SQL 语句分类

- **数据定义语言： DDL（Data Definition Language）**：CREATE、DROP、ALTER，主要是对表结构、视图、索引等的操作
- **数据查询语言：DQL（Data Query Language）**：SELECT
- **数据操纵语言：DML（Data Manipulation Language）**：INSERT、DELETE、UPDATE
- **数据控制语言：DCL（Data Control Language）**：GRANT、REVOKE、COMMIT、ROLLBACK

一、数据定义

关系数据库系统支持三级模式结构，其模式，外模式，内模式中的基本对象有模式、表、视图和索引，所以SQL的数据定义功能包括模式定义、表定义、视图和索引的定义

1. 模式的定义与删除

① 模式定义

```
create schema 模式名 authorization 用户名;
```

示例：

为用户WANG定义一个 学生-课程 模式 S-T

```
create schema "S-T" authorization WANG;
```

定义模式实际上就是定义了一个命名空间，可以在这个空间的基础上进一步定义数据库对象，基本表，视图，索引等

示例：

为用户 ZHANG 创建一个模式 TEST，并且在其中定义一个表 table1

```
create schema "TEST" authorization ZHANG
create table table1(col1 smallint,
                    col2 int,
                    col3 char(20)
                    );
```

② 删除模式

```
drop schema 模式名 CASCADE | RESTRICT
```

CASCADE | RESTRICT 必选其一

- `CASCADE` 表示删除模式的同时会删除该模式下的所有数据库对象
- `RESTRICT` 表示若该模式下存在数据库对象，则拒绝执行删除操作

2. 基本表的定义、删除、修改

> 数据库的创建和使用：
>
> CREATE DATABASE test;
> USE test;

① 定义基本表 CREATE

```
CREATE TABLE mytable (
  # int 类型，不为空，自增
  id INT NOT NULL AUTO_INCREMENT,
  # int 类型，不可为空，默认值为 1，不为空
  col1 INT NOT NULL DEFAULT 1,
  # 变长字符串类型，最长为 45 个字符，可以为空
  col2 VARCHAR(45) NULL,
  # 日期类型，可为空
  col3 DATE NULL,
  # 设置主键为 id
  PRIMARY KEY (`id`)
  # 定义外键,被参照表是mytable2
  FOREIGN KEY(col1) REFERENCES mytable2(col1)
);
```

SQL标准支持多种数据类型：

表 3.4 数 据 类 型

数据类型	含义
CHAR(*n*), CHARACTER(*n*)	长度为 *n* 的定长字符串
VARCHAR(*n*), CHARACTERVARYING(*n*)	最大长度为 *n* 的变长字符串
CLOB	字符串大对象
BLOB	二进制大对象
INT，INTEGER	长整数（4 字节）
SMALLINT	短整数（2 字节）
BIGINT	大整数（8 字节）
NUMERIC(*p*, *d*)	定点数，由 *p* 位数字（不包括符号、小数点）组成，小数点后面有 *d* 位数字
DECIMAL(*p*, *d*), DEC(*p*, *d*)	同 NUMERIC
REAL	取决于机器精度的单精度浮点数
DOUBLE PRECISION	取决于机器精度的双精度浮点数
FLOAT(*n*)	可选精度的浮点数，精度至少为 *n* 位数字
BOOLEAN	逻辑布尔量
DATE	日期，包含年、月、日，格式为 YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为 HH:MM:SS
TIMESTAMP	时间戳类型
INTERVAL	时间间隔类型

② 修改基本表 ALTER

- 向Student表中添加入学时间列

```
alter table student add entrance_date DATE;
```

- 将年龄的数据类型由字符型转为整数

```
alter table student alter column age INT;
```

- 增加课程名称必须取唯一值的约束条件

```
alter table student add unique(cname);
```

③ 删除基本表

```
DROP TABLE 表名 [RESTRICT | CASCADE];
```

默认是 RESTRICT

查询表结构

desc + table名

3. 索引的建立和删除

① 建立索引

```
create [unique][cluster] index 索引名
on 表名 (列名[次序], 列名[次序]...)
```

unique 表示此索引的每一个索引值只对应唯一的数据记录

cluster 表示该索引是聚集索引

```
create unique index sno_index on student(sno);
create unique index cno_index on cource(cno);
# sc表按学号升序和课程号降序建立唯一索引
create unique index sc_index on sc(sno asc, cno desc);
```

② 修改索引

```
alter index 旧索引名 rename to 新索引名
```

③ 删除索引

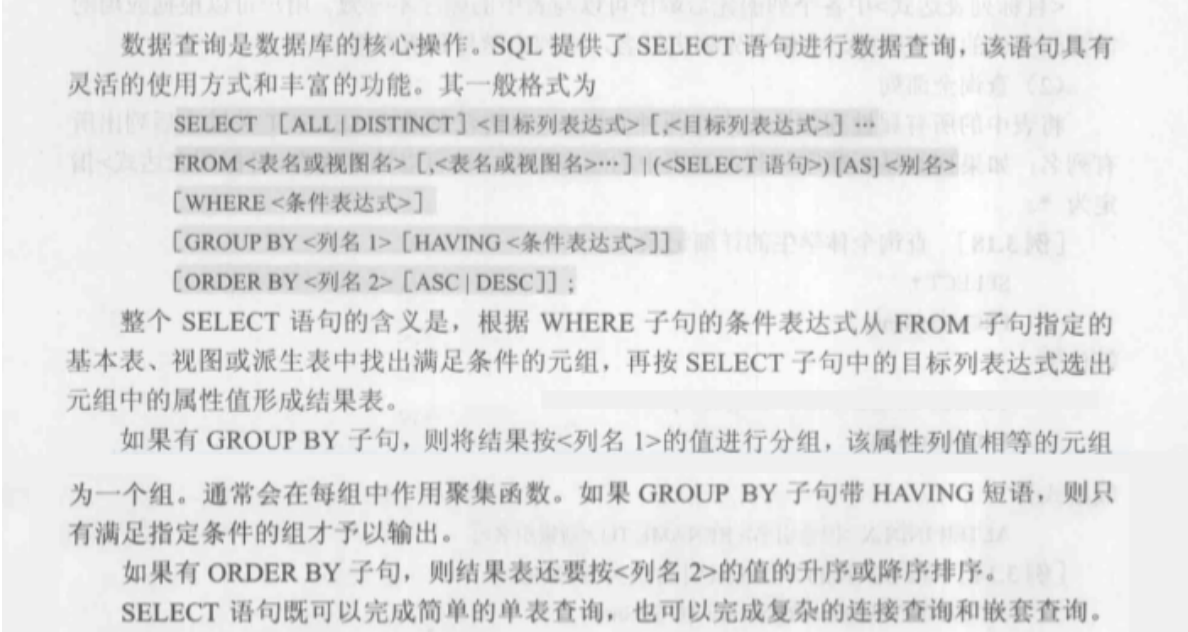
```
drop index 索引名
```

4. 数据字典

数据字典是关系数据库管理系统内部的一组系统表，它**记录了数据库中所有的定义信息**，包括关系模式定义、视图定义、索引定义、完整性约束定义、各类用户对数据库的操作权限、统计信息等。**关系数据库管理系统在执行SQL的数据定义语句时，实际上就是在更新数据字典中的相应信息**

二、数据查询

select 语句的一般格式：



数据查询是数据库的核心操作。SQL 提供了 SELECT 语句进行数据查询，该语句具有灵活的使用方式和丰富的功能。其一般格式为

SELECT [ALL | DISTINCT] <目标列表达式> [,<目标列表达式>] …
FROM <表名或视图名> [,<表名或视图名>…] | (<SELECT 语句>) [AS] <别名>
[WHERE <条件表达式>]
[GROUP BY <列名 1> [HAVING <条件表达式>]]
[ORDER BY <列名 2> [ASC | DESC]];

整个 SELECT 语句的含义是，根据 WHERE 子句的条件表达式从 FROM 子句指定的基本表、视图或派生表中找出满足条件的元组，再按 SELECT 子句中的目标列表达式选出元组中的属性值形成结果表。

如果有 GROUP BY 子句，则将结果按<列名 1>的值进行分组，该属性列值相等的元组为一个组。通常会在每组中作用聚集函数。如果 GROUP BY 子句带 HAVING 短语，则只有满足指定条件的组才予以输出。

如果有 ORDER BY 子句，则结果表还要按<列名 2>的值的升序或降序排序。

SELECT 语句既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。

> 以下所有示例全都使用下述三个表
>
> - Student(Sno,Sname, Sage, SDept) 学生表
> - Cource (Cno, Cname) 课程表
> - SC(Sno,Cno,Grade) 选修表

1. 单表查询

单表查询是指仅涉及一个表的查询

① 查询表中的若干列

查询全体学生的学号和姓名

```
select Sno, Sname
from Student;
```

目标列表达式 也可以是表达式

```
select Sname, 2020 - age
from Student;
```

用户可以为查询的列定义别名

```
select Sname, 2020 - age Birthday
from Student;
```

② 选择表中的若干元组

消除取值重复的行 distinct

使用 distinct 关键字 去除重复记录

```
select distinct Sno
from SC;
```

查询满足条件的行

使用 where 子句，常用的查询条件如下：

表 3.6 常用的查询条件

查询条件	谓词
比较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空值	IS NULL，IS NOT NULL
多重条件（逻辑运算）	AND，OR，NOT

- **比较**

```
select sname
from student
where age <= 20;
```

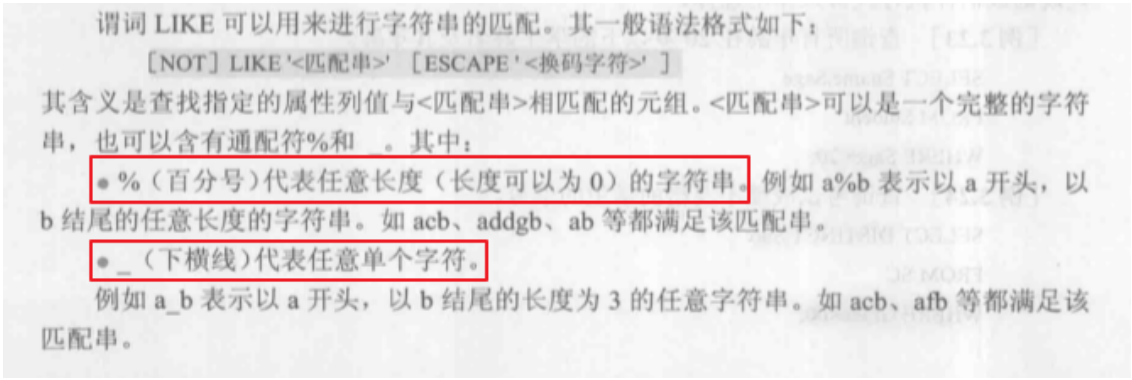
- **确定范围**

```
# 查询年龄在20-33岁之间的学生姓名和年龄
select sname,age
from student
where age between 20 and 33;
```

- **确定集合**

```
# 查询计算机系和数据系的学生姓名和性别
select sname,gender
from student
where dept in ('CS','Math');
```

- **字符匹配**



谓词 LIKE 可以用来进行字符串的匹配。其一般语法格式如下：

[NOT] LIKE '<匹配串>' [ESCAPE '<换码字符>']

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串，也可以含有通配符%和 _。其中：

- %（百分号）代表任意长度（长度可以为 0）的字符串。例如 a%b 表示以 a 开头，以 b 结尾的任意长度的字符串。如 acb、addgb、ab 等都满足该匹配串。
- _（下横线）代表任意单个字符。

例如 a_b 表示以 a 开头，以 b 结尾的长度为 3 的任意字符串。如 acb、afb 等都满足该匹配串。

```sql
# 查询姓欧阳且全名为三个汉字的学生的姓名
select sname
from student
where sname like '欧阳_';
```

- **涉及空值的查询**

```sql
# 查询所有有成绩的学生姓名
select sname
from student
where grade is not null;
```

- **多重条件查询**

  and 和 or 可用来连接多个查询条件，and 的优先级高于 or，不过可以用 括号来改变优先级

```sql
# 查询计算机系年龄20以下的学生姓名
select sname
from student
where sdept = 'CS' and sage < 20;
```

③ order by 子句

- desc 降序
- asc 升序 默认

```sql
# 院系按升序排，年龄按降序排
select *
from student
order by sdept,sage desc;
```

④ top

```sql
# 查询成绩第一的学生姓名
select Sname
from Student
where Sgrade = (
    select top 1 Sgrade
    from Student
    order by Sgrade desc
);
```

⑤ 聚集函数

函数名	功能
COUNT	对元组计数
TOTAL	求总和
MAX	求最大值
MIN	求最小值
AVG	求平均值

```
# 查询学生总人数
select count(*)
from student;

# 查询选修了课程的学生人数(学生每选一门可都会在选修表中有记录)
select count(distinct Sno) as numbers
from sc;
```

⑥ group by 子句

group by 分组：把具有相同的数据值的行放在同一组中。

分组后聚集函数将作用于每一组，即每一组都有一个聚集函数值

```
# 求各个课程号及相应的选课人数
select Cno, count(Sno)
from sc
group by Cno;
```

如果分组后还需要对这些组进行过滤，则使用 `HAVING` 短语

```
# 查询选修了三门以上课程的学生学号
select Sno
from sc
group by sno
having count(*) > 3;
```

```
# 有两个表Study(sno,cno)和Student(sno,sname)，查询选修了2或3门课的学生
select * from Student s
where s.sno in(
    select stu.sno from Studty stu
    group by stu.sno
    having count(*) >= 2
);
```

`WHERE` 过滤行，`HAVING` 过滤分组，行过滤应当先于分组过滤。

> having 与 where 功能、用法相同，**执行时机不同。**
>
> where 在开始时执行检测数据，对原数据进行过滤。

> **having 对筛选出的结果再次进行过滤。**
>
> having 字段必须是查询出来的，where 字段必须是数据表存在的。
>
> **where 不可以使用字段的别名，having 可以。因为执行 WHERE 代码时，可能尚未确定列值。**
>
> where 不可以使用聚集函数。一般需用聚集函数才会用 having
>
> SQL标准要求`HAVING 必须引用 GROUP BY 子句中的列或用于合计函数中的列`。

⚠ GROUP BY 子句出现在 WHERE 子句之后，ORDER BY 子句之前

```
SELECT col, COUNT(*) AS num
FROM mytable
where col > 2
GROUP BY col
ORDER BY num;
```

⑤ 分页

LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。

- 如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)

```
SELECT * FROM table LIMIT 5,10; // 检索记录行 6-15
```

- 为了检索从某一个偏移量到记录集的结束所有的记录行，可以指定第二个参数为 -1：

```
SELECT * FROM table LIMIT 95,-1; // 检索记录行 96-last.
```

- 如果只给定一个参数，它表示返回最大的记录行数目：

```
SELECT * FROM table LIMIT 5; //检索前 5 个记录行
```

- limit 和 offset 连用

```
select * from table limit 2 offset 1; //跳过前面1条数据，检索2条数据
```

2. 连接查询

若一个查询同时涉及两个以上的表，则称为连接查询

① 等值与非等值连接查询

```sql
# 查序每个学生及选修课的情况
select Student.*, SC.*
from Student, SC
where Student.sno = SC.sno;

# 查询选修2号课程且成绩在90分以上的所有学生的学号和姓名
select Student.sno,sname
from Student,SC
where Student.sno = SC.sno and
    SC.cno = 2 and
    SC.grade > 90;
```

② 自身连接

一个表与自己进行连接

示例:

［例 3.52］ 查询每一门课的间接先修课（即先修课的先修课）。

在 Course 表中只有每门课的直接先修课信息，而没有先修课的先修课。要得到这个信息，必须先对一门课找到其先修课，再按此先修课的课程号查找它的先修课程。这就要将 Course 表与其自身连接。

为此，要为 Course 表取两个别名，一个是 FIRST，另一个是 SECOND。

FIRST 表（Course 表）

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

SECOND 表（Course 表）

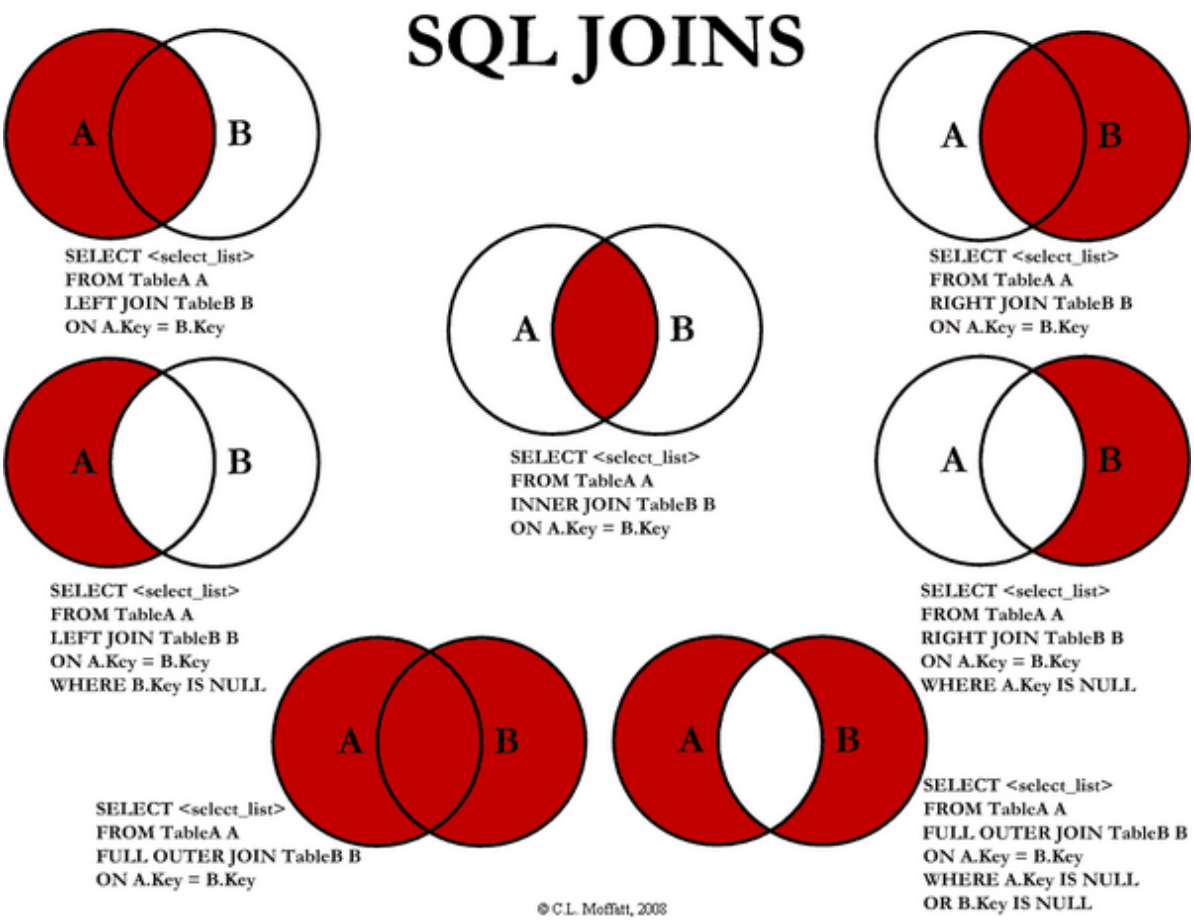
Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

```sql
select FIRST.Cno,SECOND.Cpno
from Cource FIRST, Cource Second
where FIRST.Cpno = SECOND.Cno;
```

③ 外连接

- **内连接 inner join(join):** 只连接匹配的行（默认）
- **强制驱动 straight_join**：功能同join类似，但能让左边的表来驱动右边的表，能改表优化器对于联表查询的执行顺序。
- **左外连接 left join:** 包含左边表的全部行（不管右边的表中是否存在与它们匹配的行），以及右边表中全部匹配的行
- **右外连接 right join:** 包含右边表的全部行（不管左边的表中是否存在与它们匹配的行），以及左边表中全部匹配的行
- **全外连接 outer join:** 包含左、右两个表的全部行，不管另外一边的表中是否存在与它们匹配的行。
- **交叉连接 cross join:** 生成笛卡尔积 - 它不使用任何匹配或者选取条件，而是直接将一个数据源中的每个行与另一个数据源的每个行都一一匹配

```sql
select b.s_id,b.s_name,ROUND(AVG(a.s_score),2) as avg_score
from student b
left join score a on b.s_id = a.s_id
GROUP BY b.s_id,b.s_name HAVING avg_score >=60;
```



④ 多表连接

两个以上的表进行连接

```sql
# 查询每个学生的学号，姓名，选修的课程名及成绩
select Student.Sno,Sname,Cname,Grade
from Student,Cource,SC
where Student.Sno = SC.Sno AND
    SC.Cno = Cource.Cno;
```

3. 嵌套查询

在 SQL 语言中，一个 select-from-where 语句称为一个查询块。

将一个查询块套在另一个查询块的 where 子句或 HAVING 短语的条件中的查询称为嵌套查询

① 带有 in 谓词的子查询

```sql
# 查找与小明所在同一个系的学生
select Sno,Sname,Sdept
from  Student
where Sdept in
    (Select Sdept
    From Student
    Where Sname = "小明");
```

② 带有比较运算符的子查询

```sql
# 找出每个学生超过他自己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);
```

③ 带有 ANY(SOME)或 ALL 谓词的子查询

子查询返回单值时可以用比较运算符，但返回多值时要用 ANY（有的系统用SOME）或 ALL 谓词修饰符。而使用ANY 或 ALL谓词时必须同时使用比较运算符。

>ANY	大于子查询结果中的某个值
>ALL	大于子查询结果中的所有值
<ANY	小于子查询结果中的某个值
<ALL	小于子查询结果中的所有值
>=ANY	大于等于子查询结果中的某个值
>=ALL	大于等于子查询结果中的所有值
<=ANY	小于等于子查询结果中的某个值
<=ALL	小于等于子查询结果中的所有值
=ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=（或<>）ANY	不等于子查询结果中的某个值
!=（或<>）ALL	不等于子查询结果中的任何一个值

示例：

```sql
# 查询非计算机科学系中比计算机科学系任意一个学生年龄小的学生姓名和年龄
select Sname,Sage
from Student
where Sage < ANY(select Sage
                 from Student
                 where Sdept = 'CS')
      and
      Sdept <> ‘CS';
```

④ 带有 EXISTS 谓词的子查询

`EXISTS` 代表存在，带有该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

先查询主内容，然后，根据表的每一条记录，依次判断where后面的条件是否成立

- exists 引导的内层查询如果 能查出数据，则继续外层查询
- not exists 引导的内层查询如果 查不出数据，则继续外层查询

> **exists 是先做外查询**，与 in 相反
>
> 顺序执行，如果exists 的查询结果为真，则将最外层的查询结果添加进最终结果集。对外表进行循环

示例：

```
# 查询所有选修了1号课程的学生姓名
select Sname
from Student
where exists(
    select *
    from SC
    where Sno = Student.Sno
    and Cno = 1
);
```

由exists引出的子查询，其目标列表达式通常都用 * ，因为该子查询只返回 true 或 false，给出列名无实际意义

```
# 查询选修了全部课程的学生姓名
select Sname
from Student
where not exists(
    # 首先我们要直到一共有哪些课程
    select *
    from Cource
    where not exists(
        # 其次，我们需要统计选修了所有课程的学生号
        select *
        from SC
        where Sno = Student.Sno
        and
        Cno = Cource.Cno
    )
);
```

由于没有全程量词，可将题目的意思转化为 没有一门课程是他不选修的

exists 可以理解为一个循环

```
for(循环从Student表拿一行学生数据){
    for(循环从Course表拿一行课程信息){
        for(循环在SC表拿一行进行比对){
            SC表中的这条数据判断：
              SC.Sno == Student.Sno ，  SC.Cno == Course.Cno;
            /*是否SC表中的学号 = Student表中的学号 且
              SC表中的Cno = Course表中的Cno*/
        }
    }
}
```

```
# 查询至少选修了学生001选修的全部课程的学生号码
select distinct Sno
# 代表学生X的表
from SC SCX
where not exists(
    select *
    from SC SCY
    where SCY.Sno = '001'
    and
    not exists(
        select *
        from SC SCZ
        # 匹配学号
        where SCZ.Sno = SCX.Sno
        and
        # 001选修了该课程
        SCZ.Cno = SCY.Cno
    )
);
```

翻译为：不存在这样的课程y，001选修了，而学生x没有选修

exists 和 in 的区别

> 例如：
>
> select * from A
> where id in(select id from B)
>
> **exists()适合B表比A表数据大的情况**
>
> **当A表数据与B表数据一样大时, in与exists效率差不多,可任选一个使用.**
>
> **如果查询语句使用了not in 那么内外表都进行全表扫描，没有用到索引；而not exists 的子查询依然能用到表上的索引。所以无论那个表大，用not exists都比not in要快。**

4. 集合查询

select 语句的查询结果是元组的集合，所以多个select语句的结果可进行集合操作。集合操作主要包括

- `并 UNION`：对两个结果集进行并集操作, 不包括重复行,相当于distinct, 同时进行默认规则的排序;
- union all： 对两个结果集进行并集操作, 包括重复行, 即所有的结果全部显示, 不管是不是重复;
- `交 INTERSECT`
- `差 EXCEPT`

参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同

```
# 查询计算机系的学生及年龄不大于19的学生
select *
from Student
where Sdept = 'CS'
UNION
select *
from Student
where Sage <= 19;

# 也可以用INTERSECT
select *
from Student
where Sdept = 'CS'
INTERSECT
select *
from Student
where Sage <= 19;

# 也可以用EXCEPT
select *
from Student
where Sdept = 'CS'
EXCEPT
select *
from Student
where Sage > 19;
```

5. 基于子查询的查询

子查询不仅可以出现在where子句中，还可以出现在子句中，这时子查询生成的临时派生表成为主查询的查询对象 (必须为派生关系置顶一个别名)

```
# 找出每个学生超过自己选修课程平均成绩的课程号
select Sno,Cno
from SC,(select Sno,AVG(Grade) from SC group by Sno)
        as Avg_SC(avg_sno,avg_grade);
```

6. 常用函数

① case when

聚集修改值

```
select case
        when age <25 OR age is NULL then "25岁以下"
        when age >=25 then "25岁及以上"
        end age_cut,count(*) number
from user_profile
group by age_cut
```

② day，month，year，TIMESTAMPDIFF

```
select day(date) day, count(*) question_cnt
from question_practice_detail
where year(date)=2021 and month(date)=08
group by day
```

TIMESTAMPDIFF函数，有参数设置，可以精确到天（DAY）、小时（HOUR），分钟（MINUTE）和秒（SECOND）

```
select TIMESTAMPDIFF(SECOND, '2018-03-20 09:00:00', '2018-03-22 10:00:00');
```

③ substring_index

substring_index函数适合用分割某一个字符,相当于split,然后截取某一个值

```
select device_id,
substring_index(blog_url,'/',-1) user_name
from user_submit
```

④ rank() over, partition by

```
select device_id, university, gpa
from (
    select *, rank() over (
        partition by university order by gpa
    ) rank_num
    from user_profile
) up
where up.rank_num=1
```

⑤ if()

```
select a.device_id, university,
count(if(month(b.date)=8,1,null)),
count(if(month(b.date)=8 and b.result='right',1,null))
from user_profile a
left join question_practice_detail b
on a.device_id=b.device_id
where university="复旦大学"
group by device_id
```

⑥ 数学

```
# 四舍五入
Round(number,2)
```

⑦ 窗口函数

在mysql8.0中有相关的内置函数，而且考虑了各种排名问题：

- row_number(): 同薪不同名，相当于行号，例如3000、2000、2000、1000排名后为1、2、3、4
- rank(): 同薪同名，有跳级，例如3000、2000、2000、1000排名后为1、2、2、4
- dense_rank(): 同薪同名，无跳级，例如3000、2000、2000、1000排名后为1、2、2、3
- ntile(): 分桶排名，即首先按桶的个数分出第一二三桶，然后各桶内从1排名，实际不是很常用

这三个函数必须要要与其搭档over()配套使用，over()中的参数常见的有两个，分别是

- partition by，按某字段切分
- order by，与常规order by用法一致，也区分ASC(默认)和DESC，因为排名总得有个依据

```
SELECT salary, dense_rank() over(ORDER BY salary DESC) AS rnk
FROM employee
```

三、数据更新

1. 插入数据

① 插入元组

```
insert into Student(Sno,Sname,Sgender,Sdept,Sage)
values('123','小红','男','CS',20);
```

若不指出要添加的属性，则需要添加表中的所有属性

② 插入子查询结果

```
# 对每一个系，求学生的平均年龄，并把结果存入表 Dept_age
insert into Dept_age(Sdept,Avg_age)
select Sdept,AVG(Sage)
from Student
group by Sdept;
```

2. 修改数据

① 修改某个元组的值

```
# 修改学号001的年龄
update Student
set Sage = 15
where Sno = '001';
```

② 修改多个元组的值

```
# 将所有学生年龄加1岁
update Stduent
set Sage = Sage + 1;
```

③ 带子查询的修改语句

```
# 将计算机系全体学生成绩置0
update Student
set Sgrade = 0
where Sno in(
    select Sno
    from Student
    where Sdept = 'CS'
);
```

3. 删除数据

① 删除某个元组

```
# 删除学号001学生记录
delete
from Student
where Sno = '001';
```

② 删除多个元组

```
# 删除所有学生记录
delete
from student;
```

③ 带子查询的删除语句

```
# 删除计算机系所有学生的选课记录
delete
from SC
where Sno in(
    select Sno
    from Student
    where Sdept = 'CS'
);
```

```sql
# 删除重复数据，只保留一条记录（除id以外，其余全部相同）
-- ② 删除除了分组中最小id以外的所有值，即重复数据 --
delete from Student
where id not in(
    select id from(
        -- ① 按照除id以外的任意属性就行分组排列，并选出每个分组中的最小id --
        select MIN(id) from Student
        group by Sname
    )temp
);
```

> 注意：
>
> ```sql
> delete from test
> where id not in(
>     select Min(id) from test
>  group by name
> );
> ```
>
> 这样写在 MySQL 中会报错，
>
> `You can't specify target table for update in FROM clause`
>
> **不允许使用同一表中查询的数据作为同一表的更新数据。**
>
> 我们需要在select外面套上一层，让数据库认为我们不是使用同一个表的查询数据作为更新数据

四、空值的处理

处理null：IFNULL(原函数/查询结果,处理后的值)

1. 空值的产生

比如：插入语句中没有赋值的属性，其值为空值

```sql
insert into Student(Sno,Cno)
values('123','323');
# 除了Sno,Cno 外其余属性就是空值
```

2. 空值的判断

IS NULL / IS NOT NULL

```sql
# 查询漏填信息的学生
select *
from Student
where Sname is null or Sgender is null or Sage is null or Sdept is null;
```

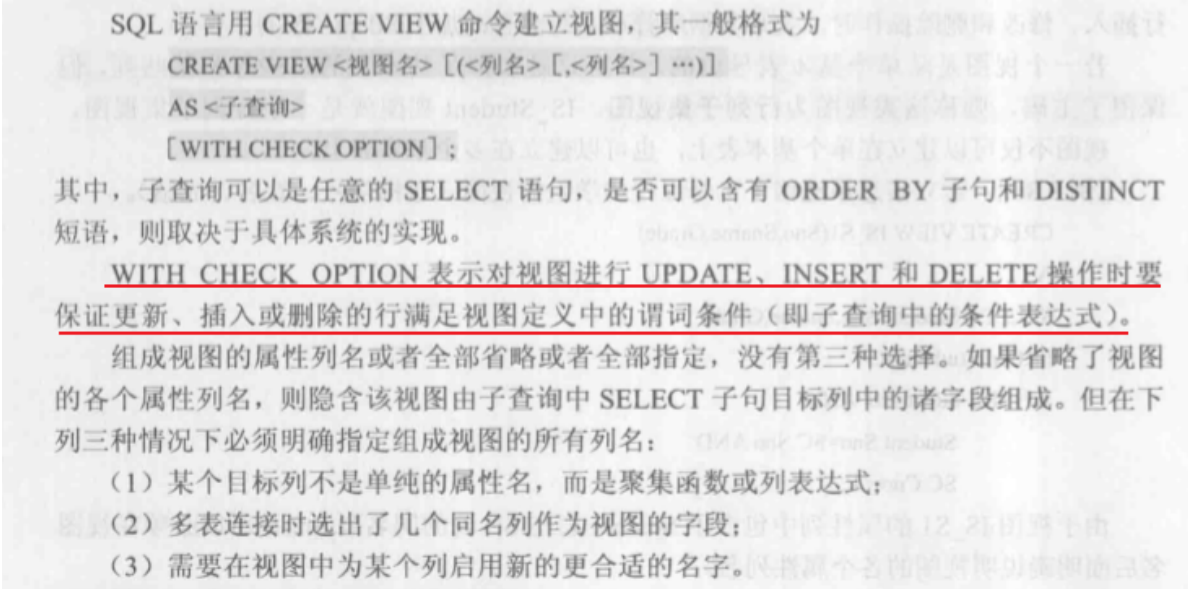
3. 空值的约束条件

- 属性定义中有 **NOT NULL** 约束条件时不能取空值

- 加了 **UNIQUE** 限制的属性不能取空值
- **主键**不能取空值

五、视图

1. 建立视图



SQL 语言用 CREATE VIEW 命令建立视图，其一般格式为

CREATE VIEW <视图名> [(<列名> [,<列名>] …)]
AS <子查询>
[WITH CHECK OPTION];

其中，子查询可以是任意的 SELECT 语句，是否可以含有 ORDER BY 子句和 DISTINCT 短语，则取决于具体系统的实现。

WITH CHECK OPTION 表示对视图进行 UPDATE、INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）。

组成视图的属性列名或者全部省略或者全部指定，没有第三种选择。如果省略了视图的各个属性列名，则隐含该视图由子查询中 SELECT 子句目标列中的诸字段组成。但在下列三种情况下必须明确指定组成视图的所有列名：

（1）某个目标列不是单纯的属性名，而是聚集函数或列表达式；

（2）多表连接时选出了几个同名列作为视图的字段；

（3）需要在视图中为某个列启用新的更合适的名字。

① 建立在单个表上的视图

```
# 建立计算机系学生视图，并要求插入/修改/删除操作时，保证该视图只有计算机系学生
create view CS_Student
as
select *
from Student
where Sdept = 'CS'
with check option;
```

由于加上了 with check option 子句，以后对视图进行 修改 / 添加 / 删除 操作时，DBMS都会自动加上 Sdept = 'CS' 这个条件

若一个视图是从单个基本表导出的，并且只是去掉了某些行某些列，但保留了主键，则称这类视图为 **行列子集视图。** 上述视图 CS_Student 就是一个行列子集视图

② 建立在多个表上的视图

```
# 建立计算机系选修了1号课程的学生的视图（包括学号，姓名，成绩）
create view CS_S1(View_Sno,View_Sname,View_Grade)
as
select Student.Sno,Sname,Grade
from Student,SC
where Student.Sno = SC.Sno and
      Sdept = 'CS' and
      SC.Cno = '1';
```

由于视图的属性列中包含了两个表的同名列 Sno，所以必须在视图名后面说明视图的各个属性列名

③ 建立在视图上的视图

```
# 建立计算机系选修了1号课程 且 成绩在90分以上的学生的视图
create view CS_S2
as
select View_Sno,View_Sname,View_Grade
from CS_S1
where View_Grade >= 90;
```

2. 删除视图

```
drop view 视图名;
```

若该视图上还导出了一个视图，则删除视图操作拒绝执行

或者可以使用 `CASCADE` 进行级联删除，删除该视图和由它导出的所有视图

```
drop view 视图名 CASCADE;
```

3. 查询视图

视图的查询和表的查询是一样的

```
# 建立计算机系学生视图，并要求插入/修改/删除操作时，保证该视图只有计算机系学生
create view CS_Student
as
select *
from Student
where Sdept = 'CS'
with check option;

# 查询选修了1号课程的计算机系学生
select CS_Student.Sno,Sname
from CS_Student,SC
where CS_Student.Sno = SC.Sno and
      SC.Cno = '1';
```

4. 更新视图

更新视图和更新表操作基本一致，不过有些时候视图是不允许更新的

目前，各个关系数据库管理系统一般都只允许对行列子集视图进行更新，而且各个系统对视图的更新还有更进一步的规定。由于各系统实现方法上的差异，这些规定也不尽相同。

例如，DB2 规定：

（1）若视图是由两个以上基本表导出的，则此视图不允许更新。

（2）若视图的字段来自字段表达式或常数，则不允许对此视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。

（3）若视图的字段来自聚集函数，则此视图不允许更新。

（4）若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

（5）若视图定义中含有 DISTINCT 短语，则此视图不允许更新。

（6）若视图定义中有嵌套查询，并且内层查询的 FROM 子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。例如，将 SC 表中成绩在平均成绩之上的元组定义成一个视图 GOOD_SC：

```
CREATE VIEW GOOD_SC
AS
SELECT Sno,Cno,Grade
FROM SC
WHERE Grade >
      (SELECT AVG(Grade)
       FROM SC);
```

导出视图 GOOD_SC 的基本表是 SC，内层查询中涉及的表也是 SC，所以视图 GOOD_SC 是不允许更新的。

（7）一个不允许更新的视图上定义的视图也不允许更新。

应该指出的是，不可更新的视图与不允许更新的视图是两个不同的概念。前者指理论上已证明其是不可更新的视图。后者指实际系统中不支持其更新，但它本身有可能是可更新的视图。

七、函数

function

```sql
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
    SET N := N-1;
    RETURN (
        SELECT salary FROM employee GROUP BY salary ORDER BY  salary DESC LIMIT N, 1
    );
END


create procedure my_insert(in insertCount int)
begin
    declare i int default 1;
    while i <= insertCount do
        insert into big_table(name, age, score, class) values (concat('martin', i), round(50*rand(),0), round(100*rand(),0), round(10*rand(),0));
        set i = i + 1;
    end while;
end;
```

```
18
19 call my_insert(2000000)
```

八、总结

思路1：单表查询

这种解法形式最为简洁直观，但仅适用于查询全局排名问题，如果要求各分组的每个第N名，则该方法不适用；而且也不能处理存在重复值的情况。

思路2：子查询

思路3：自连接

一般来说，能用子查询解决的问题也能用连接解决。

思路4：笛卡尔积

思路5：自定义变量

思路6：窗口函数

至此，可以总结MySQL查询的一般性思路是：

- 能用单表优先用单表，即便是需要用group by、order by、limit等，效率一般也比多表高
- 不能用单表时优先用连接，连接是SQL中非常强大的用法，小表驱动大表+建立合适索引+合理运用连接条件，基本上连接可以解决绝大部分问题。但join级数不宜过多，毕竟是一个接近指数级增长的关联效果
- 能不用子查询、笛卡尔积尽量不用，虽然很多情况下MySQL优化器会将其优化成连接方式的执行过程，但效率仍然难以保证
- 自定义变量在复杂SQL实现中会很有用，例如LeetCode中困难级别的数据库题目很多都需要借助自定义变量实现
- 如果MySQL版本允许，某些带聚合功能的查询需求应用窗口函数是一个最优选择。除了经典的获取3种排名信息，还有聚合函数、向前向后取值、百分位等，具体可参考官方指南。