内的随机数
```

ifnull 可以将 null 转换成一个具体值

ifnull是空处理函数。专门处理空的。在所有数据库当中，只要有NULL参与的数学运算，最终结果就是NULL。

注意：NULL只要参与运算，最终结果一定是NULL。为了避免这个现象，需要使用ifnull函数。ifnull函数用法：ifnull(数据，被当做哪个值)。如果“数据”为NULL的时候，把这个数据结构当做哪个值。

```
mysql> select ename,sal+comm as salcomm from emp;
+--------+---------+
| ename  | salcomm |
+--------+---------+
| SMITH  |    NULL |
| ALLEN  | 1900.00 |
| WARD   | 1750.00 |
| JONES  |    NULL |
| MARTIN | 2650.00 |
| BLAKE  |    NULL |
| CLARK  |    NULL |
| SCOTT  |    NULL |
| KING   |    NULL |
| TURNER | 1500.00 |
| ADAMS  |    NULL |
| JAMES  |    NULL |
| FORD   |    NULL |
| MILLER |    NULL |
+--------+---------+
14 rows in set (0.00 sec)

mysql> select ename,sal,comm from emp;
+--------+---------+---------+
| ename  | sal     | comm    |
+--------+---------+---------+
| SMITH  |  800.00 |    NULL |
| ALLEN  | 1600.00 |  300.00 |
| WARD   | 1250.00 |  500.00 |
| JONES  | 2975.00 |    NULL |
```

```
| MARTIN | 1250.00 | 1400.00 |
| BLAKE  | 2850.00 |    NULL |
| CLARK  | 2450.00 |    NULL |
| SCOTT  | 3000.00 |    NULL |
| KING   | 5000.00 |    NULL |
| TURNER | 1500.00 |    0.00 |
| ADAMS  | 1100.00 |    NULL |
| JAMES  |  950.00 |    NULL |
| FORD   | 3000.00 |    NULL |
| MILLER | 1300.00 |    NULL |
+--------+---------+---------+
14 rows in set (0.00 sec)

mysql> select ename,(sal+ifnull(comm,0))*12 as yearcomm from emp;
+--------+----------+
| ename  | yearcomm |
+--------+----------+
| SMITH  |  9600.00 |
| ALLEN  | 22800.00 |
| WARD   | 21000.00 |
| JONES  | 35700.00 |
| MARTIN | 31800.00 |
| BLAKE  | 34200.00 |
| CLARK  | 29400.00 |
| SCOTT  | 36000.00 |
| KING   | 60000.00 |
| TURNER | 18000.00 |
| ADAMS  | 13200.00 |
| JAMES  | 11400.00 |
| FORD   | 36000.00 |
| MILLER | 15600.00 |
+--------+----------+
14 rows in set (0.00 sec)
```

分组函数（多行处理函数）

多行处理函数特点：多个输入，对应1个输出。

```
count   # 计数
sum     # 求和
avg     # 平均值
max     # 最大值
min     # 最小值

注意：
    1. 分组函数在使用的时候必须先进行分组，然后才能用。如果你没有对数据进行分组，整张表默认为一组。
    2. 分组函数自动忽略NULL，你不需要提前对NULL进行处理。
    3. 分组函数中count(*)和count(具体字段)有什么区别？
        count(具体字段)：表示统计该字段下所有不为NULL的元素的总数。
```

```
        count(*)：统计表当中的总行数。（只要有一行数据count则++）
        因为每一行记录不可能都为NULL，一行数据中有一列不为NULL，则这行数据就是有效的。
    4. 分组函数不能够直接使用在where子句中。
    5. 所有的分组函数可以组合起来一起用。
```

distinct

> 把查询结果去除重复记录 **distinct**

注意：原表数据不会被修改，只是查询结果去重。

```
// distinct只能出现在所有字段的最前方。
mysql> select distinct job from emp;

// distinct出现在job,deptno两个字段之前，表示两个字段联合起来去重。
mysql> select distinct job,deptno from emp;
```

连接查询

> 多张表联合起来查询数据，被称为连接查询。

```
根据表连接的方式分类：
内连接：  等值连接  非等值连接  自连接

外连接：  左外连接（左连接）      右外连接（右连接）
```

内连接：等值连接

```
SQL99语法：
  select
    ...
  from
    a
  inner join
    b
  on
    a和b的连接条件
  where
    筛选条件
# inner可以省略（带着inner可读性更好！！！一眼就能看出来是内连接）
```

内连接：非等值连接

```
select
    e.ename, e.sal, s.grade
from
    emp e
join
    salgrade s
on
    e.sal between s.losal and s.hisal; // 条件不是一个等量关系，称为非等值连接。
```

内连接：自连接

```
select
    a.ename as '员工名', b.ename as '领导名'
from
    emp a
join
    emp b
on
    a.mgr = b.empno; //员工的领导编号 = 领导的员工编号
以上就是内连接中的：自连接，技巧：一张表看做两张表
```

外连接

```
// outer是可以省略的，带着可读性强。
select
    e.ename,d.dname
from
    emp e
right outer join
    dept d
on
    e.deptno = d.deptno;

right代表什么：表示将join关键字右边的这张表看成主表，主要是为了将这张表的数据全部查询出来，捎带着关联查询左边的表。在外连接当中，两张表连接，产生了主次关系。

带有right的是右外连接，又叫做右连接。
带有left的是左外连接，又叫做左连接。
任何一个右连接都有左连接的写法。
任何一个左连接都有右连接的写法。

思考：外连接的查询结果条数一定是 >= 内连接的查询结果条数？
    正确。


三张表，四张表怎么连接？
语法：
  select
```

```
    ...
  from
    a
  join
    b
  on
    a和b的连接条件
  join
    c
  on
    a和c的连接条件
  right join
    d
  on
    a和d的连接条件
```

一条SQL中内连接和外连接可以混合。都可以出现！

子查询

> select语句中嵌套select语句，被嵌套的select语句称为子查询。

```
子查询都可以出现在哪里呢？
select
  ..(select).
from
  ..(select).
where
  ..(select).


where子句中的子查询：

第一步：查询最低工资是多少
  select min(sal) from emp;
  +----------+
  | min(sal) |
  +----------+
  |   800.00 |
  +----------+
第二步：找出>800的
  select ename,sal from emp where sal > 800;

第三步：合并
  select ename,sal from emp where sal > (select min(sal) from emp);

from子句中的子查询：
    注意：from后面的子查询，可以将子查询的查询结果当做一张临时表。
```

```
select后面出现的子查询（这个内容不需要掌握，了解即可！！！）
    注意：对于select后面的子查询来说，这个子查询只能一次返回1条结果
```

union合并

```
案例：查询工作岗位是MANAGER和SALESMAN的员工？
select ename,job from emp where job = 'MANAGER' or job = 'SALESMAN';
select ename,job from emp where job in('MANAGER','SALESMAN');
+--------+----------+
| ename  | job      |
+--------+----------+
| ALLEN  | SALESMAN |
| WARD   | SALESMAN |
| JONES  | MANAGER  |
| MARTIN | SALESMAN |
| BLAKE  | MANAGER  |
| CLARK  | MANAGER  |
| TURNER | SALESMAN |
+--------+----------+

select ename,job from emp where job = 'MANAGER'
union
select ename,job from emp where job = 'SALESMAN';

+--------+----------+
| ename  | job      |
+--------+----------+
| JONES  | MANAGER  |
| BLAKE  | MANAGER  |
| CLARK  | MANAGER  |
| ALLEN  | SALESMAN |
| WARD   | SALESMAN |
| MARTIN | SALESMAN |
| TURNER | SALESMAN |
+--------+----------+
union的效率要高一些。对于表连接来说，每连接一次新表，
则匹配的次数满足笛卡尔积，成倍的翻。。。
但是union可以减少匹配的次数。在减少匹配次数的情况下，
还可以完成两个结果集的拼接。

a 连接 b 连接 c
a 10条记录
b 10条记录
c 10条记录
匹配次数是：1000

a 连接 b一个结果：10 * 10 --> 100次
a 连接 c一个结果：10 * 10 --> 100次
使用union的话是：100次 + 100次 = 200次。（union把乘法变成了加法运算）
```

union在使用的时候有注意事项吗?

```
//错误的: union在进行结果集合并的时候, 要求两个结果集的列数相同。
select ename,job from emp where job = 'MANAGER'
union
select ename from emp where job = 'SALESMAN';

// MYSQL可以, oracle语法严格 , 不可以, 报错。要求: 结果集合并时列和列的数据类型也要一致。
select ename,job from emp where job = 'MANAGER'
union
select ename,sal from emp where job = 'SALESMAN';
```

limit

limit作用: 将查询结果集的一部分取出来。通常使用在分页查询当中。
百度默认: 一页显示10条记录。
分页的作用是为了提高用户的体验, 因为一次全部都查出来, 用户体验差。可以一页一页翻页看

limit的使用
完整用法: limit startIndex, length
  startIndex是起始下标, length是长度。
  起始下标从0开始。

缺省用法: limit 5; 这是取前5

注意: mysql当中limit在order by之后执行!!!!!!

分页
每页显示3条记录
第1页: limit 0,3   [0 1 2]
第2页: limit 3,3   [3 4 5]
第3页: limit 6,3   [6 7 8]
第4页: limit 9,3   [9 10 11]

每页显示pageSize条记录
第pageNo页: limit (pageNo - 1) * pageSize , pageSize

```
public static void main(String[] args){
  // 用户提交过来一个页码, 以及每页显示的记录条数
  int pageNo = 5; //第5页
  int pageSize = 10; //每页显示10条

  int startIndex = (pageNo - 1) * pageSize;
  String sql = "select ...limit " + startIndex + ", " + pageSize;
}
```

记公式:
limit (pageNo-1)*pageSize , pageSize

关于DQL语句的大总结:
```
select
  ...
from
  ...
where
  ...
group by
  ...
having
  ...
order by
  ...
limit
  ...
```

执行顺序?
1.from
2.where
3.group by
4.having
5.select
6.order by
7.limit..

表的创建

建表的语法格式:(建表属于DDL语句,DDL包括:create drop alter)
```
create table 表名(字段名1 数据类型, 字段名2 数据类型, 字段名3 数据类型);

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

表名:建议以t_ 或者 tbl_开始,可读性强。见名知意。
字段名:见名知意。
表名和字段名都属于标识符。

关于mysql中的数据类型
很多数据类型,我们只需要掌握一些常见的数据类型即可。

varchar(最长255)

```
    可变长度的字符串
    比较智能，节省空间。
    会根据实际的数据长度动态分配空间。

    优点：节省空间
    缺点：需要动态分配空间，速度慢。

  char(最长255)
    定长字符串
    不管实际的数据长度是多少。
    分配固定长度的空间去存储数据。
    使用不恰当的时候，可能会导致空间的浪费。

    优点：不需要动态分配空间，速度快。
    缺点：使用不当可能会导致空间的浪费。

   varchar 和 char 我们应该怎么选择?
      性别字段你选什么？因为性别是固定长度的字符串，所以选择char。
      姓名字段你选什么？每一个人的名字长度不同，所以选择varchar。

  int(最长11)
    数字中的整数型。等同于java的int。

  bigint
    数字中的长整型。等同于java中的long。

  float
    单精度浮点型数据

  double
    双精度浮点型数据

  date
    短日期类型

  datetime
    长日期类型

  clob
    字符大对象
    最多可以存储4G的字符串。
    比如：存储一篇文章，存储一个说明。
    超过255个字符的都要采用CLOB字符大对象来存储。
    Character Large OBject:CLOB

  blob
    二进制大对象
    Binary Large OBject
    专门用来存储图片、声音、视频等流媒体数据。
    往BLOB类型的字段上插入数据的时候，例如插入一个图片、视频等，
    你需要使用IO流才行。


创建一个学生表?
```

```
  学号、姓名、年龄、性别、邮箱地址
  create table t_student(
    no int,
    name varchar(32),
    sex char(1),
    age int(3),
    email varchar(255)
  );

删除表:
drop table t_student; // 当这张表不存在的时候会报错!

// 如果这张表存在的话，删除
drop table if exists t_student;
```

插入数据insert

```
插入数据insert  (DML)

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);

  注意：字段名和值要一一对应。什么是一一对应?
    数量要对应。数据类型要对应。

insert语句中的“字段名”可以省略吗? 可以
  insert into t_student values(2); //错误的

  // 注意：前面的字段名省略的话，等于都写上了! 所以值也要都写上!
  insert into t_student values(2, 'lisi', 'f', 20, 'lisi@123.com');
  +------+------+------+------+--------------+
  | no   | name | sex  | age  | email        |
  +------+------+------+------+--------------+
  |    1 | NULL | m    | NULL | NULL         |
  |    2 | lisi | f    |   20 | lisi@123.com |
  +------+------+------+------+--------------+

一次可以插入多条记录:
  insert into t_user(id,name,birth,create_time) values
  (1,'zs','1980-10-11',now()),
  (2,'lisi','1981-10-11',now()),
  (3,'wangwu','1982-10-11',now());

  语法: insert into t_user(字段名1,字段名2) values(),(),(),();


insert插入日期
    数字格式化: format
```

```
    格式化数字：format(数字，'格式')
  select ename,format(sal, '$999,999') as sal from emp;

str_to_date：将字符串varchar类型转换成date类型
date_format：将date类型转换成具有一定格式的varchar字符串类型。


插入数据？
insert into t_user(id,name,birth) values(1, 'zhangsan', '01-10-1990'); // 1990年10
月1日
出问题了：原因是类型不匹配。数据库birth是date类型，这里给了一个字符串varchar。

怎么办？可以使用str_to_date函数进行类型转换。
str_to_date函数可以将字符串转换成日期类型date？
语法格式：
  str_to_date('字符串日期', '日期格式')

mysql的日期格式：
  %Y    年
  %m    月
  %d    日
  %h    时
  %i    分
  %s    秒

insert into t_user(id,name,birth) values(1, 'zhangsan', str_to_date('01-10-
1990','%d-%m-%Y'));

str_to_date函数可以把字符串varchar转换成日期date类型数据，
通常使用在插入insert方面，因为插入的时候需要一个日期类型的数据，
需要通过该函数将字符串转换成date。

如果你提供的日期字符串是这个格式，str_to_date函数就不需要了！！！
  %Y-%m-%d
insert into t_user(id,name,birth) values(2, 'lisi', '1990-10-01');

查询的时候可以以某个特定的日期格式展示吗？
date_format
这个函数可以将日期类型转换成特定格式的字符串。

select id,name,date_format(birth, '%m/%d/%Y') as birth from t_user;
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 10/01/1990 |
|    2 | lisi     | 10/01/1990 |
+------+----------+------------+

date_format函数怎么用？
  date_format(日期类型数据，'日期格式')
  这个函数通常使用在查询日期方面。设置展示的日期格式。

mysql> select id,name,birth from t_user;
```

```
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 1990-10-01 |
|    2 | lisi     | 1990-10-01 |
+------+----------+------------+
```

以上的SQL语句实际上是进行了默认的日期格式化，
自动将数据库中的date类型转换成varchar类型。
并且采用的格式是mysql默认的日期格式：'%Y-%m-%d'

```
select id,name,date_format(birth,'%Y/%m/%d') as birth from t_user;
```

java中的日期格式？
  yyyy-MM-dd HH:mm:ss SSS

date和datetime两个类型的区别？
    date是短日期：只包括年月日信息。
    datetime是长日期：包括年月日时分秒信息。
    mysql短日期默认格式：%Y-%m-%d
    mysql长日期默认格式：%Y-%m-%d %h:%i:%s

修改update

语法格式：
update 表名 set 字段名1=值1,字段名2=值2,字段名3=值3... where 条件;

注意：没有条件限制会导致所有数据全部更新。

```
update t_user set name = 'jack', birth = '2000-10-11' where id = 2;
+------+----------+------------+---------------------+
| id   | name     | birth      | create_time         |
+------+----------+------------+---------------------+
|    1 | zhangsan | 1990-10-01 | 2020-03-18 15:49:50 |
|    2 | jack     | 2000-10-11 | 2020-03-18 15:51:23 |
+------+----------+------------+---------------------+

update t_user set name = 'jack', birth = '2000-10-11', create_time = now() where
id = 2;
```

更新所有？
  update t_user set name = 'abc';

删除数据 delete

语法格式?
  delete from 表名 where 条件;

注意:没有条件,整张表的数据会全部删除!

```
delete from t_user where id = 2;

insert into t_user(id) values(2);

delete from t_user; // 删除所有!
```

快速删除表中的数据?【truncate比较重要,必须掌握】

```
//删除dept_bak表中的数据
delete from dept_bak; //这种删除数据的方式比较慢。

mysql> select * from dept_bak;
Empty set (0.00 sec)
```

delete语句删除数据的原理?(delete属于DML语句!!!)
  表中的数据被删除了,但是这个数据在硬盘上的真实存储空间不会被释放!!!
  这种删除缺点是:删除效率比较低。
  这种删除优点是:支持回滚,后悔了可以再恢复数据!!!

truncate语句删除数据的原理?
  这种删除效率比较高,表被一次截断,物理删除。
  这种删除缺点:不支持回滚。
  这种删除优点:快速。

用法:truncate table dept_bak; (这种操作属于DDL操作。)

大表非常大,上亿条记录???
  删除的时候,使用delete,也许需要执行1个小时才能删除完!效率较低。
  可以选择使用truncate删除表中的数据。只需要不到1秒钟的时间就删除结束。效率较高。
  但是使用truncate之前,必须仔细询问客户是否真的要删除,并警告删除之后不可恢复!

  truncate是删除表中的数据,表还在!

删除表操作?
  drop table 表名; // 这不是删除表中的数据,这是把表删除。

约束

什么是约束?
约束对应的英语单词: constraint
在创建表的时候,我们可以给表中的字段加上一些约束,来保证这个表中数据的
完整性、有效性!!!

```
约束的作用就是为了保证：表中的数据有效！！

约束包括哪些?
非空约束：not null
唯一性约束：unique
主键约束：primary key （简称PK）
外键约束：foreign key（简称FK）
检查约束：check（mysql不支持，oracle支持）
```

非空约束：not null

> 唯一性约束unique约束的字段不能重复，但是可以为NULL。

```
drop table if exists t_vip;
create table t_vip(
  id int,
  name varchar(255) not null  // not null只有列级约束，没有表级约束!
);
```

唯一性约束: unique

> 唯一性约束unique约束的字段不能重复，但是可以为NULL。

```
唯一性约束unique约束的字段不能重复，但是可以为NULL。
drop table if exists t_vip;
create table t_vip(
  id int,
  name varchar(255) unique,
  email varchar(255)
);

如何做到联合唯一约束
drop table if exists t_vip;
create table t_vip(
  id int,
  name varchar(255),
  email varchar(255),
  unique(name,email) // 约束没有添加在列的后面，这种约束被称为表级约束。
);
name和email两个字段联合起来唯一
在mysql当中，如果一个字段同时被not null和unique约束的话，该字段自动变成主键字段。
```

主键约束: primary key

```
主键约束的相关术语?
  主键约束：就是一种约束。
```

主键字段：该字段上添加了主键约束，这样的字段叫做：主键字段
主键值：主键字段中的每一个值都叫做：主键值。

什么是主键？有啥用？
主键值是每一行记录的唯一标识。
主键值是每一行记录的身份证号！！！

记住：任何一张表都应该有主键，没有主键，表无效！！

主键的特征：not null + unique（主键值不能是NULL，同时也不能重复！）

怎么给一张表添加主键约束呢？
```
drop table if exists t_vip;
// 1个字段做主键，叫做：单一主键
create table t_vip(
  id int primary key,  //列级约束
  name varchar(255)
);
```

表级约束主要是给多个字段联合起来添加约束？
```
drop table if exists t_vip;
// id和name联合起来做主键：复合主键！！！！
create table t_vip(
  id int,
  name varchar(255),
  email varchar(255),
  primary key(id,name)
);
```

在实际开发中不建议使用：复合主键。建议使用单一主键！
因为主键值存在的意义就是这行记录的身份证号，只要意义达到即可，单一主键可以做到。
复合主键比较复杂，不建议使用！！！

主键值建议使用：
int
bigint
char
等类型。

不建议使用：varchar来做主键。主键值一般都是数字，一般都是定长的！

主键除了：单一主键和复合主键之外，还可以这样进行分类？
自然主键：主键值是一个自然数，和业务没关系。
业务主键：主键值和业务紧密关联，例如拿银行卡账号做主键值。这就是业务主键！

在实际开发中使用业务主键多，还是使用自然主键多一些？
自然主键使用比较多，因为主键只要做到不重复就行，不需要有意义。
业务主键不好，因为主键一旦和业务挂钩，那么当业务发生变动的时候，
可能会影响到主键值，所以业务主键不建议使用。尽量使用自然主键。

在mysql当中，有一种机制，可以帮助我们自动维护一个主键值？
```
        drop table if exists t_vip;
        create table t_vip(
            id int primary key auto_increment, //auto_increment表示自增，从1开始，
```

```
以1递增!
            name varchar(255)
        );
```

外键约束：foreign key

外键约束涉及到的相关术语：
  外键约束：一种约束（ foreign key）
  外键字段：该字段上添加了外键约束
  外键值：外键字段当中的每一个值。

存储引擎

> 什么是存储引擎，有什么用呢？

存储引擎是MySQL中特有的一个术语，其它数据库中没有。（Oracle中有，但是不叫这个名字）
存储引擎这个名字高端大气上档次。实际上存储引擎是一个表存储/组织数据的方式。
不同的存储引擎，表存储数据的方式不同。

怎么给表添加/指定“存储引擎”呢？
```
show create table t_student;
```

可以在建表的时候给表指定存储引擎。
```
CREATE TABLE `t_student` (
  `no` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `cno` int(11) DEFAULT NULL,
  PRIMARY KEY (`no`),
  KEY `cno` (`cno`),
  CONSTRAINT `t_student_ibfk_1` FOREIGN KEY (`cno`) REFERENCES `t_class`
(`classno`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8
```

在建表的时候可以在最后小括号的")"的右边使用：
  ENGINE来指定存储引擎。
  CHARSET来指定这张表的字符编码方式。

结论：
  mysql默认的存储引擎是：InnoDB
  mysql默认的字符编码方式是：utf8

怎么查看mysql支持哪些存储引擎呢？
命令：
```
show engines \G
```

MyISAM存储引擎

它管理的表具有以下特征:
使用三个文件表示每个表:
格式文件 — 存储表结构的定义 (mytable.frm)
数据文件 — 存储表行的内容 (mytable.MYD)
索引文件 — 存储表上索引 (mytable.MYI) : 索引是一本书的目录，缩小扫描范围，提高查询效率的一种机制。
可被转换为压缩、只读表来节省空间

提示一下:
对于一张表来说，只要是主键，
或者加有unique约束的字段上会自动创建索引。

MyISAM存储引擎特点:
可被转换为压缩、只读表来节省空间
这是这种存储引擎的优势！！！！

MyISAM不支持事务机制，安全性低。

InnoDB存储引擎

这是mysql默认的存储引擎，同时也是一个重量级的存储引擎。
InnoDB支持事务，支持数据库崩溃后自动恢复机制。
InnoDB存储引擎最主要的特点是:非常安全。

它管理的表具有下列主要特征:
- 每个 InnoDB 表在数据库目录中以.frm 格式文件表示
- InnoDB 表空间 tablespace 被用于存储表的内容 (表空间是一个逻辑名称。表空间存储数据+索引。)

- 提供一组用来记录事务性活动的日志文件
- 用 COMMIT(提交)、SAVEPOINT 及ROLLBACK(回滚)支持事务处理
- 提供全 ACID 兼容
- 在 MySQL 服务器崩溃后提供自动恢复
- 多版本 (MVCC) 和行级锁定
- 支持外键及引用的完整性，包括级联删除和更新

InnoDB最大的特点就是支持事务:
以保证数据的安全。效率不是很高，并且也不能压缩，不能转换为只读，
不能很好的节省存储空间。

MEMORY存储引擎

使用 MEMORY 存储引擎的表，其数据存储在内存中，且行的长度固定，
这两个特点使得 MEMORY 存储引擎非常快。

MEMORY 存储引擎管理的表具有下列特征:
- 在数据库目录内，每个表均以.frm 格式的文件表示。
- 表数据及索引被存储在内存中。（目的就是快，查询快！）
- 表级锁机制。
- 不能包含 TEXT 或 BLOB 字段。

MEMORY 存储引擎以前被称为HEAP 引擎。

MEMORY引擎优点：查询效率是最高的。不需要和硬盘交互。
MEMORY引擎缺点：不安全，关机之后数据消失。因为数据和索引都是在内存当中。

事务

什么是事务?

一个事务其实就是一个完整的业务逻辑。
是一个最小的工作单元。不可再分。

什么是一个完整的业务逻辑?
假设转账，从A账户向B账户中转账10000.
将A账户的钱减去10000（update语句）
将B账户的钱加上10000（update语句）
这就是一个完整的业务逻辑。

以上的操作是一个最小的工作单元，要么同时成功，要么同时失败，不可再分。
这两个update语句要求必须同时成功或者同时失败，这样才能保证钱是正确的。

只有DML语句才会有事务这一说，其它语句和事务无关！！！
insert
delete
update
只有以上的三个语句和事务有关系，其它都没有关系。

因为 只有以上的三个语句是数据库表中数据进行增、删、改的。
只要你的操作一旦涉及到数据的增、删、改，那么就一定要考虑安全问题。

InnoDB存储引擎：提供一组用来记录事务性活动的日志文件

事务开启了:
insert
insert
insert
delete
update
update
update
事务结束了!

在事务的执行过程中，每一条DML的操作都会记录到“事务性活动的日志文件”中。

在事务的执行过程中，我们可以提交事务，也可以回滚事务。

提交事务?
  清空事务性活动的日志文件，将数据全部彻底持久化到数据库表中。
  提交事务标志着，事务的结束。并且是一种全部成功的结束。

回滚事务?
  将之前所有的DML操作全部撤销，并且清空事务性活动的日志文件
  回滚事务标志着，事务的结束。并且是一种全部失败的结束。

提交事务、回滚事务

提交事务：commit; 语句
回滚事务：rollback; 语句（回滚永远都是只能回滚到上一次的提交点！）

事务对应的英语单词是：transaction

测试一下，在mysql当中默认的事务行为是怎样的?
  mysql默认情况下是支持自动提交事务的。（自动提交）
  什么是自动提交?
    每执行一条DML语句，则提交一次!

  这种自动提交实际上是不符合我们的开发习惯，因为一个业务
  通常是需要多条DML语句共同执行才能完成的，为了保证数据
  的安全，必须要求同时成功之后再提交，所以不能执行一条
  就提交一条。

怎么将mysql的自动提交机制关闭掉呢?
  先执行这个命令：start transaction;

代码演示

```
演示事务:
  ---------------------------------回滚事务---------------------------------------
-
  mysql> use bjpowernode;
  Database changed
  mysql> select * from dept_bak;
  Empty set (0.00 sec)

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

  mysql> insert into dept_bak values(10,'abc', 'tj');
  Query OK, 1 row affected (0.00 sec)

  mysql> insert into dept_bak values(10,'abc', 'tj');
  Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | tj   |
|     10 | abc   | tj   |
+--------+-------+------+
2 rows in set (0.00 sec)

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

mysql> select * from dept_bak;
Empty set (0.00 sec)


---------------------------------提交事务---------------------------------------
-
mysql> use bjpowernode;
Database changed
mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
+--------+-------+------+
1 row in set (0.00 sec)

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

mysql> insert into dept_bak values(20,'abc
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_bak values(20,'abc
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_bak values(20,'abc
Query OK, 1 row affected (0.00 sec)

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

mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | abc   | tj   |
|     20 | abc   | tj   |
|     20 | abc   | tj   |
+--------+-------+------+
4 rows in set (0.00 sec)
```

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

mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | abc   | tj   |
|     20 | abc   | tj   |
|     20 | abc   | tj   |
+--------+-------+------+
4 rows in set (0.00 sec)
```

事务四个特性

A：原子性
  说明事务是最小的工作单元。不可再分。

C：一致性
  所有事务要求，在同一个事务当中，所有操作必须同时成功，或者同时失败，
  以保证数据的一致性。

I：隔离性
  A事务和B事务之间具有一定的隔离。
  教室A和教室B之间有一道墙，这道墙就是隔离性。
  A事务在操作一张表的时候，另一个事务B也操作这张表会那样？？？

D：持久性
  事务最终结束的一个保障。事务提交，就相当于将没有保存到硬盘上的数据
  保存到硬盘上！

事务的隔离性

A教室和B教室中间有一道墙，这道墙可以很厚，也可以很薄。这就是事务的隔离级别。
这道墙越厚，表示隔离级别就越高。

事务和事务之间的隔离级别有哪些呢？4个级别

读未提交：read uncommitted（最低的隔离级别）《没有提交就读到了》
  什么是读未提交？
    事务A可以读取到事务B未提交的数据。
  这种隔离级别存在的问题就是：
    脏读现象！(Dirty Read)
    我们称读到了脏数据。
  这种隔离级别一般都是理论上的，大多数的数据库隔离级别都是二档起步！

读已提交：read committed《提交之后才能读到》

```
  什么是读已提交?
    事务A只能读取到事务B提交之后的数据。
  这种隔离级别解决了什么问题?
    解决了脏读的现象。
  这种隔离级别存在什么问题?
    不可重复读取数据。
    什么是不可重复读取数据呢?
      在事务开启之后，第一次读到的数据是3条，当前事务还没有
      结束，可能第二次再读取的时候，读到的数据是4条，3不等于4
      称为不可重复读取。

  这种隔离级别是比较真实的数据，每一次读到的数据是绝对的真实。
  oracle数据库默认的隔离级别是: read committed

可重复读: repeatable read《提交之后也读不到，永远读取的都是刚开启事务时的数据》
  什么是可重复读取?
    事务A开启之后，不管是多久，每一次在事务A中读取到的数据
    都是一致的。即使事务B将数据已经修改，并且提交了，事务A
    读取到的数据还是没有发生改变，这就是可重复读。
  可重复读解决了什么问题?
    解决了不可重复读取数据。
  可重复读存在的问题是什么?
    可以会出现幻影读。
    每一次读取到的数据都是幻象。不够真实!

  早晨9点开始开启了事务，只要事务不结束，到晚上9点，读到的数据还是那样!
  读到的是假象。不够绝对的真实。

  mysql中默认的事务隔离级别就是这个!!!!!!!!!!

序列化/串行化: serializable (最高的隔离级别)
  这是最高隔离级别，效率最低。解决了所有的问题。
  这种隔离级别表示事务排队，不能并发!
  synchronized，线程同步 (事务同步)
  每一次读取到的数据都是最真实的，并且效率是最低的。

查看隔离级别: SELECT @@tx_isolation
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
mysql默认的隔离级别
```

索引 (index)

什么是索引?
索引是在数据库表的字段上添加的，是为了提高查询效率存在的一种机制。
一张表的一个字段可以添加一个索引，当然，多个字段联合起来也可以添加索引。
索引相当于一本书的目录，是为了缩小扫描范围而存在的一种机制。

对于一本字典来说，查找某个汉字有两种方式：
  第一种方式：一页一页挨着找，直到找到为止，这种查找方式属于全字典扫描。
  效率比较低。
  第二种方式：先通过目录（索引）去定位一个大概的位置，然后直接定位到这个
  位置，做局域性扫描，缩小扫描的范围，快速的查找。这种查找方式属于通过
  索引检索，效率较高。

```
select * from t_user where name = 'jack';
```

以上的这条SQL语句会去name字段上扫描，为什么？
  因为查询条件是：name='jack'

如果name字段上没有添加索引（目录），或者说没有给name字段创建索引，
MySQL会进行全扫描，会将name字段上的每一个值都比对一遍。效率比较低。

MySQL在查询方面主要就是两种方式：
  第一种方式：全表扫描
  第二种方式：根据索引检索。

注意：
  在实际中，汉语字典前面的目录是排序的，按照a b c d e f....排序，
  为什么排序呢？因为只有排序了才会有区间查找这一说！（缩小扫描范围
  其实就是扫描某个区间罢了！）

  在mysql数据库当中索引也是需要排序的，并且这个所以的排序和TreeSet
  数据结构相同。TreeSet（TreeMap）底层是一个自平衡的二叉树！在mysql
  当中索引是一个B-Tree数据结构。

  遵循左小又大原则存放。采用中序遍历方式遍历取数据。

实现原理

假设有一张用户表：t_user

```
id(PK)                  name                          每一行记录在硬盘上都有物理存储编
号
-------------------------------------------------------------------------------
100                     zhangsan                      0x1111
120                     lisi                          0x2222
99                          wangwu                    0x8888
88                          zhaoliu                   0x9999
101                     jack                          0x6666
55                          lucy                          0x5555
130                     tom                       0x7777
```

提醒1：在任何数据库当中主键上都会自动添加索引对象，id字段上自动有索引，
因为id是PK。另外在mysql当中，一个字段上如果有unique约束的话，也会自动
创建索引对象。

提醒2：在任何数据库当中，任何一张表的任何一条记录在硬盘存储上都有
一个硬盘的物理存储编号。

提醒3：在mysql当中，索引是一个单独的对象，不同的存储引擎以不同的形式
存在，在MyISAM存储引擎中，索引存储在一个.MYI文件中。在InnoDB存储引擎中
索引存储在一个逻辑名称叫做tablespace的当中。在MEMORY存储引擎当中索引
被存储在内存当中。不管索引存储在哪里，索引在mysql当中都是一个树的形式
存在。（自平衡二叉树：B-Tree）

添加索引的条件

什么条件下，我们会考虑给字段添加索引呢?
条件1：数据量庞大（到底有多么庞大算庞大，这个需要测试，因为每一个硬件环境不同）
条件2：该字段经常出现在where的后面，以条件的形式存在，也就是说这个字段总是被扫描。
条件3：该字段很少的DML(insert delete update)操作。（因为DML之后，索引需要重新排序。）

建议不要随意添加索引，因为索引也是需要维护的，太多的话反而会降低系统的性能。
建议通过主键查询，建议通过unique约束的字段进行查询，效率是比较高的。

索引的创建和删除

创建索引:
```
mysql> create index emp_ename_index on emp(ename);
```
给emp表的ename字段添加索引，起名：emp_ename_index

删除索引:
```
mysql> drop index emp_ename_index on emp;
```
将emp表上的emp_ename_index索引对象删除。

是否使用索引进行检索

1.5、在mysql当中，怎么查看一个SQL语句是否使用了索引进行检索?

```
mysql> explain select * from emp where ename = 'KING';
+----+-------------+-------+------+---------------+------+---------+------+------
+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows |
Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------
+-------------+
|  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |   14 |
Using where |
+----+-------------+-------+------+---------------+------+---------+------+------
+-------------+
```
扫描14条记录：说明没有使用索引。type=ALL

```
mysql> create index emp_ename_index on emp(ename);

mysql> explain select * from emp where ename = 'KING';
+----+-------------+-------+------+----------------+----------------+---------+-
------+------+-------------+
| id | select_type | table | type | possible_keys   | key            | key_len |
ref   | rows | Extra       |
+----+-------------+-------+------+----------------+----------------+---------+-
------+------+-------------+
|  1 | SIMPLE      | emp   | ref  | emp_ename_index | emp_ename_index | 33      |
const |    1 | Using where |
+----+-------------+-------+------+----------------+----------------+---------+-
------+------+-------------+
```

索引失效的情况

索引有失效的时候，什么时候索引失效呢?

失效的第1种情况:

```
select * from emp where ename like '%T';
```

ename上即使添加了索引，也不会走索引，为什么?

原因是因为模糊匹配当中以“%”开头了!

尽量避免模糊查询的时候以“%”开始。

这是一种优化的手段/策略。

```
mysql> explain select * from emp where ename like '%T';
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |   14
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

失效的第2种情况:

使用or的时候会失效，如果使用or那么要求or两边的条件字段都要有索引，才会走索引，如果其中一边有一个字段没有索引，那么另一个字段上的索引也会实现。所以这就是为什么不建议使用or的原因。

```
mysql> explain select * from emp where ename = 'KING' or job = 'MANAGER';
+----+-------------+-------+------+----------------+------+---------+------+---
---+-------------+
| id | select_type | table | type | possible_keys   | key  | key_len | ref  |
rows | Extra       |
+----+-------------+-------+------+----------------+------+---------+------+---
```

```
---+-------------+
  |  1 | SIMPLE      | emp   | ALL  | emp_ename_index | NULL | NULL    | NULL |
14 | Using where |
  +----+-------------+-------+------+-----------------+------+---------+------+---
---+-------------+
```

失效的第3种情况：
  使用复合索引的时候，没有使用左侧的列查找，索引失效
  什么是复合索引?
    两个字段，或者更多的字段联合起来添加一个索引，叫做复合索引。

```
  create index emp_job_sal_index on emp(job,sal);

  mysql> explain select * from emp where job = 'MANAGER';
  +----+-------------+-------+------+------------------+------------------+-----
----+-------+------+-------------+
  | id | select_type | table | type | possible_keys     | key              |
key_len | ref   | rows | Extra       |
  +----+-------------+-------+------+------------------+------------------+-----
----+-------+------+-------------+
  |  1 | SIMPLE      | emp   | ref  | emp_job_sal_index | emp_job_sal_index | 30
| const |    3 | Using where |
  +----+-------------+-------+------+------------------+------------------+-----
----+-------+------+-------------+

  mysql> explain select * from emp where sal = 800;
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
  | id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
  |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |   14
| Using where |
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

失效的第4种情况：
  在where当中索引列参加了运算，索引失效。

```
  mysql> create index emp_sal_index on emp(sal);

  explain select * from emp where sal = 800;
  +----+-------------+-------+------+---------------+---------------+---------+---
----+------+-------------+
  | id | select_type | table | type | possible_keys | key           | key_len |
ref   | rows | Extra       |
  +----+-------------+-------+------+---------------+---------------+---------+---
----+------+-------------+
  |  1 | SIMPLE      | emp   | ref  | emp_sal_index | emp_sal_index | 9       |
const |    1 | Using where |
  +----+-------------+-------+------+---------------+---------------+---------+---
----+------+-------------+

  mysql> explain select * from emp where sal+1 = 800;
```

```
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
  | id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
  |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |   14
| Using where |
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

失效的第5种情况：
在where当中索引列使用了函数

```
explain select * from emp where lower(ename) = 'smith';
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
  | id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
  |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |   14
| Using where |
  +----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

索引分类

单一索引：一个字段上添加索引。
复合索引：两个字段或者更多的字段上添加索引。

主键索引：主键上添加索引。
唯一性索引：具有unique约束的字段上添加索引。
.....

注意：唯一性比较弱的字段上添加索引用处不大。

视图

> 什么是视图?

视图：站在不同的角度去看待同一份数据。

视图创建和删除

```
创建视图对象：
  create view dept2_view as select * from dept2;

删除视图对象：
  drop view dept2_view;

注意：只有DQL语句才能以view的形式创建。
  create view view_name as 这里的语句必须是DQL语句;
```

视图的用途

```
《方便，简化开发，利于维护》
我们可以面向视图对象进行增删改查，对视图对象的增删改查，会导致
原表被操作！（视图的特点：通过对视图的操作，会影响到原表数据。）

//面向视图查询
select * from dept2_view;

// 面向视图插入
insert into dept2_view(deptno,dname,loc) values(60,'SALES', 'BEIJING');

// 查询原表数据
mysql> select * from dept2;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
|     60 | SALES      | BEIJING  |
+--------+------------+----------+

// 面向视图删除
mysql> delete from dept2_view;

// 查询原表数据
mysql> select * from dept2;
Empty set (0.00 sec)

假设有一条非常复杂的SQL语句，而这条SQL语句需要在不同的位置上反复使用。
每一次使用这个sql语句的时候都需要重新编写，很长，很麻烦，怎么办？
  可以把这条复杂的SQL语句以视图对象的形式新建。
  在需要编写这条SQL语句的位置直接使用视图对象，可以大大简化开发。
  并且利于后期的维护，因为修改的时候也只需要修改一个位置就行，只需要
  修改视图对象所映射的SQL语句。

我们以后面向视图开发的时候，使用视图的时候可以像使用table一样。
可以对视图进行增删改查等操作。视图不是在内存当中，视图对象也是
存储在硬盘上的，不会消失。
```

再提醒一下：
  视图对应的语句只能是DQL语句。
  但是视图对象创建完成之后，可以对视图进行增删改查等操作。

小插曲：
  增删改查，又叫做：CRUD。
  CRUD是在公司中程序员之间沟通的术语。一般我们很少说增删改查。
  一般都说CRUD。

  C:Create（增）
  R:Retrive（查：检索）
  U:Update（改）
  D:Delete（删）

DBA常用命令

重点掌握：
  数据的导入和导出（数据的备份）
  其它命令了解一下即可。

数据导出？
  注意：在windows的dos命令窗口中：
    mysqldump bjpowernode>D:\bjpowernode.sql -uroot -p123456

  可以导出指定的表吗？
    mysqldump bjpowernode emp>D:\bjpowernode.sql -uroot -p123456

数据导入？
  注意：需要先登录到mysql数据库服务器上。
  然后创建数据库：create database bjpowernode;
  使用数据库：use bjpowernode
  然后初始化数据库：source D:\bjpowernode.sql

数据库设计三范式

什么是数据库设计范式？
数据库表的设计依据。教你怎么进行数据库表的设计。

数据库设计范式共有？
3个。
第一范式：要求任何一张表必须有主键，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，要求所有非主键字段完全依赖主键，
不要产生部分依赖。

第三范式：建立在第二范式的基础之上，要求所有非主键字段直接依赖主键，
不要产生传递依赖。

声明：三范式是面试官经常问的，所以一定要熟记在心！

设计数据库表的时候，按照以上的范式进行，可以避免表中数据的冗余，空间的浪费。

第一范式
最核心，最重要的范式，所有表的设计都需要满足。
必须有主键，并且每一个字段都是原子性不可再分。

```
学生编号 学生姓名 联系方式
-----------------------------------------
1001        张三        zs@gmail.com,1359999999
1002        李四        ls@gmail.com,13699999999
1001        王五        ww@163.net,13488888888
```

以上是学生表，满足第一范式吗？
　不满足，第一：没有主键。第二：联系方式可以分为邮箱地址和电话

```
学生编号(pk) 学生姓名     邮箱地址          联系电话
-----------------------------------------------------
1001                张三        zs@gmail.com    1359999999
1002                李四        ls@gmail.com    13699999999
1003                王五        ww@163.net      13488888888
```

第二范式：
建立在第一范式的基础之上，
要求所有非主键字段必须完全依赖主键，不要产生部分依赖。

```
学生编号 学生姓名 教师编号 教师姓名
-----------------------------------------------------
1001            张三        001     王老师
1002            李四        002     赵老师
1003            王五        001     王老师
1001            张三        002     赵老师
```

这张表描述了学生和老师的关系：（1个学生可能有多个老师，1个老师有多个学生）
这是非常典型的：多对多关系！

分析以上的表是否满足第一范式？
　不满足第一范式。

怎么满足第一范式呢？修改

```
学生编号+教师编号(pk)          学生姓名   教师姓名
-----------------------------------------------------
1001            001             张三            王老师
1002            002             李四            赵老师
1003            001             王五            王老师
1001            002             张三            赵老师
```

学生编号 教师编号，两个字段联合做主键，复合主键（PK：学生编号+教师编号）
经过修改之后，以上的表满足了第一范式。但是满足第二范式吗？

不满足，“张三”依赖1001，“王老师”依赖001，显然产生了部分依赖。
产生部分依赖有什么缺点?
数据冗余了。空间浪费了。“张三”重复了，“王老师”重复了。

为了让以上的表满足第二范式，你需要这样设计:
使用三张表来表示多对多的关系！！！！
学生表

学生编号(pk)	学生名字
1001	张三
1002	李四
1003	王五

教师表

教师编号(pk)	教师姓名
001	王老师
002	赵老师

学生教师关系表

id(pk)	学生编号(fk)	教师编号(fk)
1	1001	001
2	1002	002
3	1003	001
4	1001	002

背口诀:
多对多怎么设计?
多对多，三张表，关系表两个外键！！！！！！！！！！！！！！

第三范式
第三范式建立在第二范式的基础之上
要求所有非主键字典必须直接依赖主键，不要产生传递依赖。

学生编号（PK）	学生姓名	班级编号	班级名称
1001	张三	01	一年一班
1002	李四	02	一年二班
1003	王五	03	一年三班
1004	赵六	03	一年三班

以上表的设计是描述：班级和学生的关系。很显然是1对多关系!
一个教室中有多个学生。

分析以上表是否满足第一范式?
满足第一范式，有主键。

分析以上表是否满足第二范式?
满足第二范式，因为主键不是复合主键，没有产生部分依赖。主键是单一主键。

分析以上表是否满足第三范式?

```
  第三范式要求：不要产生传递依赖！
  一年一班依赖01，01依赖1001，产生了传递依赖。
  不符合第三范式的要求。产生了数据的冗余。

那么应该怎么设计一对多呢？

  班级表：一
  班级编号(pk)                班级名称
  ---------------------------------------
  01                                一年一班
  02                                一年二班
  03                                一年三班

  学生表：多

  学生编号（PK）  学生姓名 班级编号(fk)
  -----------------------------------------
  1001              张三            01
  1002              李四            02
  1003              王五            03
  1004              赵六            03

  背口诀：
    一对多，两张表，多的表加外键！！！！！！！！！！！
```

总结表的设计

```
一对多：
  一对多，两张表，多的表加外键！！！！！！！！！！！

多对多：
  多对多，三张表，关系表两个外键！！！！！！！！！！！！！！

一对一：
一对一放到一张表中不就行了吗？为啥还要拆分表？
在实际的开发中，可能存在一张表字段太多，太庞大。这个时候要拆分表。
一对一怎么设计？
  没有拆分表之前：一张表
    t_user
    id      login_name      login_pwd       real_name       email
address........
    ------------------------------------------------------------------------
    1           zhangsan        123             张三                zhangsan@xxx
    2           lisi            123             李四                lisi@xxx
    ...

  这种庞大的表建议拆分为两张：
    t_login 登录信息表
    id(pk)      login_name      login_pwd
```

```
    ---------------------------------
    1               zhangsan        123
    2               lisi            123

    t_user 用户详细信息表
    id(pk)      real_name       email               address........
login_id(fk+unique)
    ------------------------------------------------------------------------------
-----------
    100         张三                zhangsan@xxx                                1
    200         李四                lisi@xxx                                    2


    口诀：一对一，外键唯一！！！！！！！！！！
```

嘱咐

数据库设计三范式是理论上的。

实践和理论有的时候有偏差。

最终的目的都是为了满足客户的需求，有的时候会拿冗余换执行速度。

因为在sql当中，表和表之间连接次数越多，效率越低。（笛卡尔积）

有的时候可能会存在冗余，但是为了减少表的连接次数，这样做也是合理的，
并且对于开发人员来说，sql语句的编写难度也会降低。

面试的时候把这句话说上：他就不会认为你是初级程序员了！