数据库的约束

I.域完整性约束

- 属性值应是域中的值，属性的值能否为**null**，由语义决定。域完整性约束是在确立关系模式时规定的，由**DBMS**负责检查。

2.实体完整性约束

- 每个关系应有一个主码，主码的值不能为**null**，这就是实体完整性约束。如果在关系模式中说明了主码，**DBMS**可以进行这项检查

3.引用完整性约束

- 不同关系之间或同一关系的不同元组间的约束。若关系**R**中有一个外码（相对于关系**S**），则**R**中每个元组的外码的值必须满足：
  - (1)或者取空值
  - (2)或者等于**S**中某个元组的主码值。**R**与**S**可以是同一个关系

4.用户定义的完整性约束

- 针对某一具体数据库的约束条件，由具体应用要求决定。

关系模式

- 关系模式表现了型的概念，它代表着表的框架
- 关系表现为值的概念,关系实例，一个关系模式下可以建立多个关系，例如在学生关系的关系模式下，可以为全校所有班机各建一个学生表。表是动态的，是数据库中数据的快照
- 关系数据库是关系的集合，其中每个关系都有自己的关系模式

基本表的操作

创建表

基本表的定义（CREATE）

- 格式

```
create table 表名（
列名 数据类型 [default 缺省值] [not null]
[，列名 数据类型 [default 缺省值] [not null]]
                ……
    [，primary key（列名 [，列名]…）]
                [，check（条件）]）
```

修改表

修改基本表定义（ALTER）

- 格式：

```
alter table 表名
        [add 子句]          增加新列
        [drop 子句]         删除列
        [modify 子句]           修改列定义
```

- 示例

```
alter table PROF
        add  LOCATION   char(30)
```

索引

- 示例：

```
create cluster index s-index on S（SN）
```

索引的删除

- 格式

```
drop index 索引名
```

索引定义

- 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，使用索引可快速访问数据库表中的特定信息
- 数据库索引是用于提高数据库表的数据访问速度的，具有以下特点：
  - 避免进行数据库全表的扫描，大多数情况，只需要扫描较少的索引页和数据页，而不是查询所有数据页。而且对于非聚集索引，有时不需要访问数据页即可得到数据。
  - 聚集索引可以避免数据插入操作，集中于表的最后一个数据页面。
  - 在某些情况下，索引可以避免排序操作。
- 索引虽然提高了数据的查询速度，但是为表设置索引要付出代价的：
  - 增加了数据库的存储空间
  - 在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

索引的优缺点

① 优点——创建索引可以大大提高系统的性能

- 第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。
- 第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。
- 第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
- 第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。
- 第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。
  ② 缺点——创建索引可以消耗存储空间，减慢数据库写入速度
- 增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？因为，增加索引也有许多不利的方面。
  - 第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。
  - 第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。
  - 第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

③ 索引的创建原则

- 在经常需要搜索的列上，可以加快搜索的速度；
- 在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；
- 经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；
- 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；
- 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；
- 在经常使用WHERE子句中的列上面创建索引，加快条件的判断速度。

④ 不适合索引的列
一般来说，不应该创建索引的的这些列具有下列特点：

- 第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。
- 第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。
- 第三，对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。
- 第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

常见查询算法

索引是数据结构和算法的结合，常见的查询算法有：
① 顺序查找

- 最基本的查询算法当然是顺序查找（linear search），也就是对比每个元素的方法，不过这种算法在数据量很大时效率是极低的。
- 数据结构：有序或无序队列
- 复杂度：O(n)

② 二分查找（binary search）

- 比顺序查找更快的查询方法应该就是二分查找了，二分查找的原理是查找过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜素过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。
- 数据结构：有序数组
- 复杂度：O(logn)

③ 二叉排序树查找

- 二叉排序树的特点是：
    - 若它的左子树不空，则左子树上所有结点的值均 小于等于 它的根结点的值；
    - 若它的右子树不空，则右子树上所有结点的值均 大于等于 它的根结点的值；
    - 它的左、右子树也分别为二叉排序树。
- 搜索的原理：
    - 若b是空树，则搜索失败；
    - 若x 等于 b的根节点的数据域之值，则查找成功；
    - 若x 小于 b的根节点的数据域之值，则查找左子树；
    - 若x 大于 b的根节点的数据域之值，则查找右子树。

- 数据结构：二叉排序树
- 时间复杂度： O(l o g 2 N)

④ 哈希散列法(哈希表)

- 其原理是首先根据key值和哈希函数创建一个哈希表（散列表），然后根据键值，通过散列函数，定位数据元素位置。
- 数据结构：哈希表
- 时间复杂度：几乎是O(1)，取决于产生冲突的多少。

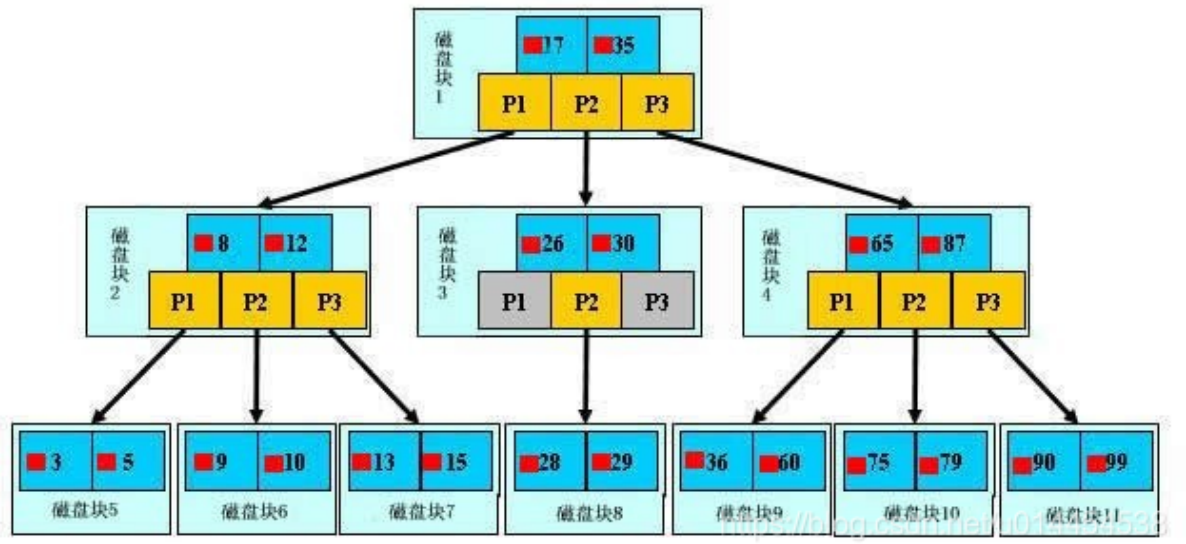
⑤ 分块查找

- 分块查找又称索引顺序查找，它是顺序查找的一种改进方法。其算法思想是将n个数据元素按块有序划分为m块（m ≤ n）。每一块中的结点不必有序，但块与块之间必须按块有序；即第1块中任一元素的关键字都必须小于第2块中任一元素的关键字；而第2块中任一元素又都必须小于第3块中的任一元素，依次类推。
- 算法流程：
  （1）先选取各块中的最大关键字构成一个索引表；
  （2）查找分两个部分：先对索引表进行二分查找或顺序查找，以确定待查记录在哪一块中；然后，在已确定的块中用顺序法进行查找。
- 这种搜索算法每一次比较都使搜索范围缩小一半。它们的查询速度就有了很大的提升，复杂度为O(√n)。

- **总结： 每种查找算法都只能应用于特定的数据结构之上。例如二分查找要求被检索数据有序，而二叉树查找只能应用于二叉查找树上，但是数据本身的组织结构不可能完全满足各种数据结构（例如，理论上不可能同时将两列都按顺序进行组织）。所以，在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。**

B树和B+树

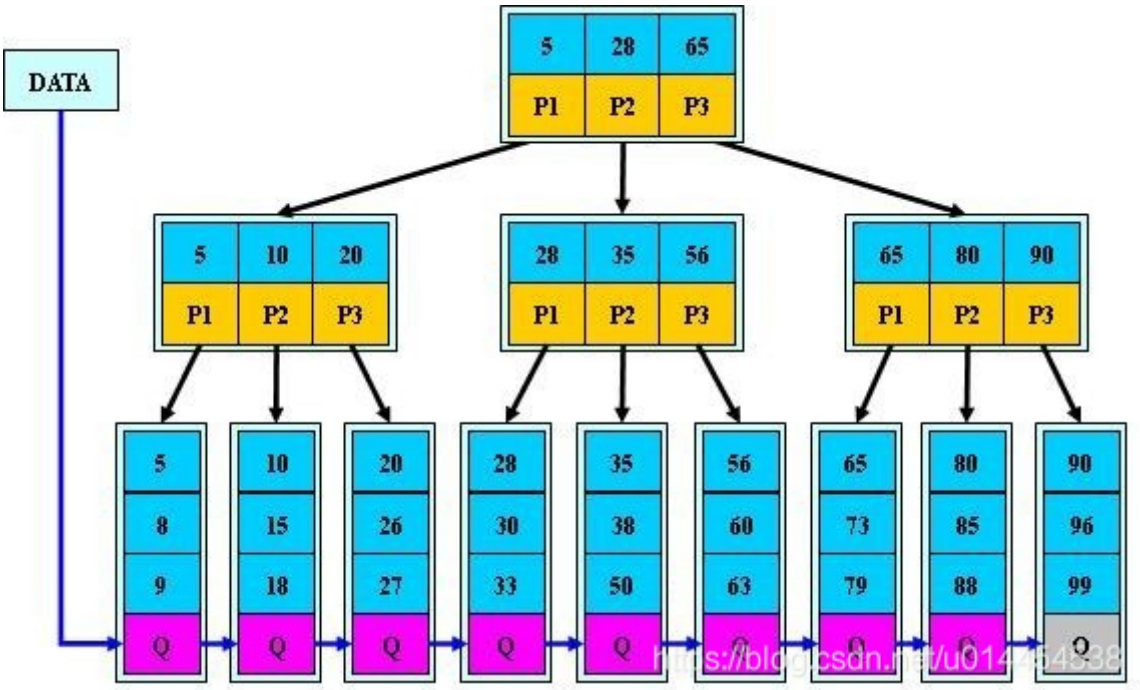
① 关于B树



- 基于上述图片的层级结构，如果我们需要查询数据是29的数据时，发生了些什么呢?
  - 1.找到磁盘块1，加载到内存中（IO一次）。
  - 2.查到在磁盘块3上，加载磁盘块3到内存（IO一次）。
  - 3.查到在磁盘块8上，加载到内存中（IO一次）。

- 总共3次IO操作就查询到了数据，如果没有这个结构，那么最差的情况需要多少次呢？估计需要O（n）次IO操作吧，这就是B树结构的好处，查询次数是指数级别的降低。

② 关于B+树：



B树和B+树的差别：
** (1）叶子节点的不同：**

- B+树的叶子节点包含全部关键字信息，以及这些关键字的指针，而且叶子节点本身按照大小顺序链接。
- B树的叶子节点只包含当前节点的信息，没有全部的信息，而且叶子节点也没有按照顺序链接。

（2）非叶子节点：

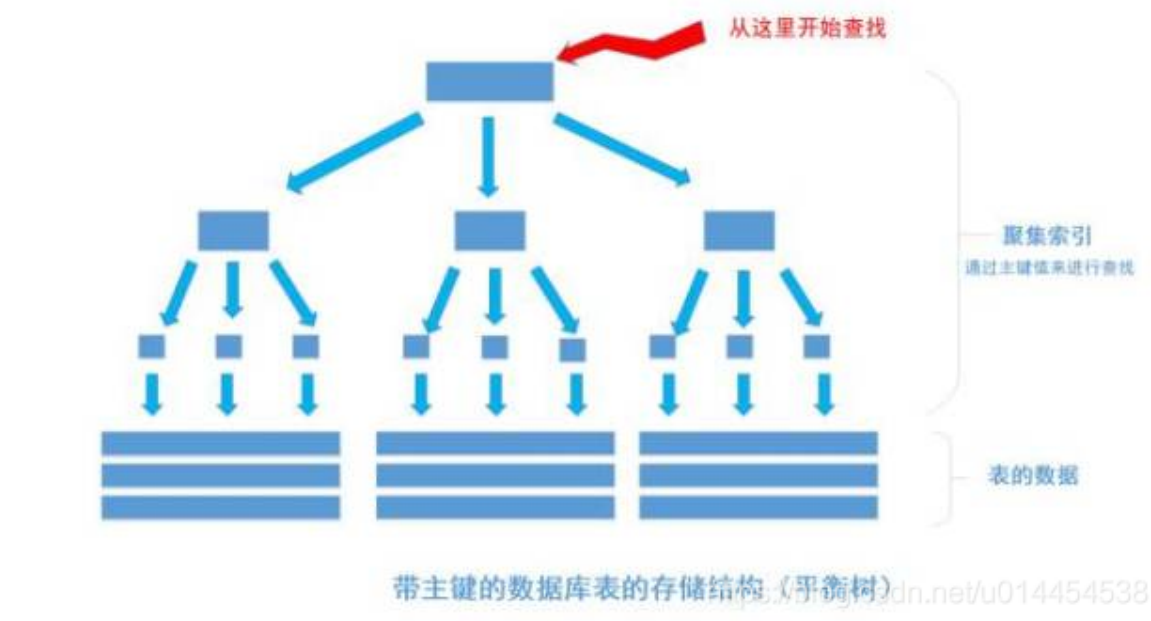
- B+树的非叶子节点可以看做是索引部分，只存储最大或者最小关键字，无法获得具体的数据。
- B树的非叶子节点包含了全部信息，可以获得全部数据。

③ 为什么B+树比B树更适合做文件索引？（数据库索引采用B+树的主要原因是什么？）

- **B+树的磁盘读写代价更低。** B+树的内部结点并没有指向关键字具体信息的指针，其内部结点比B树小，盘块能容纳的结点中关键字数量更多，一次性读入内存中可以查找的关键字也就越多。相对的，IO读写次数也就降低了。而IO读写次数是影响索引检索效率的最大因素。
- **B+树的查询效率更加稳定。**
  ① B树搜索有可能会在非叶子结点结束，越靠近根节点的记录查找时间越短，只要找到关键字即可确定记录的存在。其性能等价于在关键字全集内做一次二分查找。
  ② 而在B+树中，随机检索时，任何关键字的查找都必须走一条从根节点到叶节点的路。所有关键字的查找路径长度相同，导致每一个关键字的查询效率相当。
- **B+树的数据遍历更加方便。**
  ① 不同于B树只适合随机检索，B+树同时支持随机检索和顺序检索。
  ② B+树的叶子节点使用指针顺序连接在一起，只需要遍历叶子节点就可以实现整棵树的遍历。
  ③ 而且在数据库中基于范围的查询是非常频繁的，而B树不支持这样的操作（或者说效率太低）

数据表为什么会使用主键?

- 我们平时建表的时候都会为表加上主键， 在某些关系数据库中， 如果建表时不指定主键，数据库会拒绝建表的语句执行。 事实上， 一个加了主键的表，并不能被称之为「表」。
- 一个没加主键的表，它的数据无序的放置在磁盘存储器上，一行一行的排列的很整齐， 跟我认知中的「表」很接近。
- 如果给表上了主键，那么表在磁盘上的存储结构就由整齐排列的结构转变成了树状结构，也就是上面说的「平衡树」结构。换句话说，就是整个表就变成了一个索引。
- 一定要记住：加了主键以后，整个表变成了一个索引，也就是所谓的「聚簇索引」。 这就是为什么一个表只能有一个主键， 一个表只能有一个「聚簇索引索引」，因为主键的作用就是把「表」的数据格式转换成「索引（平衡树）」的格式放置。

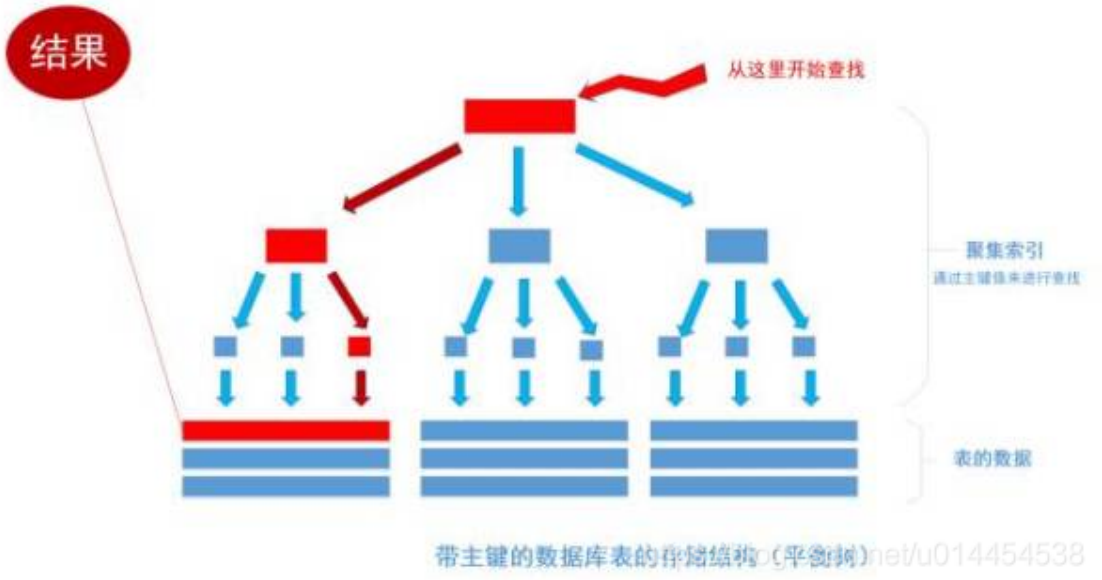


带主键的数据库表的存储结构（平衡树）

- 上图就是带有主键的表（聚簇索引）的结构图。其中树的所有结点（底部除外）的数据都是由主键字段中的数据构成，也就是通常我们指定主键的id字段，最下面部分是真正表中的数据。

```
select * from table where id = 1256;
```

- 首先根据索引定位到1256这个值所在的叶结点，然后再通过叶结点取到id等于1256的数据行，这里不讲解平衡树的运行细节， 但是从上图能看出，树一共有三层， 从根节点至叶节点只需要经过三次查找就能得到结果。如下图



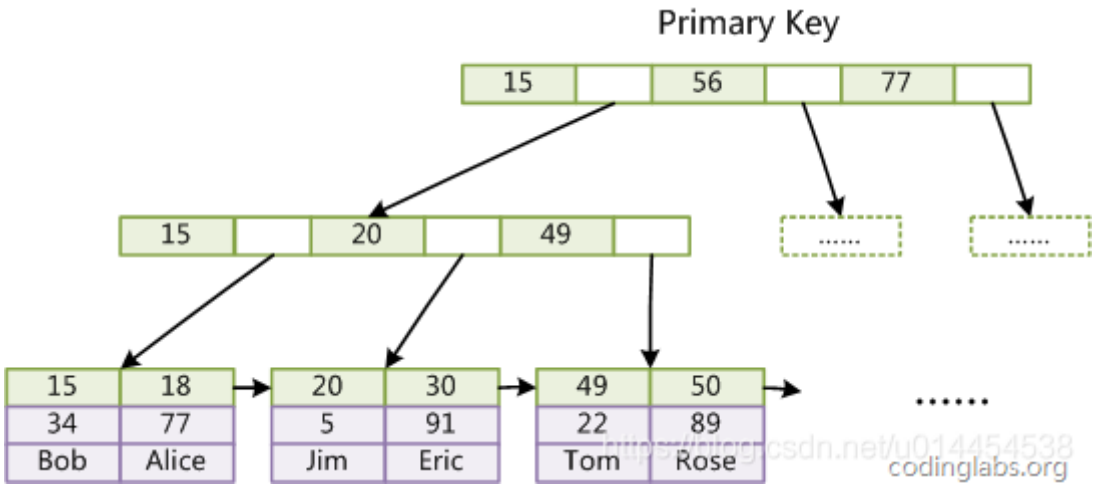
带主键的数据库表的存储结构（平衡树）

- 查找次数：n是记录总数，底数是树的分叉数，结果就是树的层次数。当有一亿条数据的时候，用计算公式表示就是log10^N，N表示记录数：100000000。结果就是查找次数，这里的结果从亿降到了个位数。利用索引会使数据库查询有惊人的性能提升。

$$查找次数 = log_{树的分叉树}^{记录总数}$$

什么是聚簇索引?

- 聚簇索引保证关键字的值相近的元组存储的物理位置也相同（所以字符串类型不宜建立聚簇索引，特别是随机字符串，会使得系统进行大量的移动操作），且一个表只能有一个聚簇索引。
- 因为由存储引擎实现索引，所以，并不是所有的引擎都支持聚簇索引。目前，只有solidDB和InnoDB（MySQL的支持）。
- **InnoDB的特点一： InnoDB的数据文件本身就是索引文件。**
  - 在InnoDB中，使用B+Tree作为索引结构，这棵树的叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。
  - 在InnoDB里，叶子节点data域保存了完整的数据记录。而MyISAM索引文件和数据文件是分离的，索引文件（叶子节点）仅保存数据记录的地址。
  - 注意： 索引的key就是数据表的主键
- 聚簇索引示意图：



SQL语句

重复元组处理

重复元组的处理

- 语法约束

  缺省为保留重复元组，也可用关键字**all**显式指明。

  若要去掉重复元组，可用关键字**distinct**或**unique**指明

- 示例

  找出所有选修课程的学生

  select distinct SNO

  from SC

更名运算

格式

old_name as new_name

为关系和属性重新命名，可出现在select和from子句中

- 关系更名

找出工资比所在系主任工资高的老师姓名及工资

```
select  P1.PNAME, P1.SAL
 from  PROF as P1, PROF as P2, DEPT
where  P1.DNO = DEPT.DNO
 and  DEPT.DEAN = P2.PNO
 and  P1.SAL > P2.SAL
```

字符串操作

列出姓名以“张”打头的教师的所有信息

```
select  *
from  PROF
where PNAME like '张%'
```

列出姓名中含有4个字符以上，且倒数第3个字符是d，倒数第2个字符是_的教师的所有信息.

```
select  *
from   PROF
where  PNAME like  '%__d\____'
```

元组显示顺序

元组显示顺序

- 命令 order by 列名 [asc | desc]
- 示例：按系名升序列出老师姓名，所在系名，同一系中老师按姓名降序排列

```
select DNAME，PNAME
from  PROF，DEPT
where  PROF.DNO = DEPT.DNO
   order by  DNAME  asc，PNAME  desc
```

分组和聚集函数

分组命令

```
group by  列名  [having  条件表达式]
```

group by将表中的元组按指定列上值相等的原则分组，然后在每一分组上使用聚集函数，得到单一值

having则对分组进行选择，只将聚集函数作用到满足条件的分组上

空值测试

空值测试

```
is  [not]  null
```

测试指定列的值是否为空值

示例：找出年龄值为空的老师姓名

```
select  PNAME
from   PROF
where   PAGE is null
```

不可写为 where PAGE = null

集合成员资格

in 子查询

表达式 [not] in （子查询）

判断表达式的值是否在子查询的结果中

示例：列出张军和王红同学的所有信息

```
select  *
from  S
where  SNAME in （‘张军’，‘王红’）
```

```
列出选修了001号和002号课程的学生的学号
select  SNO
from  SC
where  SC.CNO =  ‘001’
    and  SNO  in
            （select  SNO
             from  SC
          where  CNO = ‘002’）
```

数据库与文件系统

文件系统和数据库系统之间的区别：

（1） 文件系统用文件将数据长期保存在外存上，数据库系统用数据库统一存储数据；

（2） 文件系统中的程序和数据有一定的联系，数据库系统中的程序和数据分离；

（3） 文件系统用操作系统中的存取方法对数据进行管理，数据库系统用DBMS统一管理和控制数据；

（4） 文件系统实现以文件为单位的数据共享，数据库系统实现以记录和字段为单位的数据共享。

文件系统和数据库系统之间的联系：

（1） 均为数据组织的管理技术；

（2） 均由数据管理软件管理数据，程序与数据之间用存取方法进行转换；

（3） 数据库系统是在文件系统的基础上发展而来的。

文件系统是操作系统用于明确存储设备（常见的是磁盘，也有基于NAND Flash的固态硬盘）或分区上的文件的方法和数据结构；即在存储设备上组织文件的方法。操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统。

文件系统由三部分组成：文件系统的接口，对对象操纵和管理的软件集合，对象及属性。从系统角度来看，文件系统是对文件存储设备的空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时撤销文件等。