数据库笔记（二）

SQL语言

操作的对象和结果都是集合

C/S架构，client和DB server端口通信

SQL语句可以作为在终端输入的命令执行，也可以作为嵌入程序的脚本执行

数据类型

- 预定义类型：数字、字符串、二进制串、布尔、日期date、时间间隔interval、XML类型
- 原子构造类型：引用
- 组合构造类型：集合、字段、行（元组）
- 空值

DDL（openGauss）

如果不特殊设置的话，创建者作为数据库的owner，被授权的user可以访问数据库

先创建用户，创建Database，然后在库内创建数据表Table，View和Index可选

Create（创建），Alter（更改），Drop（销毁）

schema：逻辑划分数据库，用户可以直接访问默认的访问空间，访问其他用户的空间要带上模式名，Select Bob.s

连接上数据库 gsql -d postgres -p 26000 -r

DQL

SELECT... FROM... WHERE...（可以没有where）

首先执行from，然后where（查询条件），最后select（选出字段）

变量表达式要先定义后使用，所以执行顺序就很重要，在select语句中定义的就不可以在其他语句中使用，select和from中可以定义变量，where中不可以定义变量

选择对应where，投影对应select，连接（笛卡尔积）对应from

SELECT * FROM Doctor; (！！SQL语句用;结束)

SELECT可以设置列别名，SELECT 原名 别名，查询的结果是一个新的关系，不会改变原来的关系

DISTINCT 合并查询中的重复元组（SQL的数学基础不是集合，是bag运算，多重集合，查询的结果允许重复）

SELECT INTO 将查询结果创建为新的表

SELECT * INTO R2 FROM R1 WHERE 0=1　复制一张表头（属性值和域）和旧表完全一样的新表，内容为空

SELECT TOP n

条件查询

where后面的行选择表达式的形式：

NULL 只能用is (not)，不能用=或<>

BETWEEN...AND... （包含端点值）

属性列 **（NOT）IN** 集合

LIKE 模糊查询 %代表任意长度的字符串，_代表任意单个字符，'bio%'，'%bio%'（bio出现在任意位置），'%bio'，[]代表范围[a-f]，[^]不再范围内，当用户要查询的字符串本身就含有 % 或 _ 时，要使用 ESCAPE '<换码字符>' 短语对通配符进行转义。

？汉字和英语字符的编码长度不一样，需要用几个下划线

带算术表达式的查询，SELECT和WHERE后面可以对某个字段做算术运算，FROM后面是关系，不能用算术运算，只能用集合运算

EXISTS 和 **NOT EXISTS**只关心子查询结果是否为空，不关心具体结果内容，返回true或false，因此子查询SELECT子句后常用 *

全称量词，关系代数中用除法，SQL中用 NOT EXISTS，eg：申请了所有学校，不存在一所学校他没有申请

谓词，**ANY/ALL** 至少有一个值满足/所有值都满足，一般用来修饰集合（返回集合的子查询）有点类似于存在和任意，> ANY等价于> MIN()

统计查询

没有循环，分支语句，统计用**聚集函数**来完成，SELECT语句后面加聚合函数，一次聚合函数查询返回**一条记录**，select sID，sName，MAX(GPA)，如果只有一个最高的GPA就可以查询，有多个就会报错，注意select后面给出的字段的行数一定要是相同的

求平均值、总和都不会算入NULL，平均值=总和÷计数，计数也不会加入NULL

COUNT(EXP) 统计EXP字段非NULL的行数，COUNT(*) 统计表的总行数

聚集函数中用ALL表示对所有非空值计算，DISTINCT表示重复值只算一次，AVG(DISTINCT column_name)

注意语义，语法可能正确但语义不符，注意需不需要去重

聚集函数不能在where语句后面直接使用

GROUP BY （对某个字段做聚集）... **HAVING**（对每个分组做统计筛选，不能用where只能having）

eg1: select rid,count(rid) from rating group by rid;

eg2: select reviewer.rid from reviewer,rating where reviewer.rid=rating.rid group by reviewer.rid having count(stars)>=3;

先from,where，再group by，再做select

group by必须与聚集函数一起使用，不能返回一个分组之后的表，只能对每个分组返回用聚集函数计算得到的结果

做了group by之后select除了聚集函数还可以选group by后面的字段，选其他的很可能行数不一样导致报错

？ HAVING在你的数据库中是否支持单独使用，认为每一行都是一个分组。openGauss不支持

ORDER BY 排序查询，必须放在查询语句的最后部分（实际也是最后执行的），且一条查询语句只能有一个

ASC，DESC表示升序和降序，系统默认升序，可以有多个关键字，多重排序，从前往后用，默认空值NULL最小

连接查询

连接的时候条件中一般都会加上表名作为前缀来避免混淆

from R1,R2就是R1和R2做笛卡尔积

条件连接

```
◆FROM R1,R2
  WHERE R1.A1 比较运算符 R2.A2

◆FROM R1 INNER JOIN R2 ON ( R1.A1 比较运算符 R2.A2 )

◆FROM R1 INNER JOIN R2 USING( A1 )
```

JOIN 默认是 INNER JOIN

USING 等值同名

自然连接

在所有同名属性（组合）上做等值连接，删除属性值重复的列

R1 NATURAL JOIN R2

自连接

在FROM语句中为关系取别名

自连接的时候还要注意自己连自己和一前一后的情况

外连接

当左右两张表的连接属性相等，其他属性都存在时，外连接和内连接的结果相同

R1 LEFT OUTER JOIN R2

？ 不支持外连接

嵌套查询

不相关：子查询能够独立运行，不依赖外部查询

不相关的子查询大多可以用连接查询代替，但不一定，集合的交集和差集可以用in，not in+子查询代替，自然连接一定可以用相同属性是否在选出的集合中来代替

如果连接和子查询等效，一般使用连接，连接需要做笛卡尔积，运算效率较低，但是join只需要一次I/O操作就可以缓存下两张表，子查询需要两次，而且建立索引后不需要做全部的笛卡尔积，所以实际操作中连接的效率更高

eg：找出申请者总数超过1000的高校，计算其申请者GPA差距的最大值。

```
SELECT MAX(MX-MN)
FROM
(SELECT MAX(GPA) MX,MIN(GPA) MN
FROM Apply
GROUP BY cID
HVING COUNT(DISTINCT sID) >=1000) ;
```

相关子查询

？**相关**子查询只能在比较运算符的**右边**，与=,<>,>,<相连时必须返回非空的单值集合

使用子查询或者定义变量可以减少计算量，多次出现时只算一次，但要注意执行的顺序

子查询可以出现在SELECT，FROM，WHERE中，但是大多是WHERE中

eg1：各高校的报考者中，GPA超过该学校平均分的学生

！！where后面的条件是对每一行操作的，所以where后面的嵌套查询就相当于有了两层循环，外层是遍历A表的行，所以传给每一个子查询的是A中固定的一行，在子查询中遍历B表的行，找出B的行中cID与A的cID相等的，然后算AVG

```
SELECT sID,sName, GPA
FROM Student NATURAL JOIN Apply A
WHERE AND GPA>
    (SELECT AVG(GPA)
     FROM Student NATURAL JOIN Apply B
     WHERE A.cID=B.cID) ;
```

所以用in和not in的不相关子查询往往能和用exist和not exist的相关子查询等价

用in把所有符合条件的筛出一个集合，一行行判断属性是否在这个集合内

用exist把某一行传进子查询，做筛选，返回存在或不存在

eg2：申请了计算机专业但没有申请通信专业的学生学号和姓名

```
select sID,sName from Student where sID in (select sID from Apply where
major='计算机') and sID not in (select sID from Apply where major='通信') ;

select sID,sName from Student S where exists (select * from Apply A where
S.sID=A.sID and major='计算机') and not exists (select * from Apply B where
S.sID=B.sID and major='通信');
```

eg3：表示关系代数中的除法，采用双嵌套 not exist，R(X，Y)÷S(Y，Z)

象集：每个X对应的所有Y

遍历每个X，对固定的X遍历每个Y，去看Y是否属于这个X对应的象集（求象集就是最后一次遍历，select * from R R2 where）

难就难在判断X的象集是否包含所有的Y，子查询只能放在右边，所以不能判断子查询是否包含别人，只能遍历所有的Y，一个个判断是否属于象集

所以要在外层查询里from Y，在子查询里用S.Y传到

```
select distinct R.X from R R1
where not exists
(
    select S.Y from S
    where not exists
    (
        select * from R R2
        where R2.X=R1.X and R2.Y=S.Y
    )
)
```

集合查询

union, intersect, except

属性名不一定要相同，属性的数量和域相同即可，SQL会用第一个关系的属性名

select的结果是可以有重复记录的，但是做了union之后会去除重复记录，因为union是集合运算，如果指定union all就会保留重复记录

很多数据库不支持交集和差集，就使用in,not in和嵌套查询

！！易错点

1. sql的基础是包运算，允许重复，注意想想要不要去重，比如查询申请了计算机专业的学生人数，学生可能申请了几所不同学校的计算机专业，导致sID出现多次，所以要COUNT(distinct sID)

2. 多张表有同名的属性时注意要表名.来区分
3. 使用聚合函数注意要不要distinct，NULL

DML

插入

字段列表和插入值必须对应

如果每个字段都要插入，且插入顺序与字段顺序一致时可以省略字段列表

value 一次插入一行

select 子句将查询的结果全部插入

先写出字段，需要查询的先写成select，后面再写from ... where

更新

update ... set ... [where（只更新满足条件的]

删除

只能一行行删除，要删除一个值就是更新为NULL

delete from ... [where]

delete from 没有where和drop不同，delete只是把所有记录删掉，表的模式和数据结构（表头）还在，drop是把表都删了

DCL

授权（grant）：grant create session to scot

回滚（rollback）

提交（commit）

新建用户（create user）

视图

创建视图：create view <视图名> [列名] as <子查询>

可以不指明列名，如果select中使用了聚集函数、算术表达式，或者有列名冲突就要指明

视图是一个虚表，可以看作是一个关系模式，一个表头

定义（创建）视图的时候不执行select语句，在做视图查询的时候才会执行，所以不会产生数据冗余

定义视图的时候不使用order by和distinct，order by没有意义，创建的虚表没有内容

查询视图：select ... from 视图

修改视图：对视图的修改实际上是对基表的修改，所以要符合基表的约束，限制条件很多，视图只能来自单张基表，不含聚集函数，group by，distinct，计算表达式，因此意义不大

删除视图：drop view <视图名>

可以在视图上定义新的视图，但不能在视图上定义新的基本表

好处：简化用户的操作，提高数据安全性，逻辑独立性

索引

建立索引是为了提高查找的效率，属于内模式范畴

索引可以采用B树，B+树，哈希表等结构

大多DBMS都会在主键上自动建立索引，也可以由DBA和owner手动完成，使用和维护索引由DBMS自动完成

- 聚簇索引

表中记录的物理存储位置和索引的顺序一致，索引是某个属性的升序或降序，查询时就可以做二分查找

一个基本表最多建立一个

一般只做查询，很少做修改，因为重新排序的开销极大

Create Cluster Index <索引名> on <表名>[列名]

- 非聚簇索引

用额外的空间来顺序存储索引，然后每个索引项用指针指向实际存储的数据，数据不需要顺序存储

查找效率比索引低，因为要用指针去找，不是直接的

可以建立多个

聚簇的辅助键索引和主键关联，先找到对应的主键值，然后利用主键索引进行查找

实际建索引的时候，DBMS已经建立了默认索引，用户能建立的索引类型已经确定了，不能指定是cluster

- 建立索引的原则：

适量建立，索引要占据空间，花费维护时间

数据量大的时候建立

建索引的列不要有太多重复值，优先在主键上建立索引，对select，where中用的多的字段建立索引

在数据初始化完了之后建立索引

等值查询多用哈希表建索引更快，非等值的只能用B+树

完整性约束

约束一般在定义数据库表后被创建（写在create语句里面），当更新数据库的时候就会检查更新是否符合约束，不符合就会被拒绝

主键约束：唯一，最小，主键的属性值不允许空值

唯一约束 Unique， 主键只能有一个，Unique约束可以建立多个，？是否允许null

非空约束 NOT NULL

用户自定义约束：Check

约束条件放在列定义的后面或者所有列定义完之后

create table <表名> {

列名 数据类型 Primary Key/Unique/Not null/CHECK(Dsex IN (‘男’, ‘女’))

}

create table <表名> {

列名 数据类型

Primary Key（某列名）

}

行级约束，列级约束，表级约束？？

参照完整性约束：

先定义被参照表，再定义参照表