西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY 计算机学院 SCHOOL OF COMPUTER SCIENCE

西北工业大学 | Northwestern Polytechnical University

数据库系统原理 | Database System

第三章 关系数据库标准语言SQL

2024.09

企业进课堂（OceanBase）

本章目录

□索引实现：MySQL索引

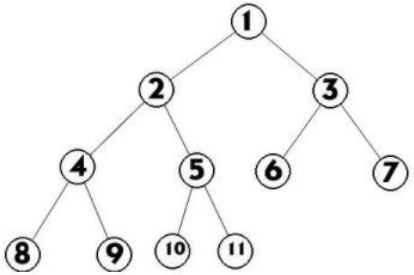
- B-Tree 索引：最常见的索引类型，大部分引擎都支持B树索引。
- HASH 索引：只有Memory引擎支持，使用场景简单。
- R-Tree 索引(空间索引)：空间索引是MyISAM的一种特殊索引类型，主要用于地理空间数据类型。
- Full-text (全文索引)：全文索引也是MyISAM的一种特殊索引类型，主要用于全文索引，InnoDB从MYSQL5.6版本提供对全文索引的支持。

MyISAM、InnoDB引擎、Memory三个常用引擎类型比较

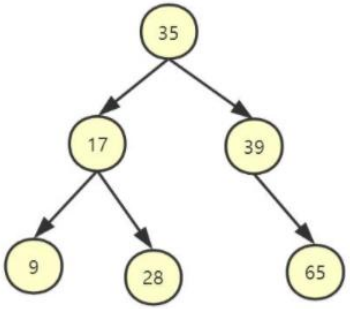
索引	MyISAM引擎	InnoDB引擎	Memory引擎
B-Tree 索引	支持	支持	支持
HASH 索引	不支持	不支持	支持
R-Tree 索引	支持	不支持	不支持
Full-text 索引	不支持	暂不支持	不支持

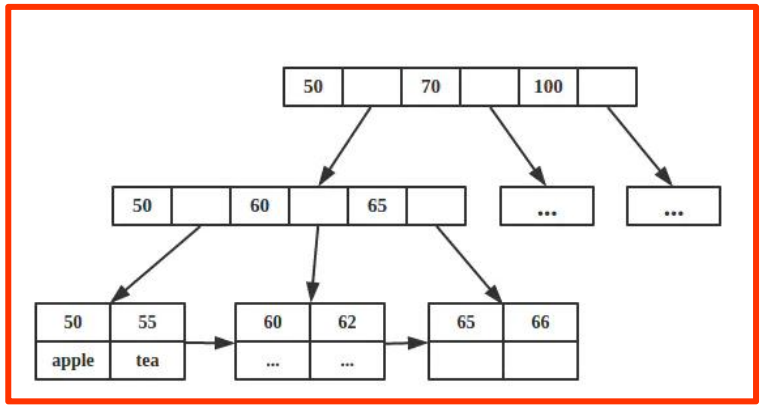
索引的实现 – B+树

二叉树

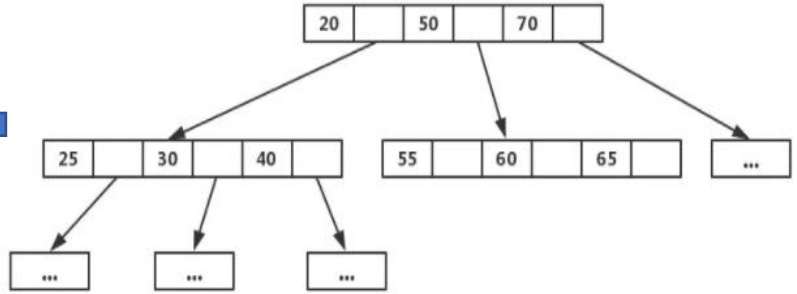


平衡二叉树





B+树（所有数据在叶节点，
叶节点增加链指针）

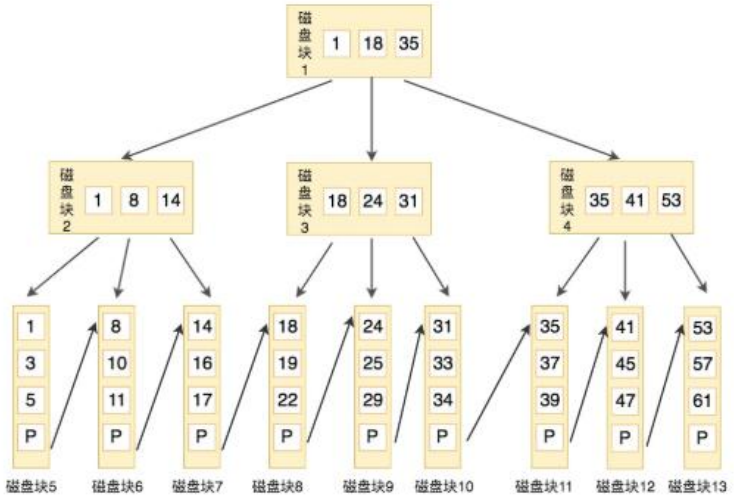


B树（非叶节点和叶节点都存数据）

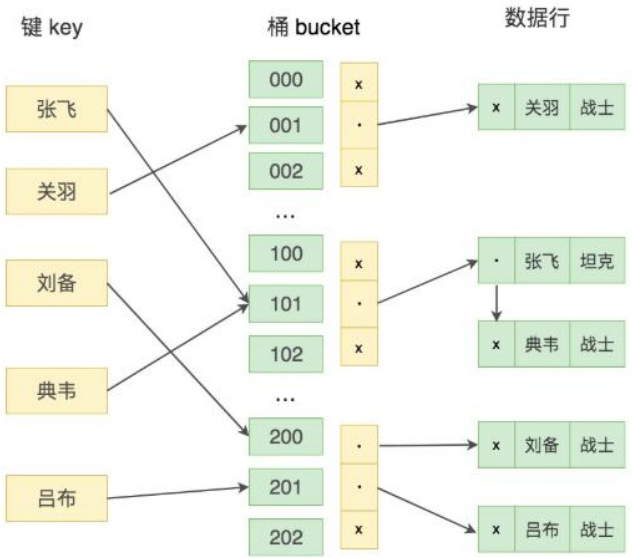
西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

□索引的实现

二叉树、平衡二叉树、B树、B+树



哈希表

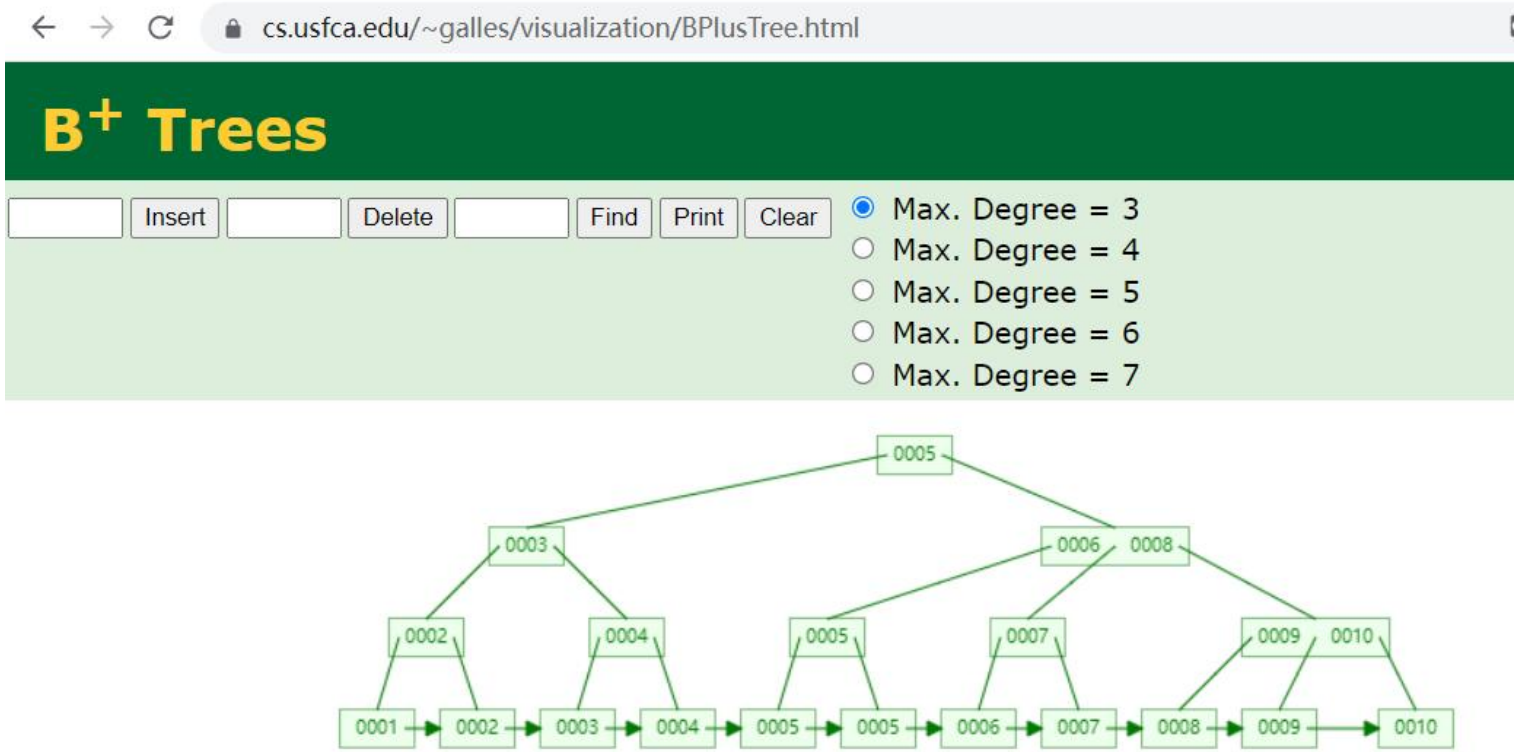


拓展思考题（课后讨论题）：

1. 数据库为什么采用B+树而不是平衡二叉树，又为什么不是B树呢？
2. Hash索引和B+树索引分别适用什么类型的查询？

□索引的实现

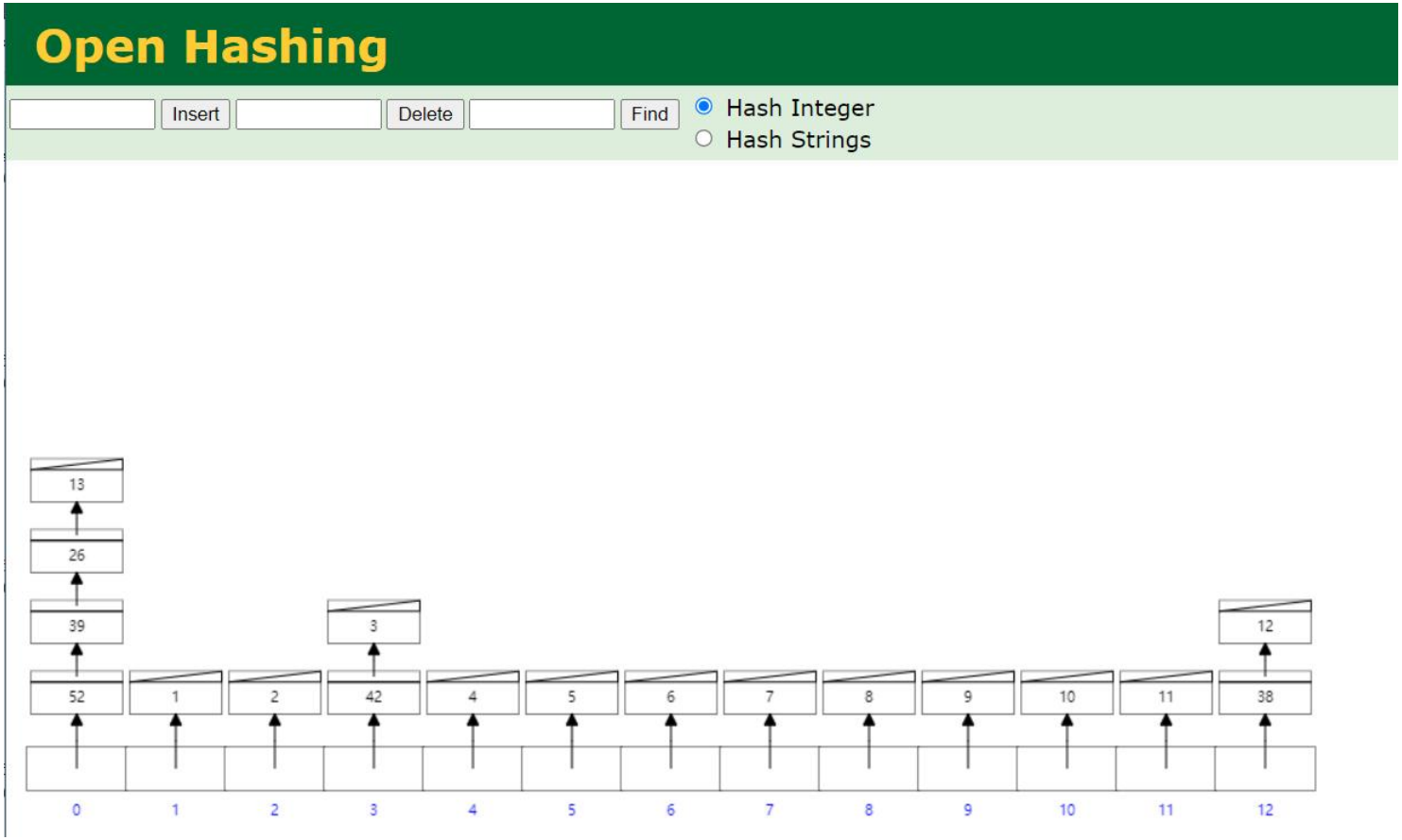
B+树可视化：https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html



西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

□索引的实现

Hash可视化：https://www.cs.usfca.edu/~galles/visualization/OpenHash.html



索引的最左匹配原则

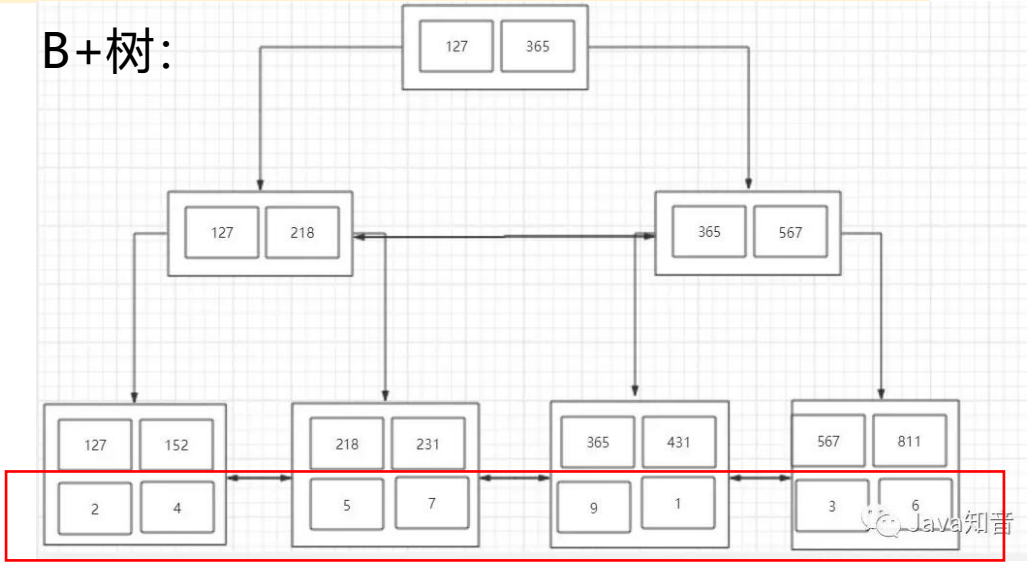
create index idx_obj on user(age asc, height asc, weight asc)

联合索引排序：从左往右依次比较大小。下例中，先比较age的大小，如果age的大小相同，那么比较height的大小，如果height也无法比较大小， 那么就比较weight的大小，最终对这个索引进行排序。

数据：

id	age	height	weight	name
2	1	2	7	小吉
4	1	5	2	小美
5	2	1	8	小尼
7	2	3	1	小白
9	3	6	5	小青
1	4	3	1	小泰
3	5	6	7	小蔡
8	6	2	7	小王

B+树：

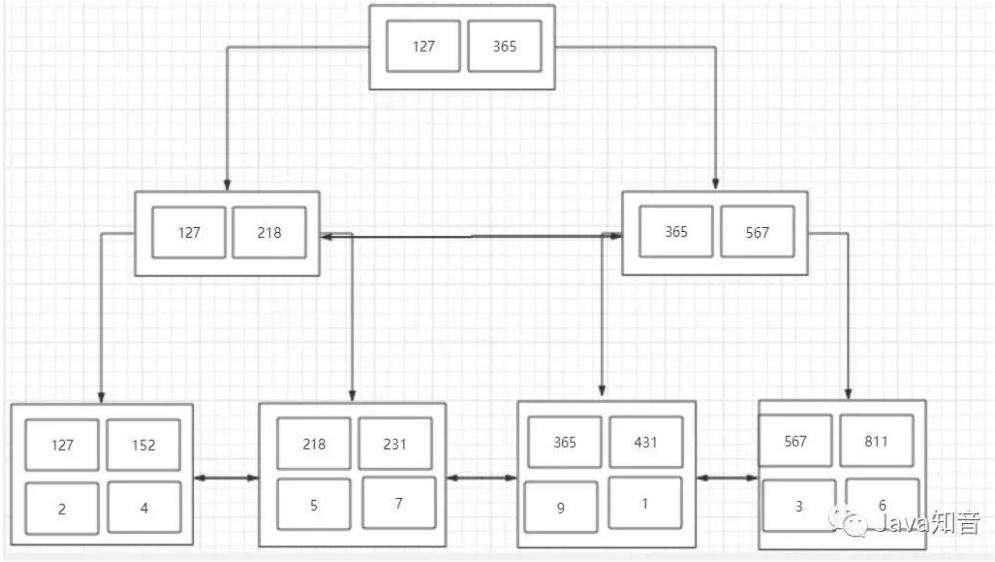


https://www.cnblogs.com/alimayun/p/12170889.html

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

索引的最左匹配原则

create index idx_obj on user(age asc,height asc,weight asc)



只要无法进行排序比较大小的，就无法走联合索引。

127

```
SELECT * FROM user WHERE age=1 and height = 2 and weight = 7
```

```
SELECT * FROM user WHERE height = 2 and weight = 7
```

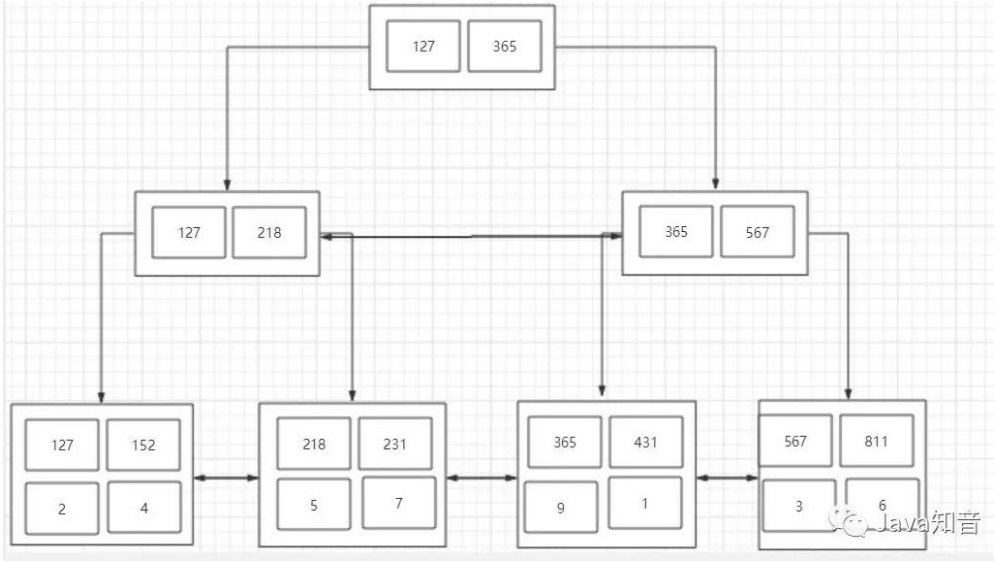
?27
(127/427)=>不同子树

https://www.cnblogs.com/alimayun/p/12170889.html

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

索引的最左匹配原则

create index idx_obj on user(age asc,height asc,weight asc)



以下情况走索引表吗?

```
SELECT * FROM user WHERE age=1 and height = 2
```

✓

```
SELECT * FROM user WHERE age=1 and weight = 7
```

✓

1?7 < 365

```
SELECT * FROM user WHERE age>1
```

✗

https://www.cnblogs.com/alimayun/p/12170889.html

单选题 1分

设置

user_id	user_name	user_gender
110000	student_100000	1
210000	student_200000	1
.....	.....	.....
1010000	Student_1000000	1

请比较耗时从小到大（从快到慢）。

主键：user_id

索引：(user_id, user_name)

1. SELECT user_id, user_name, user_gender FROM user_gender
   WHERE user_id = 900001
2. SELECT user_id, user_name, user_gender FROM user_gender
   WHERE user_name = 'student_890001'

A 1>2

B 2>1

C 1=2

提交

3.3 数据定义 - 补充Explain查看

Explain：查看查询的执行计划（性能分析之必备!）

```
mysql> select * from student;
+------+-------+---------+------------+--------+
| Sno  | Sname | Sgender | Sbirthdate | Smajor |
+------+-------+---------+------------+--------+
| 2001 | 李勇  | 男      | 2000-01-01 | MA     |
| 2002 | 刘晨  | 女      | 2001-02-01 | Is     |
| 2003 | 王敏  | 女      | 1999-10-01 | cs     |
| 2004 | 张立  | 男      | 2001-06-01 | Is     |
+------+-------+---------+------------+--------+
4 rows in set (0.00 sec)

mysql> explain select * from student;
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | student | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    4 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from student where sno="2001";
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table   | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | student | NULL       | const | PRIMARY       | PRIMARY | 34      | const |    1 |   100.00 | NULL  |
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

若期望强制指定查询时使用某个索引:

select sno, sname from userinfo use index(idx_name) where user_id>00

3.3 数据定义 - 补充Explain查看


```
mysql> explain select * from student where sno="2001";
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table   | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | student | NULL       | const | PRIMARY       | PRIMARY | 34      | const |    1 |   100.00 | NULL  |
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
```

列名	描述
id	select查询的序列号，表示查询中执行select子句或表的顺序
select_type	查询类型
table	表名
partitions	匹配的分区信息
type	针对单表的访问方法
possible_keys	可能用到的索引
key	实际使用的索引

列名	描述
key_len	实际使用的索引长度
ref	当使用索引等值查询时，与索引列进行等值匹配的对象信息
rows	预估的需要读取的记录数
filtered	针对预估的需要读取的记录，经过搜索条件过滤后剩余记录条数的百分比
Extra	一些额外的信息

3.3 数据定义 - 补充Explain查看

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

select_type：

- SIMPLE
- PRIMARY
- UNION
- DEPENDENT UNION
- UNION RESULT
- SUBQUERY
- DEPENDENT SUBQUERY
- DERIVED

通常按照执行速度：
system>const>eq_ref>ref>
range>index>all

https://dev.mysql.com/doc/refman/8.0ut.html

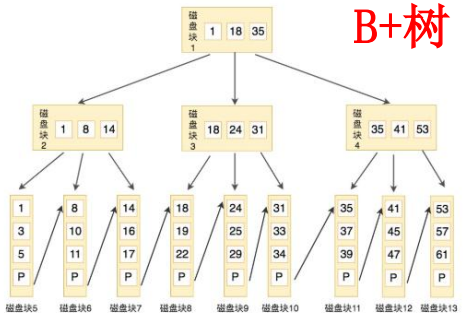
type：

- system
- const
- eq_ref
- ref
- fulltext
- ref_or_null
- index_merge
- unique_subquery
- index_subquery
- range
- index
- all（全表扫描）

单选题 1分

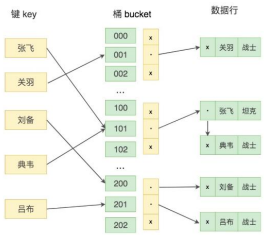
设置

以下说法错误的是?



B+树

A B+树结构的索引更适合范围查询



Hash表

B Hash表结构的索引更适合随机点查

C 对于范围查询，B+树和Hash表性能基本相当

D 以上都不对

提交

本章目录

3.4.3 数据查询 - 嵌套查询

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 1. 带有IN谓词的子查询

例：查询选修了课程名为“信息系统”的学生学号和姓名

```
SELECT sno，sname
FROM   student
 WHERE sno  IN
       (SELECT sno
        FROM   SC
                    WHERE  cno IN
          (SELECT cno
           FROM course
           WHERE cname= ‘信息系统’ ))
```

③ 最后在Student关系中取出Sno和Sname

② 然后在SC关系中找出选修了3号课程的学生学号

① 首先在Course关系中找出“信息系统”的课程号,结果为3号

不相关子查询：即子查询的执行不依赖于父查询的条件。

3.4.3 数据查询 - 嵌套查询

- 2. 带有比较运算符的子查询

例：查询与刘晨同在一系的学生学号和姓名

```
SELECT Sno, Sname, Smajor
FROM Student
WHERE Smajor =
        (SELECT Smajor
        FROM Student
        WHERE Sname='刘晨')
```

当能确切知道内层查询返回单值时，可用比较运算符（>，<，=，>=，<=，!=或< >）。

Sno	Sname	Smajor
2001	刘晨	CS
2002	王勇	CS
2003	...	MA

3.4.3 数据查询 - 嵌套查询

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 相关子查询

例：查询每个学生超过自己所选修课程平均成绩的课程号。

```
select sno,cno
from sc x
where grade >= (select avg(grade)
                from sc y
                where y.sno = x.sno)
```

sno	cno	grade
2001	1	80
2001	2	90
2002	2	70

说说查询的具体过程？

1）外查询取出SC的一个元组x，如2001传给内层查询，执行内查询得到该学生平均成绩85；
2）将平均成绩带入执行外查询，找出2001的选课中比85分高的课程号；
3）取外循环SC中的下一个元组，重复1-3的步骤，直至SC表扫描结束。

3.4.3 数据查询 - 嵌套查询

- 3. 带有ANY（SOME）或ALL谓词的子查询

> ANY(SOME)	大于子查询结果中的某个值
< ANY(SOME)	小于子查询结果中的某个值
>= ANY(SOME)	大于等于子查询结果中的某个值
<= ANY(SOME)	小于等于子查询结果中的某个值
= ANY(SOME)	等于子查询结果中的某个值
!=（或<>）ANY(SOME)	不等于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ALL	小于子查询结果中的所有值
>= ALL	大于等于子查询结果中的所有值
<= ALL	小于等于子查询结果中的所有值
= ALL	等于子查询结果中的所有值
!=（或<>）ALL	不等于子查询结果中的任何一个值

3.4.3 数据查询 - 嵌套查询

- 3. 带有ANY（SOME）或ALL谓词的子查询

(5 < **some** | 0, 5, 6 |) = true (read: 5 < some tuple in the relation)

(5 < **some** | 0, 5 |) = false

(5 = **some** | 0, 5 |) = true

(5 ≠ **some** | 0, 5 |) = true (since 0 ≠ 5)

(= **some**) ≡ **in**

However, (≠ **some**) ≠ **not in**

ANY(SOME)：至少1个

(5 < **all** | 0, 5, 6 |) = false

(5 < **all** | 6, 10 |) = true

(5 = **all** | 4, 5 |) = false

(5 ≠ **all** | 4, 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**

However, (= **all**) ≠ **in**

ALL：所有

3.4.3 数据查询 - 嵌套查询

■ 3. 带有ANY（SOME）或ALL谓词的子查询

例：查询非CS业中比CS专业任意一个学生的年龄小（出生晚）的学生姓名和出生日期

```
SELECT Sname, Sbirthdate
FROM   Student
WHERE Sbirthdate > ANY
                  (SELECT Sbirthdate
                   FROM  Student
                   WHERE SMajor= 'CS')
AND SMajor <> ' CS '
```

1. DBMS执行此查询时，首先处理子查询，找出CS专业中所有学生的年龄，构成一个集合(1999/9/1, 2000/1/8, 2001/9/1)
2. 处理父查询，找所有不是CS专业且出生日期大于集合中任意一个值的学生信息

集函数实现


```
SELECT Sname, Sbirthdate
FROM   Student
WHERE Sbirthdate >
            (SELECT MIN(Sbirthdate)
             FROM  Student
             WHERE SMajor= 'CS')
    AND SMajor <> ' CS '
```

3.4.3 数据查询 - 嵌套查询

- 3. 带有ANY（SOME）或ALL谓词的子查询

  - ANY、ALL谓词与集函数及IN谓词的等价转换关系：

谓词	=	<>/!=	<	<=	>	>=
ANY	IN	-	<MAX	<=MAX	>MIN	>=MIN
ALL	-	NOT IN	<MIN	<=MIN	>MAX	>=MAX

3.4.3 数据查询 - 嵌套查询

- 4. 带有EXISTS谓词的子查询

  - 存在量词∃
  - 带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。
    - 若内层查询结果非空，则where返回true
    - 若内层查询结果为空，则where返回false
  - 由EXISTS引出的子查询，其目标列表达式通常都用*

- 4. 带有EXISTS谓词的子查询

[例] 查询所有选修了1号课程的学生姓名。

```
RANGE   sc x
GET W (student.sname):
        ∃x(x.sno=student.sno ∧x.cno='1')
```

```
SELECT sname
FROM   student
WHERE  EXISTS
    ( SELECT  *
      FROM sc
      WHERE sno =  student.sno AND cno= '1')
```

(1)首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值sno处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表；

(2)然后再取外层表的下一个元组，重复这一过程，直至外层表全部检查完为止。

相关子查询：即子查询的条件与父查询当前值相关。

3.4.3 数据查询 - 嵌套查询

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 4. 带有EXISTS谓词的子查询

NOT EXISTS谓词

- 若内层查询结果非空，则where返回false
- 若内层查询结果为空，则where返回true

例:查询没有选修1号课程的学生姓名。

```
SELECT   sname
FROM     student
WHERE    NOT EXISTS
         ( SELECT *
           FROM   sc
           WHERE  sno = student.sno
                  AND   cno = '1' )
```

```
RANGE   SC X
    GET W (Student.Sname):
    ¬∃X (X.Sno=Student.Sno
          ∧X.Cno='1')
```

4. 带有EXISTS谓词的子查询

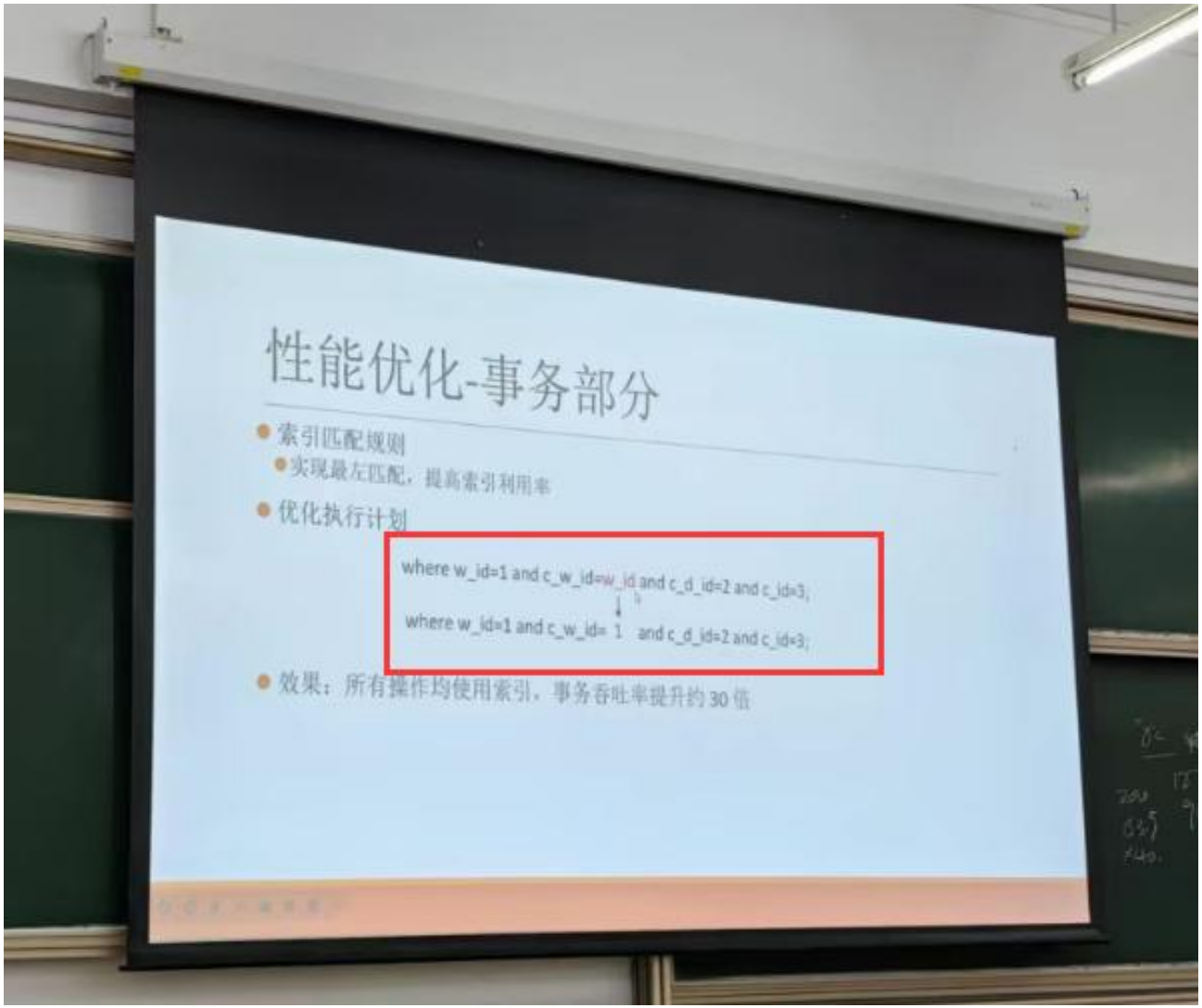
不同形式的查询间的改写与替换

- 一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换
- 所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换

https://blog.csdn.net/OceanBaseGFBK/article/details/126894948?spm=1001.2014.3001.5502

3.4.3 数据查询 - 嵌套查询

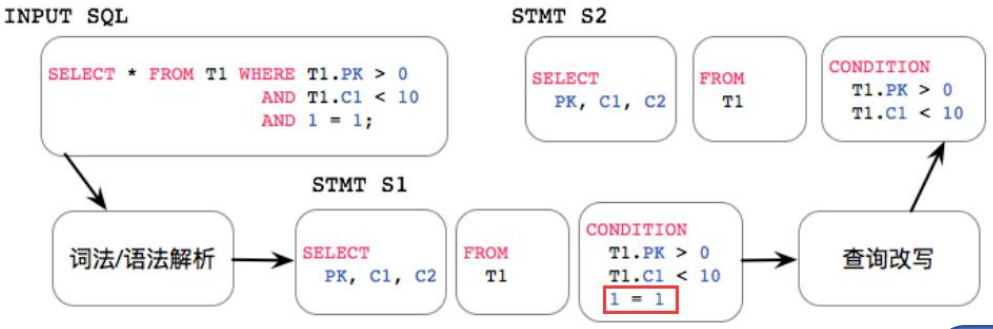
SQL改写



3.4.3 数据查询 - 嵌套查询

SQL改写

1. 基于规则的改写



1）模式匹配
2）执行改写

2. 基于代价的改写

这样改写后的SQL性能一定好吗？

Q1:

```
SELECT * FROM T1 WHERE C1 < 20000 OR C2 < 30 ;
```

=>

Q2:

```
SELECT /*SEL_1*/ * FROM T1 WHERE C1 < 20000
UNION ALL
SELECT /*SEL_2*/ * FROM T1 WHERE C2 < 30 AND LNNVL (C1 < 20000);
```

1）模式匹配
2）执行改写
3）代价验证

引自：https://open.oceanbase.com/blog/10900257?currentPage=1

- 4. 带有EXISTS谓词的子查询

不同形式的查询间的替换--- 举例

[例] 查询与“刘晨”在同一个主修专业学习的学生。
可以理解为：输出那些学生，当他(她)与“刘晨”主修专业相同时。

用带EXISTS谓词的子查询替换：

```
SELECT  Sno, Sname, Smajor
FROM    Student  S1
WHERE   EXISTS
        ( SELECT  *
          FROM  Student S2
          WHERE  S2.Smajor = S1.Smajor  AND
                 S2.Sname = ‘刘晨’ ; )
```

```
SELECT Sno, Sname, Smajor
FROM Student
WHERE Smajor =
      (SELECT Smajor
       FROM Student
       WHERE Sname='刘晨')
```

3.4.3 数据查询 - 嵌套查询

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 4. 带有EXISTS谓词的子查询

[例] Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

- Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

若不存在以下情况OK：有一些生物系的课，该学生课没有选（差集不为空）

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took

3.4.3 数据查询 - 嵌套查询

- 4. 带有EXISTS谓词的子查询

  - 等价变换：

$$(\forall y)p \rightarrow q \equiv \neg(\exists y(\neg(p \rightarrow q))) \equiv \neg(\exists y(\neg(\neg p \vee q)) \equiv \neg \exists y(p \wedge \neg q)$$

- 用EXISTS/NOT EXISTS实现全称量词

  SQL语言中没有全称量词∀（For All）

  可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

$$\forall x\ P(x) \equiv \neg(\exists x(\neg P(x)))$$

4. 带有EXISTS谓词的子查询

[例] 查询选修了全部课程的学生姓名。

方法一

```
RANGE   SC SCX
        Course CX
    GET W (Student.Sname):
        ∀CX ∃SCX (SCX.Sno=Student.Sno ∧SCX.Cno= CX.Cno)
```

- 等价变换：

$$\forall x\ P(x) \equiv \neg(\exists x(\neg P(x)))$$

```
¬∃CX ¬∃SCX (SCX.Sno=Student.Sno ∧SCX.Cno= CX.Cno)
```

含义：没有一门课是他不选的
（不存在一门课，该学生没有选）

3.4.3 数据查询 - 嵌套查询

[例] 查询选修了全部课程的学生姓名。

含义：没有一门课是他不选的
（不存在一门课，该学生没有选）

```
SELECT   Sname
FROM     Student
 WHERE    NOT EXISTS
           ( SELECT    *
             FROM   Course
             WHERE NOT   EXISTS
                    ( SELECT   *
                      FROM   SC
                      WHERE   Sno = Student.Sno AND
                              Cno = Course.Cno   ) ) ;
```

$\neg \exists CX \neg \exists SCX$ (SCX.Sno=Student.Sno ∧SCX.Cno= CX.Cno)

3.4.3 数据查询 - 嵌套查询

[例] 查询选修了全部课程的学生学号。

方法二

```
select   sno
from     sc
group by sno
having count(cno) =
        (select count(cno) from course)
```

having count 的妙用！

练习：查询全部学生都选修的课程课号？

3.4.3 数据查询 - 嵌套查询

■4. 带有EXISTS谓词的子查询

［例］查询至少选修了学生95002选修的全部课程的学生学号。

p：学生95002选修了课程Y
q：学生X也选修了课程Y

$\forall y\ P \rightarrow q$

方法一

```
RANGE Course CX
      SC SCX
      SC SCY

   GET W (Student.Sno):
       ∀CX (∃SCX (SCX.Sno='95002' ∧SCX.Cno= CX.Cno)
         ⇒ ∃SCY (SCY.Sno=Student.sno ∧SCY.Cno= CX.Cno))
```

3.4.3 数据查询 - 嵌套查询

[例] 查询至少选修了学生95002选修的全部课程的学生学号。

$$\forall y\ p \rightarrow q \equiv \neg \exists y(\neg(p \rightarrow q))$$
$$\equiv \neg \exists y\ (\neg\ (\neg\ p \vee q)\ )$$
$$\equiv \neg \exists y\ \ (p \wedge \neg\ q)$$

不存在这样的课程y：
95002选了y，
而学生x没有选

```
GET W (Student.Sno):
    ∀CX (∃SCX (SCX.Sno='95002' ∧SCX.Cno= CX.Cno)
      ⇒ ∃SCY (SCY.Sno=Student.sno ∧SCY.Cno= CX.Cno))
```

```
GET W (Student.Sno):
 ¬∃CX (∃SCX (SCX.Sno='95002' ∧SCX.Cno= CX.Cno)
    ∧ (¬∃SCY (SCY.Sno=Student.sno ∧SCY.Cno= CX.Cno)))
```

3.4.3 数据查询 - 嵌套查询

［例］查询至少选修了学生95002选修的全部课程的学生学号。

不存在这样的课程y：
95002选了y，而学生x没有选

```
SELECT  distinct sno
FROM    sc scx
WHERE   NOT EXISTS
          ( SELECT   *
            FROM   sc scy
            WHERE scy.sno = 95002 and
                  NOT  EXISTS
                   ( SELECT  *
                     FROM   sc scz
                     WHERE   scz.sno = scx.sno AND
                             scz.cno = scy.cno  ) ) ;
```

比从Student表选择效率优化

比从Course表选择效率优化

不存在学生X对该课的选课记录

```
GET W (Student.Sno):
 ¬ ∃ CX (∃SCX (SCX.Sno= '95002'
∧SCX.Cno= CX.Cno)
∧ (¬ ∃SCY (SCY.Sno=Student.sno
∧SCY.Cno= CX.Cno)))
```

3.4.3 数据查询 - 嵌套查询

［例］查询至少选修了学生95002选修的全部课程的学生学号。 方法二

```
select  sno
from    sc
where cno in
      (select cno from sc where sno = 95002)
 group by sno
 having count(cno) =
        (select count(cno) from sc where sno = 95002)
```

95002选修的课程

95002选修的课程的数目

having count 的妙用！

3.4.4 数据查询 - 集合查询

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 集合查询
  - 标准SQL直接支持的集合操作种类
    - 并操作(UNION)
  - 一般商用数据库支持的集合操作种类
    - 并操作(UNION)
    - 交操作(INTERSECT)
    - 差操作(EXCEPT)

注意点：

- 参加UNION操作的各结果表的列数必须相同
- 对应项的数据类型也必须相同

3.4.4 数据查询 - 集合查询

例： 查询选修了课程1或者选修了课程2的学生。

方法一：

```
SELECT Sno  FROM SC  WHERE Cno='1'
UNION
SELECT Sno FROM SC WHERE Cno='2'
```

方法二：

```
SELECT  DISTINCT  Sno
FROM SC
WHERE Cno='1'  OR  Cno= '2'
```

3.4.4 数据查询 - 集合查询

- 集合查询

例： 查询选修了课程1又选修了课程2的学生。

方法一：

```
SELECT Sno
FROM SC
WHERE Cno=’ 1 ’
INTERSECT
SELECT Sno
FROM SC
WHERE Cno= ’ 2 ’
```

方法二：

```
SELECT  Sno
FROM SC
WHERE Cno=’ 1 ’ AND Sno IN
    ( SELECT Sno
      FROM sc
      WHERE cno=‘2’)
```

INTERSECT，EXCEPT：MySQL不支持

3.4.5 数据查询 - 基于派生表查询

子查询：在FROM子句中 => 临时派生表

例： 查询选修了1号课程的学生姓名

```
select sname
from student, (select sno from sc where cno='1') as sc1
where student.sno = sc1.sno
```

也可以写作：

```
with sc1(sno) as
(select sno from sc where cno='1')

select sname
from student, sc1
where student.sno = sc1.sno
```

3.4.5 数据查询 - 基于派生表查询

子查询：还可以在select子句中 => 标量子查询
（Scalar子查询，返回一个具体值）

例： 查询每门课程的选课人数信息。

```
select cno,cname,
       (select count(*) from sc
        where sc.cno = course.cno)
        as num_stud
from course;
```


```
select sc.cno, cname,count(sno) as num_stud
from sc,course
where sc.cno = course.cno
group by sc.cno, cname
```

3.4.5 数据查询

```
1  select cno,cname,
2         (select count(*) from sc
3           where sc.cno = course.cno)
4           as num_stud
5  from course
```

Result Grid | Filter Rows: | Export:

cno	cname	num_stud
1	数据库	1
2	高等数学	2
3	信息系统	2
4	操作系统	0
5	数据结构	0
6	c语言	1

```
1  select  sc.cno, cname, COUNT(sno) as num_stud
2  from sc,course
3  where sc.cno=course.cno
4  group by sc.cno, cname
```

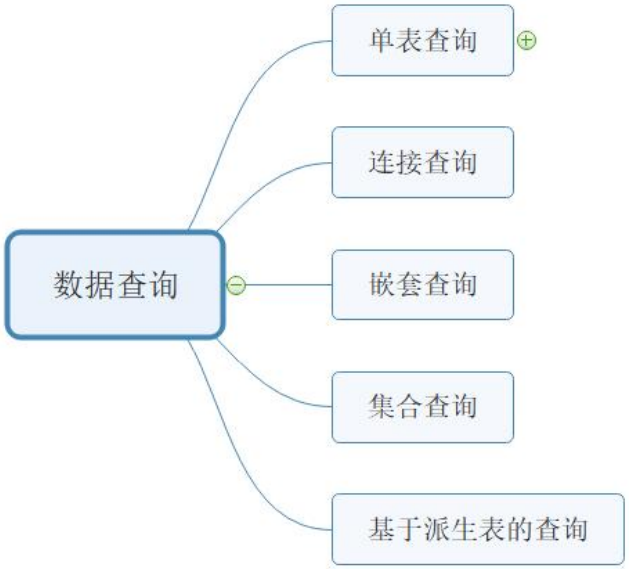
Result Grid | Filter Rows: | Export: | Wrap

cno	cname	num_stud
1	数据库	1
2	高等数学	2
3	信息系统	2
6	c语言	1

为什么
结果不同？

```
select course.cno, cname, count(sno) as
num_stud
from course left join sc
on sc.cno = course.cno
group by sc.cno, cname
order by course.cno;
```

3.4.5 数据查询 小结



综合应用中的部分注意点：

1. 集函数

- 集函数只能用于 SELECT子句和 HAVING短语之中，而不能出现在 WHERE子句中。
- 集函数没有复合功能，平均成绩最高，不能写成MAX(AVG(Grade))。

2. 查询的输出

结果输出列只能取自最外层查询所使用的表，对于子查询中的属性是不能作为最终的输出的。如果输出的属性涉及多个表，则最外层查询需要使用连接查询。

3. ORDER BY子句通常在最后只对最终的结果排序。

3.4 数据查询 - 小结

- 嵌套查询（难点：带EXISTS谓词的查询）
- 综合查询

```
Select [ ALL|DISTINCT ] <输出属性列表>
From          <一个或多个数据库表或视图>
[Where        <查询条件> ]
[Group By     <分组条件>[HAVING <条件表达式>] ]
[Order By     <结果排序> [ASC|DESC]
```

本章重点与难点

- 重点
  - 数据查询
  - 数据更新
  - 视图
- 难点
  - 数据查询（嵌套查询）

第三章和第五章书面作业：

第三章：第3题、第6题、第8题

第五章：第7题