关系数据库标准语言SQL

1. SQL 的基本概念

   支持 SQL 的关系数据库管理系统同样支持数据库三级模式结构。如 图 1 SQL 对关系数据库模式的支持。其中外模式包括若干视图（view）和部分基本表（base table），数据库模式包括若干基本表，内模式包括若干存储文件（stored file）。

   基本表是本身独立存在的表，在关系数据库管理系统中有个关系就对应一个基本表。一个或多个基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

   存储文件的逻辑结构组成了关系数据库系统的内模式。存储文件的物理结构对最终用户是隐蔽的。

   视图是从一个或几个基本表导出的表。它本身不独立存储在数据库中，即数据库中只存放视图的定义而不存放视图对应的数据。这些数据仍存放在导出视图的基本表中，因此视图是一个虚表。视图的概念与基本表等同，用户可以在视图上再定义视图。

2. 学生-课程数据库

   该数据库包含三个表：

   > 学生表：Student(Sno,Sname,Ssex,Sage,Sdept)
   > 课程表：Course(Cno,Cname,Cpno,Ccredit)
   > 学生选课表：SC(Sno,Cno,Grade)

   关系中主码加下划线表示。各表数据如下：

   学生表

学号(Sno)	姓名(Sname)	性别(Ssex)	年龄(Sage)	系(Sdept)
140406001	张三	男	18	计算机
140406002	李四	男	19	计算机
140406003	王二	男	17	计算机
140405001	韩六	男	18	网络
140405002	赵红花	女	19	网络

   课程表

课程号(Cno)	课程名(Cname)	先行课(Cpno)	学分(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	Java	6	4

   学生选课表

学号(Sno)	课程号(Cno)	成绩(Grade)
140406001	1	84
140406001	2	92
140406001	3	88
140405002	2	99
140405002	3	98

3. 数据定义
   i. 模式（数据库）的定义与删除
      - 定义模式

        语法

        ```
        CREATE SCHEMA <模式名> AUTHORIZATION <用户名>;
        ```

        注：若没有指定<模式名>,那么<模式名>隐含为<用户名>。

        要创建模式，调用该命令的用户必须拥有数据库管理员权限，或者获得了数据库管理员授予的 `CREATE SCHEMA` 的权限。

        例如在 SQL Server 中创建一个模式：

        ```
        create database <数据库名>;
        ```

      - 删除模式

        语法：

        ```
        DROP SCHEMA <模式名>;
        ```

        其中 CASCADE 和 RESTRICT 必选一个。选择前一个表示级联，在删除模式的同时把该模式中所有的数据库对象全部删除；选择后一个表示限制，若该模式已经定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。

        例如在 SQL Server 中：

        ```
        drop database <数据库名>;
        ```

   ii. 基本表的定义、删除与修改
      - 定义基本表

        创建了一个模式就建立了一个数据库的命名空间，一个框架。在这个空间首先要定义的是该模式包含的数据库基本表。
        语法：

```
CREATE TABLE <表名>(
  <列名> <数据类型> [列级完整性约束],
  <表级完整性约束>
  );
```

建表的同时通常还可以定义与该表有关的完整性约束条件，这些完整性约束条件被存入系统的数据字典中，当用户操作表中数据时由关系数据库管理系统自动检查该操作是否违背这些完整性约束条件。若完整性约束条件涉及该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。
例如在 SQL Server 中：

```
create table Course(
  Cno char(4) primary key,//列级完整性约束，设置为主码
  Cname char(40) not null,//列级完整性约束条件，设置不能取空值
  Cpno char(4) unique,//列级约束条件，取唯一值
  Ccredit smallint,
  foreign key (Cpno) references Course(Cno)//表级完整性约束条件，Cpno 是外码，被参照表是 Course,
  );
```

- 模式与表

  每一个基本表都属于某一个模式，一个模式包含多个基本表。当定义基本表时一般可以有三种方法定义它所属的模式：

  - 在表名中明显地给出模式名：

    ```
    create table "数据库名".表名();
    ```

  - 在创建模式语句中同时创建表;
  - 设置所属的模式，这样在创建表时表名中不必给出模式名。

  当用户创建表（其他数据库对象也一样）时若没有指定模式，系统根据搜索路径（serch path）来确定该对象所属的模式。
  使用下面的语句可以显示当前的搜索路径：

  ```
  show serch_path;
  ```

  管理员也可以设置搜索路径：

  ```
  set serch_path to "模式名",public;
  ```

- 修改基本表

  语法：

  ```
  alter tale <表名>
  [add [column] <新列名> <数据类型> [完整性约束]]
  [add <表级完整性约束>]
  [drop [column] <列名> [CASCADE|RESTRICT]]
  [drop constraint <完整性约束名> [RESTRICT|CASCADE]]
  [alter column <列名> <数据类型>];
  ```

add 子句用于增加新列、新的列级完整性约束条件和新的表级完整性约束条件和新的表级完整性约束条件。drop column 子句用于删除表中的列;drop constraint 子句用于删除指定的完整性约束条件；alter column 子句用于修改原有的列定义，包括列名和数据类型。

- 删除基本表

  语法：

  ```
  drop table <表名> [RESTRICT|CASCADE]
  ```

- 数据类型

  关系模型中一个很重要的概念是域。每一个属性来自一个域，它的取值必须是域中的值。在 SQL 中域的概念用数据类型来实现，定义表的各个属性时需要指明其数据类型及长度。

数据类型	含义
char(n),character(n)	长度为 n 的字符串
varchar(n),charactervarying(n)	最大长度为 n 的变长
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,将输入日期时需要加单引号
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

iii. 索引的建立与删除

当表的数据量比较大时，查询操作会比较耗时。建立索引是加快查询速度的有效手段。数据库索引类类似于图书后面的索引，能加快定位到需要查询的内容。用户可以根据应用环境的需要建立一个或多个索引，以提供多种存储路径，加快

查找速度。
数据库索引由多种类型，常见索引包括顺序文件上的索引、B+ 数索引、散列索引、位图索引等。顺序文件上的索引是针对按指定属性值升序或降序存储的关系，在该属性上建立一个顺序索引文件，索引文件由属性值和相应的元组指针组成。B+ 索引树索引是将索引属性组织成 B+ 树形式，B+ 树的叶结点为属性值和相应的元组指针。B+ 树索引具有动态平衡的优点。散列索引是建立若干个桶，将索引属性按照其散列函数映射到相应桶中，桶中存放索引属性值和相应的元组指针。散列索引具有查找速度快的特点。位图索引是用位向量记录索引属性中可能出现的值，每个向量对应一个可能值。
索引虽然能够加速数据库查询，但需要占用一定的存储空间，当基本表更新时，索引要进行相应的维护，这些都会增加数据库的负担，因此要根据实际应用的需要有选择地创建索引。
一般来说，建立与删除索引由数据库管理员或表的属主，即建立表的人，负责完成。关系数据库管理系统在执行查询时会自动选择合适的索引作为存取路径，用户不必也不能显示地选择索引。索引是关系数据库管理系统的内部实现技术，属于内模式范围。

- 建立索引
  在 SQL 语言中，建立索引使用 create index 语句，其一般格式为

```
create [UNIQUE] [CLUSTER] index <索引名> on <表名>(<列名>[<次序>] [, <列名> <次序>]……);
```

  其中，<表名> 是要建索引的基本表的名字。索引可以建立在该表的一列或多列上，各列名之间用逗号分隔。每个<列名>后面还可以用<次序>指定索引值的排列次序，可选 ASC（升序） 或 DESC（降序），默认值为 ASC。
  UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录。
  CLUSTER 表示要建立的索引是聚族索引。
- 修改索引
  语法：

```
alter index <> rename to <新索引名>;
```

- 删除索引
  语法：

```
drop index <索引名>;
```

4. 数据查询

数据查询时数据库的核心操作。SQL 提供 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。其一般语法为：

```
select [ALL|DISTINCT] <目标列表达式>[,<目标表达式>]……
from <表名或视图名>[,<表名或视图名>]|(
```

- 单表查询
  - 选择表中若干列
    - 查询指定列
      语法：

```
select <目标表达式> from <表名>;//目标列表达式指定要查询的属性列,目标列表达式中各个列的先后顺
例如查询全体学生的学号和姓名：
select Sno,Sname from Student;
```

- 查询全部列（整张表）
  语法：

```
select * from <表名>;
例如查询全体学生的详细记录：
select * from Student;
```

  - 查询经过计算的值
    语法：

```
select <目标列表表达式> from <表名>;//目标列表表达式可以为一个表达式,字符串常量、函数等。
例如查询学生的出生年：
select Sname,2017-Sage from Student;
查询所在系并用小写字母表示：
select lower(Sdept) from Student;
使用别名表示小写后的列：
select lower(Sdept) dept from Student;
```

- 选择表中的若干元组
  - 消除取值重复的行
    在查询的列前面使用关键字 distinct 消除他们：

```
例如查询选了课程并有成绩的人：
select distinct Sno from SC;
```

  - 查询满足条件的元组
    用于查询满足条件的元组可以通过 w here 子句实现。
    - 比较大小
      用于进行比较大小的运算符一般包括 `=(等于)、>(大于)、<(小于)、>=(大于等于)、<=(小于等于)、!=或<>(不等于)、!>(不大于)、!<(不小于)` 以及="" `not+上述运算比较符`。="" 例如查询计算机系全体学生：select Sname from Student w here Sdept='CS'; 例如查询年龄小于 20 的学生： select Sname from Student w here Sage<20; <="" pre="">
    - 确定范围
      使用 `betw een……and……` 和 `not betw een……and……` 可以用来查找属性值在（或不在）指定范围内的元组，其中 betw een 后跟下限，and 后跟上限。

```
查询年龄在 18~20 岁的学生名：
select Sname from Student where age between 18 and 20;
查询年龄不在 18~20 岁的学生名：
select Sname from Student where age not between 18 and 20;
```

    - 确定集合
      使用 `in` 可以用来查询属性值属于指定集合的元组。`not in` 查询属性值不属于指定集合的元组。

```
查询计算机（CS)、数学（MA）、和信息系（IS）的学生姓名：
select Sname from Sdept in ('CS','MA','IS');
查询不是计算机（CS）、数学（MA）的学生姓名：
select Sname from Sdept not in ('CS','MA');
```

    - 字符匹配
      谓语 like 可以用来进行字符串匹配。一般语法如：`[not] like '<匹配串>' [escape '<换码字符>']`。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，也可以是含有通配符 % 和 _ 。 其中 `%` 代表任意长度（长度可为 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；`_` 表示代表任意单个字符,例如 a_b 表示以 a 开头 b 结尾的长度为 3 的任意字符串，注意当字符集为 GBK 时汉字只需一个 _ ,而为 ASCII 时一个汉字需要两个 _ 。若想要查询的字符串本身就含有通配符 % 或 _ ,这时需要使用 `escape '<换码字符>'`。

```
查询所有姓李的同学：
select Sname from Student where Sname like '李%';
```

- 涉及空值的查询
  使用 `is null` 查询空值记录。使用 `is not null` 查询不为空值的记录。

```
查询选修课程而没有成绩的学号：
select Sno from SC where Grade is null;//不可以使用 Grade=null;
查询选修课程而有成绩的学号：
select Sno from SC where Grade is not null;
```

- 多重条件查询
  逻辑运算符 AND 和 OR 可用来连接多个查询条件，AND 优先级高于 OR，可以使用括号改变优先级。

```
查询计算机系且年龄小于 19 的同学学号：
select Sno from Student where Sdept='CS' and Sage<19; <="" pre="">
```

- ORDER BY 子句
  用户可以使用 ORDER BY 子句对查询的结果按照一个属性列升序（ASC）或者降序（DESC）排列，默认值为升序。

```
查询选修 3 号课程的学生的学号及成绩，结果按成绩降序排列。
select Sno,Grade from SC where Cno=3 order by Grade desc;
```

- 聚集函数
  SQL 提供许多聚集函数，如下：
  - count(*) 统计元组个数
  - count([DISTINCT|ALL] <列名>) 统计一列中值的个数
  - sum([DISTINCT|ALL] <列名>) 计算一列值的总和（此列必须为数值型）
  - avg([DISTINCT|ALL] <列名>) 计算一列值的平均值（此列必须为数值型）
  - max([DISTINCT|ALL] <列名>) 求一列中最大值
  - min([DISTINCT|ALL] <列名>) 求一列中最小值

  注意： where 子句中是不能用聚集函数作为条件表达式的。聚集函数只能用于 select 子句和 group by 中 having 子句。

```
查询学号为 201215121 选修课程的总学分数。
select sum(Ccredit) from SC,Course where Sno=201215121 and SC.Cno=Course.Cno;
```

- GROUP BY 子句
  GROUP BY 子句将查询结果按某一列或多列的值分组，值相等的为一组。对查询结果分组的目的是为了细化聚集函数的作用对象。如果未对查询结果进行分组，聚集函数将作用于整个查询结果。分组后聚集函数将作用于每一个组，即每一组都有一个函数值。

```
查询选修了三门以上课程的学生学号：
select Sno from SC group by Sno havin count(\*)>3;//这里先对 Sno 进行分组，再用聚集函数对每一组计
```

- 连接查询
  若一个表查询同时涉及两个以上的表，则称为连接查询。
  - 等值与非等值查询
    查询的 where 子句中用来连接两个表的条件称为连接条件或连接谓语，其一般格式为：`[<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>` 或 `[<表名1>.]<列名1> between [<表名2>.]<列名2> and [<表名2>.]<列名2>`。当运算符为 = 时，称为等值连接。使用其他运算符称为非等值连接。**一条 SQL 语句可以同时完成选择和连接查询，这时 WHERE 子句是由连接谓语和选择谓语组成的复合条件。**

```
例如查询每个学生及其选修课程的情况：
select Student.*,SC.* from Student,SC where Student.Sno=SC.Sno;
```

  - 自身连接
    连接操作不仅可以在两个表之间进行，也可以是一个表与其自己进行连接，称为表的自身连接。要进行表的自身连接，就要对表取别名。为 Course 取两个别名为 FIRST、SECOND。

```
例如查询没一门课的先修课：
select FIRST.Cno,SECOND.Cpno from Course FIRST,Course SECOND where FIRST.Cpo=SECOND.Cno;
```

  - 外连接
    有时想把 Student 表与 SC 表连接起来查看学生的选课成绩情况，可是部分学生没有选课，所以在连接时导致 Student 中这些元组在连接时被舍弃了。但我们仍想把 Student 的悬浮元组保存在结果关系中，而在 SC 表的属性上填空值 NULL，这时就需要外连接。
    例如：

```
select Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade from Student left join SC on (Student.Sno=
```

  - 多表连接
    在 where 子句中用 and 连接。例如：

```
select Student.Sno,Sname,Cname,Grade from Student,SC,Course where Student.Sno=SC.Sno and SC.Cnc
```

- 嵌套查询

在 SQL 语言里，一个 `select-from-where` 语句称为一个查询块。将一个查询块嵌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。例如：

```
SELECT Sname From Student WHERE Sno IN(SELECT Sno FROM SC WHERE Cno='2');
```

SQL 语言允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。需要特别指出的是，子查询的 `SELECT` 语句中不能使用 `ORDER BY` 子句，`ORDER BY` 子句只能对最终查询结果排序。

  - 带 IN 谓语的子查询

    在嵌套查询中，子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最常用的谓语。
    例如：查询选修了课程名为 “信息系统” 的学生学号和姓名。
    分析：该查询涉及学号、姓名和课程名三个属性。学号和姓名存放在 Student 表中，课程名存放在 Course 表中，但 Student 与 Course 两个表之间没有直接联系，必须通过 SC 表建立它们二者之间的关系，所以本表实际上涉及三个关系。

```
SELECT Sno,Sname FROM WHERE Sno IN(SELECT Sno FROM SC WHERE Cno IN(SELCET Cno FROM Course WHERE
//分析
1. SELCET Cno FROM Course WHERE Cname='信息系统' 找出‘信息系统’的课程号，结果为3.
```

```
2. SELECT Sno FROM SC WHERE Cno IN(SELCET Cno FROM Course WHERE Cname='信息系统') 在 SC 关系中找
3. 最后在 Student 关系中取出 Sno 和 Sname。
```

同样可以使用连接查询实现：

```
SELECT Student.Sno,Snmae FROM Student,SC,Course WHERE Student.Sno=SC.Sno AND SC.Cno=Course.Cno
```

- 带有比较运算符的子查询
  当用户能确切知道内层查询返回的是单个值时，可以用 `>、<、=、>=、<=、!=或 <="">` 等比较运算符。
  例如：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=s.Sno);
```

- 带有 ANY（SOME）或 ALL 谓语的子查询
  子查询返回单值时可以使用比较运算符，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 谓词时则必须同时使用比较运算符。其语义如下：
  - >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

  等于查询结果中的所有值

- !=(<>)ANY

  不等于子查询中的某个值

- !=(<>)ALL

  不等于子查询中的任何一个值

例如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';
```

-