openGauss

轻松上手openGauss第七期

——openGauss SQL语言（下）

openGauss

- openGauss资料团队
- 贾军锋

- 2020.12.28

openGauss

目 录
CONTENTS

openGauss

PART 1

where 限定查询

openGauss

where子句

WHERE限定查询中，WHERE子句构成了一个行选择表达式，用来缩小SELECT查询的范围。

- **SQL限定查询基本语法**

```
SELECT [DISTINCT] {*|column [alias],...}
FROM    table
[WHERE  contition(s)];
-- condition是返回值为布尔型的任意表达式，任何不满足该条件的行都不会被检索。
```

- **基本使用示例：**

```
mydb=# select * from emp where deptno=10;
 empno | ename  |   job    | mgr  |      hiredate       |   sal   | comm | deptno
-------+--------+----------+------+---------------------+---------+------+--------
  7782 | 花羊羊 | 部门经理 | 7839 | 2020-06-08 00:00:00 | 2450.00 |      |     10
  7839 | 快羊羊 | CEO      |      | 2020-11-09 00:00:00 | 5000.00 |      |     10
  7934 | Tom    | CFO      | 7782 | 2011-09-11 00:00:00 | 1300.00 |      |     10
```

openGauss

where子句示例一(比较限定操作符)

- **基本数值比较限定语句**

```
postgres=# select * from emp where deptno =  10;
postgres=# select * from emp where deptno != 10;
postgres=# select * from emp where deptno <= 10;
postgres=# select * from emp where sal > nvl(comm,0);   -- 对空值的列进行比较要对空值进行处理
```

- **使用伪列rownum限定输出条目**

```
mydb=# select * from emp where rownum < 3;
 empno | ename  |  job   | mgr  |      hiredate       |   sal   |  comm  | deptno
-------+--------+--------+------+---------------------+---------+--------+--------
  7369 | 喜洋洋 | 工程师 | 7902 | 2018-12-17 00:00:00 |  800.00 |        |     20
  7499 | 美羊羊 | 销售   | 7698 | 2020-02-20 00:00:00 | 1600.00 | 300.00 |     30

mydb=# select * from emp where rownum > 3;
 empno | ename | job | mgr | hiredate | sal | comm | deptno
-------+-------+-----+-----+----------+-----+------+--------
```

- **使用between and对数值进行查询限定**

```
mydb=# select * from emp where sal between 800 and 1000;
 empno | ename  |  job   | mgr  |      hiredate       |  sal   | comm | deptno
-------+--------+--------+------+---------------------+--------+------+--------
  7369 | 喜洋洋 | 工程师 | 7902 | 2018-12-17 00:00:00 | 800.00 |      |     20
  7900 | 小灰灰 | 工程师 | 7698 | 2016-05-21 00:00:00 | 950.00 |      |     30

mydb=# select * from emp where sal between 1000 and 800;
 empno | ename | job | mgr | hiredate | sal | comm | deptno
-------+-------+-----+-----+----------+-----+------+--------
```

SQL语句未报错
但无结果输出

openGauss

where子句示例二(比较限定操作符)

- 使用between and 对字符串进行限定

```
mydb=# select * from dept1;
 deptno | dname |    loc
--------+-------+-----------
     10 | 财务  | 北京
     20 | 研发  | 上海
     30 | 销售  | 武汉
     40 | 生产  | 西安
    101 | A     | Oakland
    102 | Baaa  | Berkeley
    103 | c     | NewOrleans
    104 | C     | Denver
    105 | Caaa  | Denver
```

字符限定

```
mydb=# select * from dept1 where dname between 'A' and 'C';
 deptno | dname |    loc
--------+-------+-----------
    101 | A     | Oakland
    102 | Baaa  | Berkeley
    103 | c     | NewOrleans
    104 | C     | Denver
```

```
mydb=# select * from dept1 where loc between '北' and '西';
 deptno | dname | loc
--------+-------+------
     10 | 财务  | 北京
     20 | 研发  | 上海
     30 | 销售  | 武汉
```

- 使用between and对时间进行限定查询

```
mydb=# select * from emp
mydb-# where hiredate between '2020-06-06' and sysdate;
 empno | ename  |    job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7654 | 慢羊羊 | 销售      | 7698 | 2020-09-08 00:00:00 | 1250.00 | 1400.00 |     30
  7782 | 花羊羊 | 部门经理  | 7839 | 2020-06-08 00:00:00 | 2450.00 |         |     10
  7839 | 快羊羊 | CEO       |      | 2020-11-09 00:00:00 | 5000.00 |         |     10
  7844 | 灰太狼 | 销售      | 7698 | 2020-09-11 00:00:00 | 1500.00 |    0.00 |     30
  7521 | 懒羊羊 | 销售      | 7698 | 2020-12-22 14:41:17 | 1250.00 |  500.00 |     30
```

openGauss

where子句示例三(比较限定操作符)

- **列举(in)查询限定**

```
mydb=# select * from dept where deptno not in (10,30);
 deptno | dname | loc
--------+-------+------
     20 | 研发  | 上海
     40 | 生产  | 西安

mydb=# select * from dept where loc in ('西安','武汉');
 deptno | dname | loc
--------+-------+------
     30 | 销售  | 武汉
     40 | 生产  | 西安
```

- **like模糊匹配查询**：用LIKE进行某个字符串值的通配符匹配,选出特定行

```
mydb=# select * from emp1 where ename like '%J%';
 empno | ename | job  | mgr  |      hiredate       |   sal   |  comm  | deptno
-------+-------+------+------+---------------------+---------+--------+--------
  7902 | Jerry | CTO  | 7566 | 2017-06-08 00:00:00 | 3000.00 |        |     20
  7521 | J_rr% | 销售 | 7698 | 2020-12-22 14:41:17 | 1250.00 | 500.00 |     30

mydb=# select * from emp1 where ename like '%j%';
 empno | ename | job | mgr | hiredate | sal | comm | deptno
-------+-------+-----+-----+----------+-----+------+--------

mydb=# select * from emp1 where ename like '_e__y';
 empno | ename | job | mgr  |      hiredate       |   sal   | comm | deptno
-------+-------+-----+------+---------------------+---------+------+--------
  7902 | Jerry | CTO | 7566 | 2017-06-08 00:00:00 | 3000.00 |      |     20

mydb=# select * from emp1 where ename like 'J\_rr\%';
 empno | ename | job  | mgr  |      hiredate       |   sal   |  comm  | deptno
-------+-------+------+------+---------------------+---------+--------+--------
  7521 | J_rr% | 销售 | 7698 | 2020-12-22 14:41:17 | 1250.00 | 500.00 |     30
```

openGauss

where子句示例四(比较限定操作符)

- 转义字符【\】

```
mydb=# select * from dept1;
 deptno |  dname   | loc
--------+----------+------
      1 | Hell\_o% | 长安
      2 | Hell_oA  | 京城
      3 | Hell_OA  |
      4 | Hell_o   |

mydb=# select * from dept1 where dname like 'Hell\_o%';
 deptno |  dname  | loc
--------+---------+------
      2 | Hell_oA | 京城
      4 | Hell_o  |

mydb=# select * from dept1 where dname like 'Hell\\\_o\%';
 deptno |  dname   | loc
--------+----------+------
      1 | Hell\_o% | 长安
```

- 空值限定【is null/is not null】

```
mydb=# select * from dept1 where loc is null;
 deptno |  dname  | loc
--------+---------+-----
      3 | Hell_OA |
      4 | Hell_o  |

mydb=# select * from dept1 where loc is not null;
 deptno |  dname   | loc
--------+----------+------
      1 | Hell\_o% | 长安
      2 | Hell_oA  | 京城
```

```
mydb=# select * from dept1 where loc isnull;
 deptno |  dname  | loc
--------+---------+-----
      3 | Hell_OA |
      4 | Hell_o  |

mydb=# select * from dept1 where loc notnull;
 deptno |  dname   | loc
--------+----------+------
      1 | Hell\_o% | 长安
      2 | Hell_oA  | 京城
```

不建议的写法

openGauss

where子句示例五(逻辑限定操作符)

常用的逻辑操作符有AND、OR和NOT 三类，他们的运算结果也有三个值，分别为TRUE、FALSE和NULL，其中NULL代表未知。（已在上一期给大家介绍）

【AND】示例

```
mydb=# select * from emp1 where sal>=1100 and job='工程师';
 empno | ename  |   sal   |  job   | mgr
-------+--------+---------+--------+------
  7876 | 红太狼 | 1100.00 | 工程师 | 7788

mydb=# select * from emp1 where sal>=800 and sal<=1000;
 empno | ename  |  sal   |  job   | mgr
-------+--------+--------+--------+------
  7369 | 喜洋洋 | 800.00 | 工程师 | 7902
  7900 | 小灰灰 | 950.00 | 工程师 | 7698

mydb=# select * from emp1 where sal between 800 and 1000;
 empno | ename  |  sal   |  job   | mgr
-------+--------+--------+--------+------
  7369 | 喜洋洋 | 800.00 | 工程师 | 7902
  7900 | 小灰灰 | 950.00 | 工程师 | 7698
```

```
mydb=# select * from emp1 where mgr=7902 or mgr=7566 or mgr=7788;
 empno | ename  |   sal   |  job   | mgr
-------+--------+---------+--------+------
  7369 | 喜洋洋 |  800.00 | 工程师 | 7902
  7788 | 瘦羊羊 | 3000.00 | 架构师 | 7566
  7876 | 红太狼 | 1100.00 | 工程师 | 7788

mydb=# select * from emp1 where mgr in (7902,7566,7788);
 empno | ename  |   sal   |  job   | mgr
-------+--------+---------+--------+------
  7369 | 喜洋洋 |  800.00 | 工程师 | 7902
  7788 | 瘦羊羊 | 3000.00 | 架构师 | 7566
  7876 | 红太狼 | 1100.00 | 工程师 | 7788
```

【OR】示例

openGauss

where子句示例六(逻辑限定操作符)

【NOT】示例

```
mydb=# select * from emp1 where job!='工程师' and job!='销售' and job!='部门经理';
 empno | ename  |   sal   |  job   | mgr
-------+--------+---------+--------+------
  7788 | 瘦羊羊 | 3000.00 | 架构师 | 7566
  7839 | 快羊羊 | 5000.00 | CEO    |
  7934 | Tom    | 1300.00 | CFO    | 7782

mydb=# select * from emp1 where job<>'工程师' and job<>'销售' and job<>'部门经理';
 empno | ename  |   sal   |  job   | mgr
-------+--------+---------+--------+------
  7788 | 瘦羊羊 | 3000.00 | 架构师 | 7566
  7839 | 快羊羊 | 5000.00 | CEO    |
  7934 | Tom    | 1300.00 | CFO    | 7782

mydb=# select * from emp1 where job not in ('工程师','销售','部门经理');
 empno | ename  |   sal   |  job   | mgr
-------+--------+---------+--------+------
  7788 | 瘦羊羊 | 3000.00 | 架构师 | 7566
  7839 | 快羊羊 | 5000.00 | CEO    |
  7934 | Tom    | 1300.00 | CFO    | 7782
```

```
mydb=# select ename,job,sal from emp1 where job='销售' or job='工程师' and sal>1500;
 ename  | job  |   sal
--------+------+---------
 美羊羊 | 销售 | 1600.00
 慢羊羊 | 销售 | 1250.00
 灰太狼 | 销售 | 1500.00
 懒羊羊 | 销售 | 1250.00

mydb=# select ename,job,sal from emp1 where (job='销售' or job='工程师') and sal>1500;
 ename  | job  |   sal
--------+------+---------
 美羊羊 | 销售 | 1600.00

mydb=# select ename,job,sal from emp1 where job in ('销售','工程师') and sal>1500;
 ename  | job  |   sal
--------+------+---------
 美羊羊 | 销售 | 1600.00
```

【优先级】示例

openGauss

查询结果排序(order by)

在select语句中，使用order by子句可以对输出的行进行排序，desc表示降序，asc表示升序(默认)。

order by子句跟在select语句之后，可以使用：列名、别名、列位置号或表达式等方式调用order by子句对输出结果进行排序。

如果有where限定查询子句，则order by子句跟在where子句后面，order by子句是 select 语句中最后一个子句。

Tips:

- select 语句中如果有 distinct 关键字，order by 后面的列必须出现在 select 子句中。
- order by 根据字符排序时，区分大小写。
- 如果要支持**中文拼音排序**，需要在初始化数据库时指定字符集为zh_CN.UTF-8或zh_CN.GBK。

initdb –E UTF8 –D ../data --locale=zh_CN.UTF-8 或 initdb –E GBK –D ../data --locale=zh_CN.GBK

或者重新创建数据库：

```
create database mydb with encoding='UTF-8' LC_COLLATE='zh_CN.UTF-8'  LC_CTYPE='zh_CN.UTF-8';
```

- **基本示例:**

```
mydb=# select * from emp order by empno limit 5;
 empno | ename  |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+----------+------+---------------------+---------+---------+--------
  7369 | 喜洋洋 | 工程师   | 7902 | 2018-12-17 00:00:00 |  800.00 |         |     20
  7499 | 美羊羊 | 销售     | 7698 | 2020-02-20 00:00:00 | 1600.00 |  300.00 |     30
  7521 | 懒羊羊 | 销售     | 7698 | 2020-12-22 14:41:17 | 1250.00 |  500.00 |     30
  7566 | 沸羊羊 | 部门经理 | 7839 | 2020-04-02 00:00:00 | 2975.00 |         |     20
  7654 | 慢羊羊 | 销售     | 7698 | 2020-09-08 00:00:00 | 1250.00 | 1400.00 |     30
```

openGauss

查询结果排序(order by)示例一

- **desc降序**

```
mydb=# select * from dept order by deptno desc;
 deptno | dname | loc
--------+-------+------
     40 | 生产  | 西安
     30 | 销售  | 武汉
     20 | 研发  | 上海
     10 | 财务  | 北京

mydb=# select * from dept order by 1 desc;
 deptno | dname | loc
--------+-------+------
     40 | 生产  | 西安
     30 | 销售  | 武汉
     20 | 研发  | 上海
     10 | 财务  | 北京

mydb=# select deptno,dname "Name",loc from dept order by "Name" desc;
 deptno | Name | loc
--------+------+------
     20 | 研发 | 上海
     30 | 销售 | 武汉
     40 | 生产 | 西安
     10 | 财务 | 北京
```

列的别名使用双引号
建议排序时别名也使用双引号
避免大小写和特殊字符造成语句错误

```
mydb=# explain select * from dept order by deptno desc;
                          QUERY PLAN
---------------------------------------------------------------
 Sort  (cost=1.08..1.09 rows=4 width=102)
   Sort Key: deptno DESC
   ->  Seq Scan on dept  (cost=0.00..1.04 rows=4 width=102)

mydb=# explain select * from dept order by 1 desc;
                          QUERY PLAN
---------------------------------------------------------------
 Sort  (cost=1.08..1.09 rows=4 width=102)
   Sort Key: deptno DESC
   ->  Seq Scan on dept  (cost=0.00..1.04 rows=4 width=102)

mydb=# explain select deptno,dname "Name",loc from dept order by "Name" desc;
                          QUERY PLAN
---------------------------------------------------------------
 Sort  (cost=1.08..1.09 rows=4 width=102)
   Sort Key: dname DESC
   ->  Seq Scan on dept  (cost=0.00..1.04 rows=4 width=102)
```

- **排序中的null值**（null值在列排序时，默认把 null 值看做无限大）

```
mydb=# select * from emp1;
 ename  |   sal   |  comm
--------+---------+---------
 灰太狼 | 1500.00 |    0.00
 慢羊羊 | 1250.00 | 1400.00
 沸羊羊 | 2975.00 |
 美羊羊 | 1600.00 |  300.00
 懒羊羊 | 1250.00 |  600.00

mydb=# select * from emp1 order by comm;
 ename  |   sal   |  comm
--------+---------+---------
 灰太狼 | 1500.00 |    0.00
 美羊羊 | 1600.00 |  300.00
 懒羊羊 | 1250.00 |  600.00
 慢羊羊 | 1250.00 | 1400.00
 沸羊羊 | 2975.00 |
```

```
mydb=# select * from emp1 order by comm nulls first;
 ename  |   sal   |  comm
--------+---------+---------
 沸羊羊 | 2975.00 |
 灰太狼 | 1500.00 |    0.00
 美羊羊 | 1600.00 |  300.00
 懒羊羊 | 1250.00 |  600.00
 慢羊羊 | 1250.00 | 1400.00

mydb=# select * from emp1 order by comm desc nulls last;
 ename  |   sal   |  comm
--------+---------+---------
 慢羊羊 | 1250.00 | 1400.00
 懒羊羊 | 1250.00 |  600.00
 美羊羊 | 1600.00 |  300.00
 灰太狼 | 1500.00 |    0.00
 沸羊羊 | 2975.00 |
```

openGauss

查询结果排序(order by)示例二

- **多个列排序:混合排序**
  - 首先根据最左边的列进行排序，如果这一列的值相同，则根据下一个表达式进行比较，依此类推。
  - 如果对于所有声明的表达式都相同，则按随机顺序返回。

```
mydb=# select * from emp1 order by deptno,sal desc;
 ename  |   sal    |  comm   | deptno
--------+----------+---------+--------
 沸羊羊 | 2975.00 |         |     20
 喜洋洋 |  800.00 |         |     20
 暖羊羊 | 2850.00 |         |     30
 美羊羊 | 1600.00 |  300.00 |     30
 慢羊羊 | 1250.00 | 1400.00 |     30

mydb=# select * from emp1 order by 4,2 desc;
 ename  |   sal    |  comm   | deptno
--------+----------+---------+--------
 沸羊羊 | 2975.00 |         |     20
 喜洋洋 |  800.00 |         |     20
 暖羊羊 | 2850.00 |         |     30
 美羊羊 | 1600.00 |  300.00 |     30
 慢羊羊 | 1250.00 | 1400.00 |     30
```

```
mydb=# select * from emp1 order by comm desc nulls last,deptno asc;
 ename  |   sal    |  comm   | deptno
--------+----------+---------+--------
 慢羊羊 | 1250.00 | 1400.00 |     30
 美羊羊 | 1600.00 |  300.00 |     30
 喜洋洋 |  800.00 |         |     20
 沸羊羊 | 2975.00 |         |     20
 暖羊羊 | 2850.00 |         |     30

mydb=# select * from emp1 order by comm desc nulls last,deptno desc;
 ename  |   sal    |  comm   | deptno
--------+----------+---------+--------
 慢羊羊 | 1250.00 | 1400.00 |     30
 美羊羊 | 1600.00 |  300.00 |     30
 暖羊羊 | 2850.00 |         |     30
 喜洋洋 |  800.00 |         |     20
 沸羊羊 | 2975.00 |         |     20
```

openGauss

PART 2

多表连接查询

openGauss

多表连接查询

多表连接查询，即SQL语句用一个连接从多个表中获取数据。
常见的多表连接查询类型如下：

连接类型	基本概念
等值/非等值连接	这两种连接同时包含在内连接和外连接中，因为内连接和外连接都是需要连接条件的，条件为 = 则为等值连接，反之为非等值连接。
自然连接	等值连接的一种，使用natural join后面可以不使用on接查询条件，默认会将关联表中的相同字段进行比较，查询出的结果相同的字段会去重（值必须相等）。
内连接	使用inner join和join连接都行，重点是要有查询条件，条件使用on或者where引导查询都行，查询出的结果为两表都匹配的记录。
外连接	分为左外连接、右外连接、全外连接等，要有查询条件，条件只能使用on引导查询。左外连接查询出的结果除了两表都匹配的记录外，还会查询出左表的其余记录，同时右表对应记录置为null,左外连接则相反。
内连接、外连接实际上都是在笛卡尔积（join）的基础上对记录进行筛选。	

- **基本语法：**

```
SELECT   table1.column,table2.column
FROM     table1,table2
WHERE    table1.column1=table2.column2;
```

Tips：如果某个列的名字在多个表中出现,那么需要在列的名字前面加上表名作为前缀。

多表连接查询笛卡尔积现象

笛卡尔积连接特征：

1> 连接条件被忽略；

2> 第一个表中的所有的行与第二个表中的所有的行相连接。

Tips：如果在WHERE子句中加入条件，那么久可以避免笛卡尔积。

```
postgres=# select * from emp1;
 empno | ename  |   sal   | deptno
-------+--------+---------+--------
  7369 | 喜洋洋 |  800.00 |     20
  7499 | 美羊羊 | 1600.00 |     30
```

```
postgres=# select * from dept1;
 deptno | dname | loc
--------+-------+------
     10 | 财务  | 北京
     20 | 研发  | 上海
     30 | 销售  | 武汉
```

```
postgres=# select * from emp1,dept1;
 empno | ename  |   sal   | deptno | deptno | dname | loc
-------+--------+---------+--------+--------+-------+------
  7369 | 喜洋洋 |  800.00 |     20 |     10 | 财务  | 北京
  7369 | 喜洋洋 |  800.00 |     20 |     20 | 研发  | 上海
  7369 | 喜洋洋 |  800.00 |     20 |     30 | 销售  | 武汉
  7499 | 美羊羊 | 1600.00 |     30 |     10 | 财务  | 北京
  7499 | 美羊羊 | 1600.00 |     30 |     20 | 研发  | 上海
  7499 | 美羊羊 | 1600.00 |     30 |     30 | 销售  | 武汉
```

多表连接查询(自然连接)

```
postgres=# select * from salgrade;
 grade | losal | hisal
-------+-------+-------
     1 |   700 |  1200
     2 |  1201 |  1400
     3 |  1401 |  2000
     4 |  2001 |  3000
     5 |  3001 |  9999
```

```
postgres=# select e.ename,e.sal,s.losal,s.grade
           from emp e join salgrade s
           on (e.sal between s.losal and s.hisal)
           where deptno in (10,20);
 ename  |   sal   | losal | grade
--------+---------+-------+-------
 喜洋洋 |  800.00 |   700 |     1
 红太狼 | 1100.00 |   700 |     1
 Tom    | 1300.00 |  1201 |     2
 沸羊羊 | 2975.00 |  2001 |     4
 花羊羊 | 2450.00 |  2001 |     4
 瘦羊羊 | 3000.00 |  2001 |     4
 Jerry  | 3000.00 |  2001 |     4
 快羊羊 | 5000.00 |  3001 |     5
```

```
postgres=# select * from emp;
 empno | ename  |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+----------+------+---------------------+---------+---------+--------
  7369 | 喜洋洋 | 工程师   | 7902 | 2018-12-17 00:00:00 |  800.00 |         |     20
  7499 | 美羊羊 | 销售     | 7698 | 2020-02-20 00:00:00 | 1600.00 |  300.00 |     30
  7521 | 懒羊羊 | 销售     | 7698 | 2020-02-21 00:00:00 | 1250.00 |  500.00 |     30
  7566 | 沸羊羊 | 部门经理 | 7839 | 2020-04-02 00:00:00 | 2975.00 |         |     20
  7654 | 慢羊羊 | 销售     | 7698 | 2020-09-08 00:00:00 | 1250.00 | 1400.00 |     30
  7698 | 暖羊羊 | 部门经理 | 7839 | 2020-05-01 00:00:00 | 2850.00 |         |     30
  7782 | 花羊羊 | 部门经理 | 7839 | 2020-06-08 00:00:00 | 2450.00 |         |     10
  7788 | 瘦羊羊 | 架构师   | 7566 | 2010-09-09 00:00:00 | 3000.00 |         |     20
  7839 | 快羊羊 | CEO      |      | 2020-11-09 00:00:00 | 5000.00 |         |     10
  7844 | 灰太狼 | 销售     | 7698 | 2020-09-11 00:00:00 | 1500.00 |    0.00 |     30
  7876 | 红太狼 | 工程师   | 7788 | 2009-01-01 00:00:00 | 1100.00 |         |     20
  7900 | 小灰灰 | 工程师   | 7698 | 2016-05-21 00:00:00 |  950.00 |         |     30
  7902 | Jerry  | CTO      | 7566 | 2017-06-08 00:00:00 | 3000.00 |         |     20
  7934 | Tom    | CFO      | 7782 | 2011-09-11 00:00:00 | 1300.00 |         |     10
```

```
postgres=# explain select e.ename,e.sal,s.losal,s.grade
           from emp e join salgrade s
           on (e.sal between s.losal and s.hisal)
           where deptno in (10,20);
                                  QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..3.14 rows=4 width=22)
   Join Filter: ((e.sal >= (s.losal)::numeric) AND (e.sal <= (s.hisal)::numeric))
   ->  Seq Scan on emp e  (cost=0.00..1.18 rows=8 width=14)
         Filter: (deptno = ANY ('{10,20}'::numeric[]))
   ->  Materialize  (cost=0.00..1.07 rows=5 width=12)
         ->  Seq Scan on salgrade s  (cost=0.00..1.05 rows=5 width=12)
```

openGauss

多表连接查询(等值/非等值连接)

➢ **等值连接**

-- 查询10号部门员工的姓名和部门名称

```
mydb=# select emp.ename,emp.deptno,dept.deptno,dept.dname
mydb-# from emp,dept
mydb-# where emp.deptno=dept.deptno
mydb-# and emp.deptno=10;
 ename  | deptno | deptno | dname
--------+--------+--------+-------
 花羊羊 |     10 |     10 | 财务
 快羊羊 |     10 |     10 | 财务
 Tom    |     10 |     10 | 财务
(3 rows)
```

➢ **非等值连接 【运算符可以包括>、>=、<=、<和<>】**

-- 查询员工的工资以及对应的工资等级

```
mydb=# select e.empno,e.ename,e.sal,s.losal,s.hisal,s.grade
mydb-# from emp e,salgrade s
mydb-# where e.sal>=s.losal and e.sal<=s.hisal
mydb-# limit 3;
 empno | ename  |   sal   | losal | hisal | grade
-------+--------+---------+-------+-------+-------
  7369 | 喜洋洋 |  800.00 |   700 |  1200 |     1
  7876 | 红太狼 | 1100.00 |   700 |  1200 |     1
  7900 | 小灰灰 |  950.00 |   700 |  1200 |     1
(3 rows)
```

多表连接查询(SQL1999)

➢ **SQL1999语法介绍**

```
select      查询列表
from        表1  别名  [连接类型]
join        表2  别名
on          连接条件
where       [筛选条件]
group by    [分组]
having      [筛选条件]
order by    [排序列表]

内连接:    inner
外连接:
            左外:  left [outer]
            右外:  right[outer]
            全外:  full [outer]
交叉连接:  cross
```

➢ **SQL1999 写法(标准 SQL)**

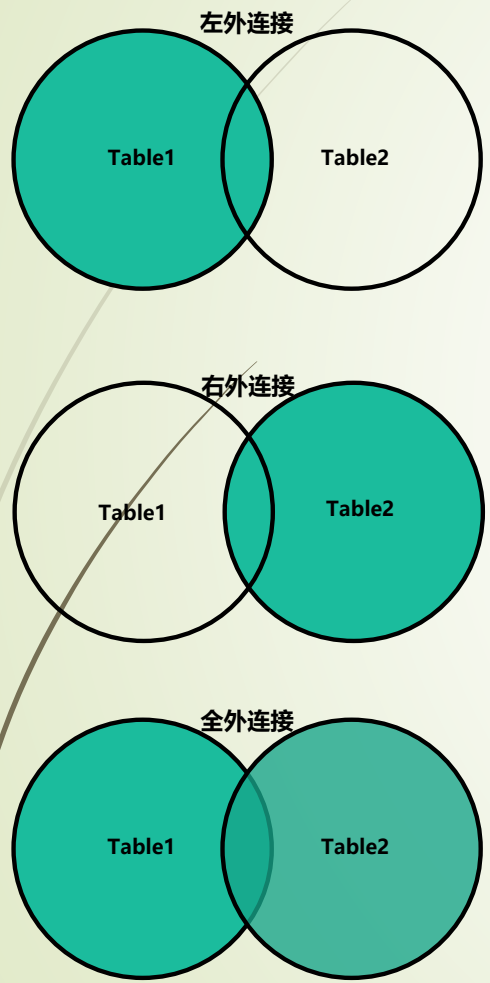
-- 查询10号部门员工的姓名和部门名称

```
mydb=# select emp.ename,emp.deptno,dept.deptno,dept.dname
mydb-# from   emp inner join dept
mydb-# on     (emp.deptno=dept.deptno)
mydb-# where  emp.deptno=10;
 ename  | deptno | deptno | dname
--------+--------+--------+-------
 花羊羊 |     10 |     10 | 财务
 快羊羊 |     10 |     10 | 财务
 Tom    |     10 |     10 | 财务
(3 rows)
mydb=# select emp.ename,deptno,dept.dname
mydb-# from   emp inner join dept
mydb-# on     (emp.deptno=dept.deptno)
mydb-# where  emp.deptno=10;
ERROR:  column reference "deptno" is ambiguous
LINE 1: select emp.ename,deptno,dept.dname
                         ^
CONTEXT:  referenced column: deptno
```

-- 当联接表的列同名，则可以使用 USING 语法来简化 ON 语法

```
mydb=# select emp.ename,deptno,dept.dname
mydb-# from   emp inner join dept
mydb-# using(deptno)
mydb-# where emp.deptno=10;
 ename  | deptno | dname
--------+--------+-------
 花羊羊 |     10 | 财务
 快羊羊 |     10 | 财务
 Tom    |     10 | 财务
(3 rows)
```

外连接查询



外连接分为左外连接(LEFT OUTER JOIN)、右外连接(RIGHT OUTER JOIN)、全外连接(FULL OUTER JOIN)三种。

外连接常用来查出在一个表中,外连接不只列出与条件相匹配的行,而且还列出左表(左外连接时)、右表(右外连接时)、两个表(全外连接时)所有符合搜索条件的数据行。

外连接分为左外连接(LEFT OUTER JOIN)、右外连接(RIGHT OUTER JOIN)、全外连接(FULL OUTER JOIN)三种。外连接常用来查出在一个表中,外连接不只列出与条件相匹配的行,而且还列出左表(左外连接时)、右表(右外连接时)、两个表(全外连接时)所有符合搜索条件的数据行。

- **SQL1999语法**

右外连接语法:

```
SELECT  table1.column , table2.column
FROM    table1 right outer join table2
on   (table1.column = table2.column);
```

左外连接语法:

```
SELECT table1.column , table2.column
FROM   table1 left outer join table2
on   (table1.column = table2.column);
```

全外连接语法:

```
SELECT table1.column , table2.column
FROM   table1 full outer join table2
on   (table1.column = table2.column);
```

右外连接语法2:

```
SELECT table1.column , table2.column
FROM   table1,table2
WHERE  table1.column(+)=table2.column;
```

左外连接语法2:

```
SELECT table1.column , table2.column
FROM   table1,table2
WHERE  table1.column = table2.column(+);
```

openGauss

外连接查询(全外连接)

➢ 测试表emp1

```
mydb=# select * from emp1;
 empno | ename  |    job     | mgr  |   sal   |  comm   | deptno
-------+--------+------------+------+---------+---------+--------
  7369 | 喜洋洋 | 工程师     | 7902 |  800.00 |         |     20
  7499 | 美羊羊 | 销售       | 7698 | 1600.00 |  300.00 |     30
  7566 | 沸羊羊 | 部门经理   | 7839 | 2975.00 |         |     20
  7654 | 慢羊羊 | 销售       | 7698 | 1250.00 | 1400.00 |     30
  7698 | 暖羊羊 | 部门经理   | 7839 | 2850.00 |         |     30
(5 rows)
```

➢ 测试表dept1

```
mydb=# select * from dept1;
 deptno | dname | loc
--------+-------+------
     20 | 研发  | 上海
     40 | 生产  | 西安
(2 rows)
```

➢ 全外连接查询

```
mydb=# select e.ename,e.sal,d.deptno,d.dname
mydb-# from emp1 e full outer join dept1 d
mydb-# on (e.deptno=d.deptno);
 ename  |   sal   | deptno | dname
--------+---------+--------+-------
 沸羊羊 | 2975.00 |     20 | 研发
 喜洋洋 |  800.00 |     20 | 研发
        |         |     40 | 生产
 暖羊羊 | 2850.00 |        |
 慢羊羊 | 1250.00 |        |
 美羊羊 | 1600.00 |        |
(6 rows)
```

openGauss

外连接查询(左外|右外连接)

- **左外连接查询**

```
mydb=# select e.ename,e.sal,d.deptno,d.dname
mydb-# from emp1 e left outer join dept1 d
mydb-# on (e.deptno=d.deptno);
 ename  |   sal    | deptno | dname
--------+----------+--------+-------
 沸羊羊 | 2975.00 |     20 | 研发
 喜洋洋 |  800.00 |     20 | 研发
 暖羊羊 | 2850.00 |        |
 慢羊羊 | 1250.00 |        |
 美羊羊 | 1600.00 |        |
(5 rows)
```

```
mydb=# select e.ename,e.sal,d.deptno,d.dname
mydb-# from emp1 e,dept1 d
mydb-# where e.deptno=d.deptno(+);
 ename  |   sal    | deptno | dname
--------+----------+--------+-------
 沸羊羊 | 2975.00 |     20 | 研发
 喜洋洋 |  800.00 |     20 | 研发
 暖羊羊 | 2850.00 |        |
 慢羊羊 | 1250.00 |        |
 美羊羊 | 1600.00 |        |
(5 rows)
```

- **右外连接查询**

```
mydb=# select e.ename,e.sal,d.deptno,d.dname
mydb-# from emp1 e right outer join dept1 d
mydb-# on (e.deptno=d.deptno);
 ename  |   sal    | deptno | dname
--------+----------+--------+-------
 沸羊羊 | 2975.00 |     20 | 研发
 喜洋洋 |  800.00 |     20 | 研发
        |         |     40 | 生产
(3 rows)
```

```
mydb=# select e.ename,e.sal,d.deptno,d.dname
mydb-# from emp1 e,dept1 d
mydb-# where e.deptno(+)=d.deptno;
 ename  |   sal    | deptno | dname
--------+----------+--------+-------
 沸羊羊 | 2975.00 |     20 | 研发
 喜洋洋 |  800.00 |     20 | 研发
        |         |     40 | 生产
(3 rows)
```

openGauss

自连接查询和多表关联查询

- **自连接查询**

-- 查询员工姓名以及其经理姓名

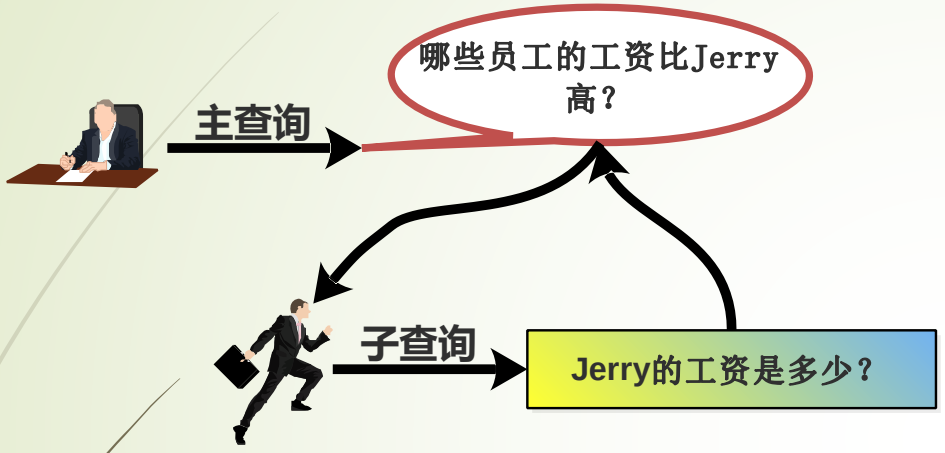
```
mydb=# select e.empno,e.ename,e.mgr mgrno,m.ename mgrname
mydb-# from emp e,emp m
mydb-# where e.mgr=m.empno;
 empno | ename  | mgrno | mgrname
-------+--------+-------+---------
  7369 | 喜洋洋 |  7902 | Jerry
  7499 | 美羊羊 |  7698 | 暖羊羊
  7566 | 沸羊羊 |  7839 | 快羊羊
  7654 | 慢羊羊 |  7698 | 暖羊羊
  7698 | 暖羊羊 |  7839 | 快羊羊
  7782 | 花羊羊 |  7839 | 快羊羊
  7788 | 瘦羊羊 |  7566 | 沸羊羊
  7844 | 灰太狼 |  7698 | 暖羊羊
  7876 | 红太狼 |  7788 | 瘦羊羊
  7900 | 小灰灰 |  7698 | 暖羊羊
  7902 | Jerry  |  7566 | 沸羊羊
  7934 | Tom    |  7782 | 花羊羊
  7521 | 懒羊羊 |  7698 | 暖羊羊
(13 rows)
```

- **多表关联查询**

-- 查询员工的姓名、部门名称、工资等级

```
mydb=# select e.ename,d.dname,s.grade
mydb-# from emp e,dept d,salgrade s
mydb-# where (e.sal between s.losal and s.hisal)
mydb-# and (e.deptno=d.deptno);
 ename  | dname | grade
--------+-------+-------
 喜洋洋 | 研发  |     1
 红太狼 | 研发  |     1
 小灰灰 | 销售  |     1
 慢羊羊 | 销售  |     2
 Tom    | 财务  |     2
 懒羊羊 | 销售  |     2
 美羊羊 | 销售  |     3
 灰太狼 | 销售  |     3
 沸羊羊 | 研发  |     4
 暖羊羊 | 销售  |     4
 花羊羊 | 财务  |     4
 瘦羊羊 | 研发  |     4
 Jerry  | 研发  |     4
 快羊羊 | 财务  |     5
(14 rows)
```

子查询



子查询基本语法：

```
SELECT    select_list
FROM      table
WHERE     expr operator
(SELECT   select_list FROM table);
```

说明：

- ✓ 在主查询执行之前先执行子查询；
- ✓ 主查询语句中调用子查询的结果。

查询示例SQL语句：

```
postgres=# select ename,sal
           from emp
           where sal > (select sal
                        from emp
                        where ename='Jerry');
 ename  |   sal
--------+---------
 快羊羊 | 5000.00
```

- ➢ **子查询使用建议：**
- ✓ 子查询通常包括在括号里面；
- ✓ 子查询通常放在比较操作符的右边；
- ✓ 子查询中不建议包含ORDER BY子句；
- ✓ 对于单行的子查询使用单行的比较操作符；
- ✓ 对于多行的子查询要使用多行的比较操作符。

openGauss

子查询示例一

- 单行子查询

-- 查询工作和7369相同且工资比 7900 高的员工

```
mydb=# select ename,sal from emp
mydb-# where job = (select job from emp where empno=7369)
mydb-# and sal > (select sal from emp where empno=7900);
 ename  |   sal
--------+---------
 红太狼 | 1100.00
(1 row)
```

- 多行子查询

操作符	含义
IN	等于列表中的某一个值
ANY	与列表中的任意值比较
ALL	与列表中的所有值相比较

注意：如果表中存在空值，则可能因为空值的比较没有意义，可能返回空值，即未选定任何行。

```
mydb=# -- 查询是经理的员工
mydb=# select * from emp1
mydb-# where empno in (select mgr from emp1);
 empno | ename  |   job    | mgr  |   sal   | comm | deptno
-------+--------+----------+------+---------+------+--------
  7698 | 暖羊羊 | 部门经理 | 7839 | 2850.00 |      |     30
```

```
mydb=# -- 查询非销售人员中，工资不高于所有销售岗位的人员信息
mydb=# select * from emp1
mydb-# where sal < any (select sal from emp1 where job='销售') and job<>'销售';
 empno | ename  |  job   | mgr  |  sal   | comm | deptno
-------+--------+--------+------+--------+------+--------
  7369 | 喜洋洋 | 工程师 | 7902 | 800.00 |      |     20
```

```
mydb=# -- 查询工资比各部门平均工资都高的员工
mydb=# select empno,job,sal from emp1
mydb-# where sal > all (select avg(sal) from emp1 group by deptno);
 empno |   job    |   sal
-------+----------+---------
  7566 | 部门经理 | 2975.00
  7698 | 部门经理 | 2850.00
```

openGauss

子查询示例二

- 查询平均工资最低的岗位

```
mydb=# select job,avg(sal) from emp
mydb-# group by job
mydb-# having avg(sal)=(select min(jobavg)
mydb(#                  from (select avg(sal) jobavg from emp group by job));
  job   |         avg
--------+----------------------
 工程师 | 950.0000000000000000
```

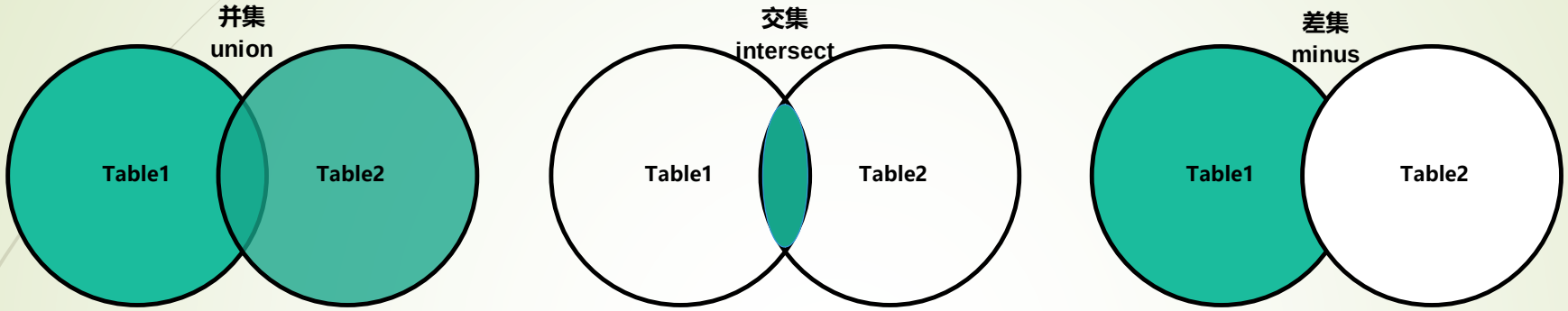
- 查询员工及其部门信息

```
mydb=# select e.*,(select d.dname from dept d where e.deptno = d.deptno)
mydb-# from emp1 e;
 empno | ename  |   job    | mgr  |   sal   |  comm   | deptno | dname
-------+--------+----------+------+---------+---------+--------+-------
  7369 | 喜洋洋 | 工程师   | 7902 |  800.00 |         |     20 | 研发
  7499 | 美羊羊 | 销售     | 7698 | 1600.00 |  300.00 |     30 | 销售
  7566 | 沸羊羊 | 部门经理 | 7839 | 2975.00 |         |     20 | 研发
  7654 | 慢羊羊 | 销售     | 7698 | 1250.00 | 1400.00 |     30 | 销售
  7698 | 暖羊羊 | 部门经理 | 7839 | 2850.00 |         |     30 | 销售
```

- 查询经理的个人信息

```
mydb=# -- 查询经理的个人信息
mydb=# select m.empno,m.ename,m.sal,m.job from emp m
mydb-# where exists (select 1 from emp e where e.mgr=m.empno)
mydb-# limit 3;
 empno | ename  |   sal   |   job
-------+--------+---------+----------
  7566 | 沸羊羊 | 2975.00 | 部门经理
  7698 | 暖羊羊 | 2850.00 | 部门经理
  7782 | 花羊羊 | 2450.00 | 部门经理
```

复合查询



- 测试表数据

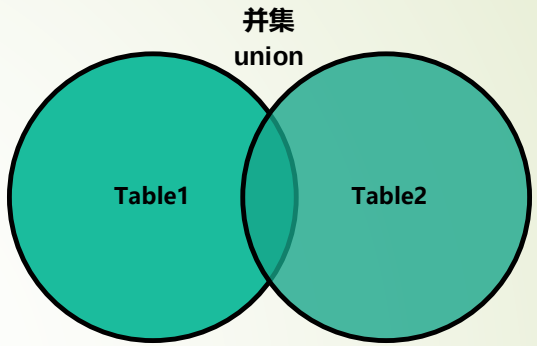
```
mydb=# select * from emp1;
 empno | ename  |   job    |   sal   | deptno
-------+--------+----------+---------+--------
  7369 | 喜洋洋 | 工程师   |  800.00 |     20
  7902 | Jerry  | CTO      | 3000.00 |     20
  7839 | 快羊羊 | CEO      | 5000.00 |     10
  7782 | 花羊羊 | 部门经理 | 2450.00 |     10
```

```
mydb=# select * from emp2;
 empno | ename  |   job    |   sal   | deptno
-------+--------+----------+---------+--------
  7369 | 喜洋洋 | 工程师   |  800.00 |     20
  7902 | Jerry  | CTO      | 3000.00 |     20
  7499 | 美羊羊 | 销售     | 1600.00 |     30
  7521 | 懒羊羊 | 销售     | 1250.00 |     30
```

openGauss

复合查询(并集union)

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。



➢ **一般形式**

```
select_statement
UNION [ALL]
select_statement;
```

➢ **限制要求**

✓ UNION 涉及的表，其列的顺序、列名、数据类型必须相同。
✓ UNION 子句默认不包含重复的行，除非声明了ALL子句。

➢ **union并集查询(默认去重)**

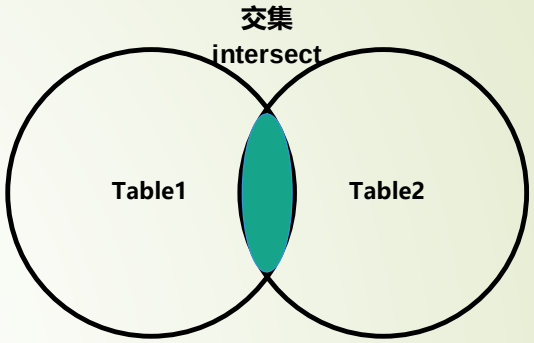
```
mydb=# select * from emp1
mydb-# union
mydb-# select * from emp2;
 empno | ename  |   job    |   sal   | deptno
-------+--------+----------+---------+--------
  7839 | 快羊羊 | CEO      | 5000.00 |     10
  7499 | 美羊羊 | 销售     | 1600.00 |     30
  7902 | Jerry  | CTO      | 3000.00 |     20
  7369 | 喜洋洋 | 工程师   |  800.00 |     20
  7782 | 花羊羊 | 部门经理 | 2450.00 |     10
  7521 | 懒羊羊 | 销售     | 1250.00 |     30
```

➢ **union all并集查询(不去重)**

```
mydb=# select * from emp1
mydb-# union all
mydb-# select * from emp2;
 empno | ename  |   job    |   sal   | deptno
-------+--------+----------+---------+--------
  7369 | 喜洋洋 | 工程师   |  800.00 |     20
  7902 | Jerry  | CTO      | 3000.00 |     20
  7839 | 快羊羊 | CEO      | 5000.00 |     10
  7782 | 花羊羊 | 部门经理 | 2450.00 |     10
  7369 | 喜洋洋 | 工程师   |  800.00 |     20
  7902 | Jerry  | CTO      | 3000.00 |     20
  7499 | 美羊羊 | 销售     | 1600.00 |     30
  7521 | 懒羊羊 | 销售     | 1250.00 |     30
```

复合查询(交集intersect)

交集intersect用来计算多个SELECT语句返回行集合的交集，不含重复的记录。



一般形式：

```
select_statement
INTERSECT
select_statement;
```

select_statement可以是任何没有FOR UPDATE子句的SELECT语句。

- **intersect 差集查询示例**

```
mydb=# select * from emp1
mydb-# intersect
mydb-# select * from emp2;
 empno | ename  |  job   |   sal   | deptno
-------+--------+--------+---------+--------
  7369 | 喜洋洋 | 工程师 |  800.00 |     20
  7902 | Jerry  | CTO    | 3000.00 |     20
```

- **约束条件**
  - SQL语句中的多个intersect操作符是从左向右计算的，除非用圆括号进行标识。
  - 当对多个SELECT语句的执行结果进行union和intersect操作的时候，会优先处理intersect。

复合查询(差集minus)

MINUS子句(与EXCEPT子句具有相同的功能和用法)

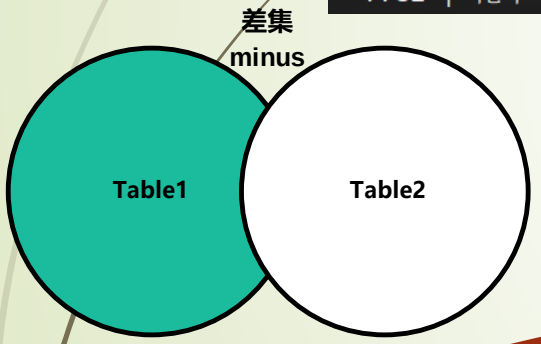
➢ **一般形式**

```
select_statement
minus [ all ]
select_statement;
```

- ✓ select_statement是任何没有FOR UPDATE子句的select表达式。
- ✓ minus操作符计算存在于左边select语句的输出而存在于右边select语句不输出。
- ✓ minus的结果不包含任何重复的行，除非声明了all选项。
- ✓ 除非用圆括号指明顺序，否则同一个select语句中的多个minus操作符是从左向右计算的。

```
mydb=# select * from emp1;
 empno | ename  |   job    |   sal   | deptno
-------+--------+----------+---------+--------
  7369 | 喜洋洋 | 工程师   |  800.00 |     20
  7902 | Jerry  | CTO      | 3000.00 |     20
  7839 | 快羊羊 | CEO      | 5000.00 |     10
  7782 | 花羊羊 | 部门经理 | 2450.00 |     10
```

```
mydb=# select * from emp2;
 empno | ename  |  job   |   sal   | deptno
-------+--------+--------+---------+--------
  7369 | 喜洋洋 | 工程师 |  800.00 |     20
  7902 | Jerry  | CTO    | 3000.00 |     20
  7499 | 美羊羊 | 销售   | 1600.00 |     30
  7521 | 懒羊羊 | 销售   | 1250.00 |     30
  7521 | 懒羊羊 | 销售   | 1250.00 |     30
```



差集minus查询

```
mydb=# select * from emp1
mydb-# minus
mydb-# select * from emp2;
 empno | ename  |   job    |   sal   | deptno
-------+--------+----------+---------+--------
  7839 | 快羊羊 | CEO      | 5000.00 |     10
  7782 | 花羊羊 | 部门经理 | 2450.00 |     10
```

```
mydb=# select * from emp2
mydb-# minus
mydb-# select * from emp1;
 empno | ename  | job  |   sal   | deptno
-------+--------+------+---------+--------
  7499 | 美羊羊 | 销售 | 1600.00 |     30
  7521 | 懒羊羊 | 销售 | 1250.00 |     30
```

```
mydb=# select * from emp2
mydb-# minus all
mydb-# select * from emp1;
 empno | ename  | job  |   sal   | deptno
-------+--------+------+---------+--------
  7499 | 美羊羊 | 销售 | 1600.00 |     30
  7521 | 懒羊羊 | 销售 | 1250.00 |     30
  7521 | 懒羊羊 | 销售 | 1250.00 |     30
```

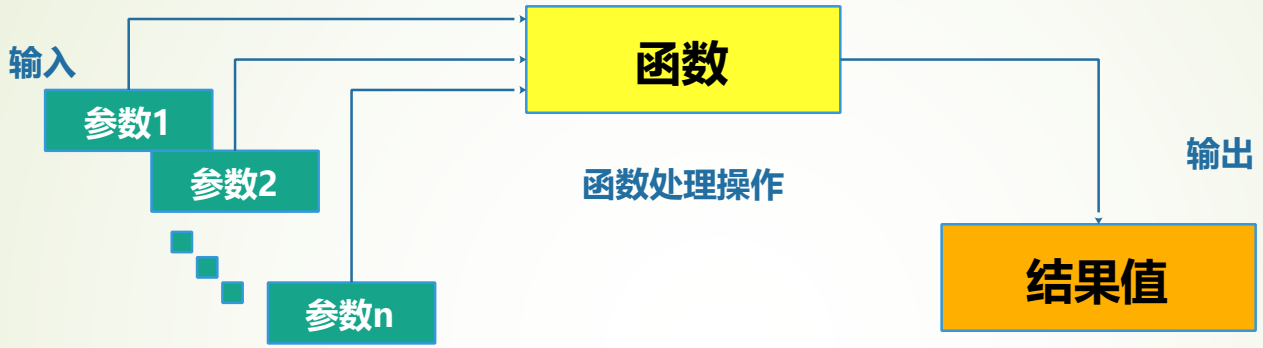
openGauss

PART 3

SQL 函数

单行函数(概述)

SQL函数这里指的是数据库内置函数，可以运用在SQL语句中实现特定的功能。

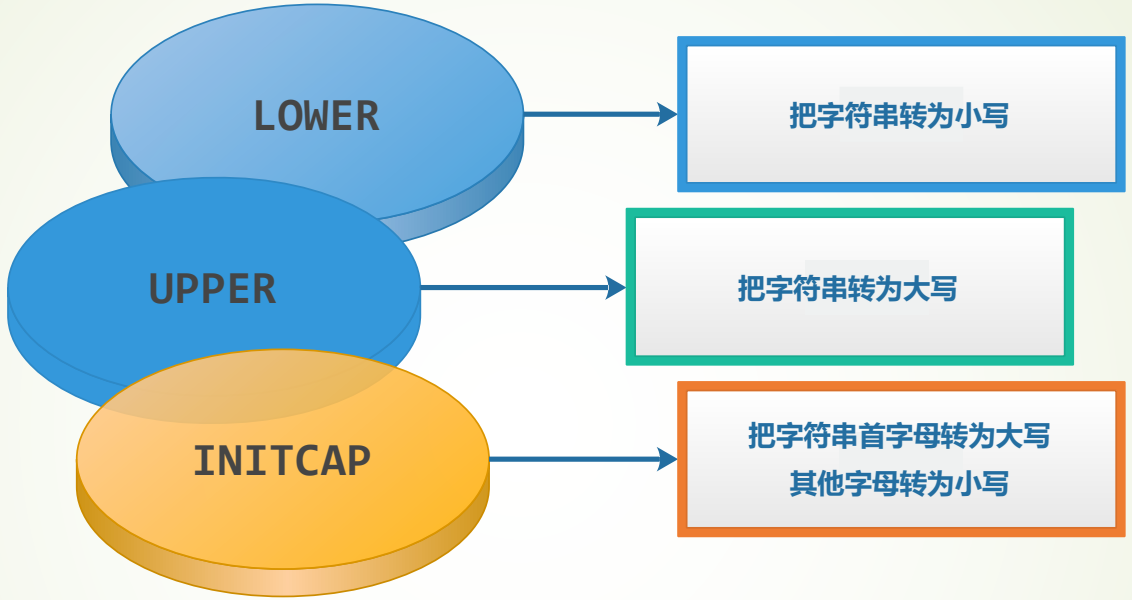


单行函数对于每一行数据进行计算后得到一行输出结果。SQL单行函数根据数据类型分为**字符函数**、**数值函数**、**日期函数**、**转换函数**以及**其他通用的函数等**。

➢ **单行函数基本特性**

- 单行函数对单行操作
- 每行返回一个结果
- 有可能返回值与原参数数据类型不一致（转换函数）
- 单行函数可以写在SELECT、WHERE、ORDER BY子句中
- 有些函数没有参数，有些函数包括一个或多个参数
- 函数可以嵌套

单行函数(大小写转换函数)



- 使用示例

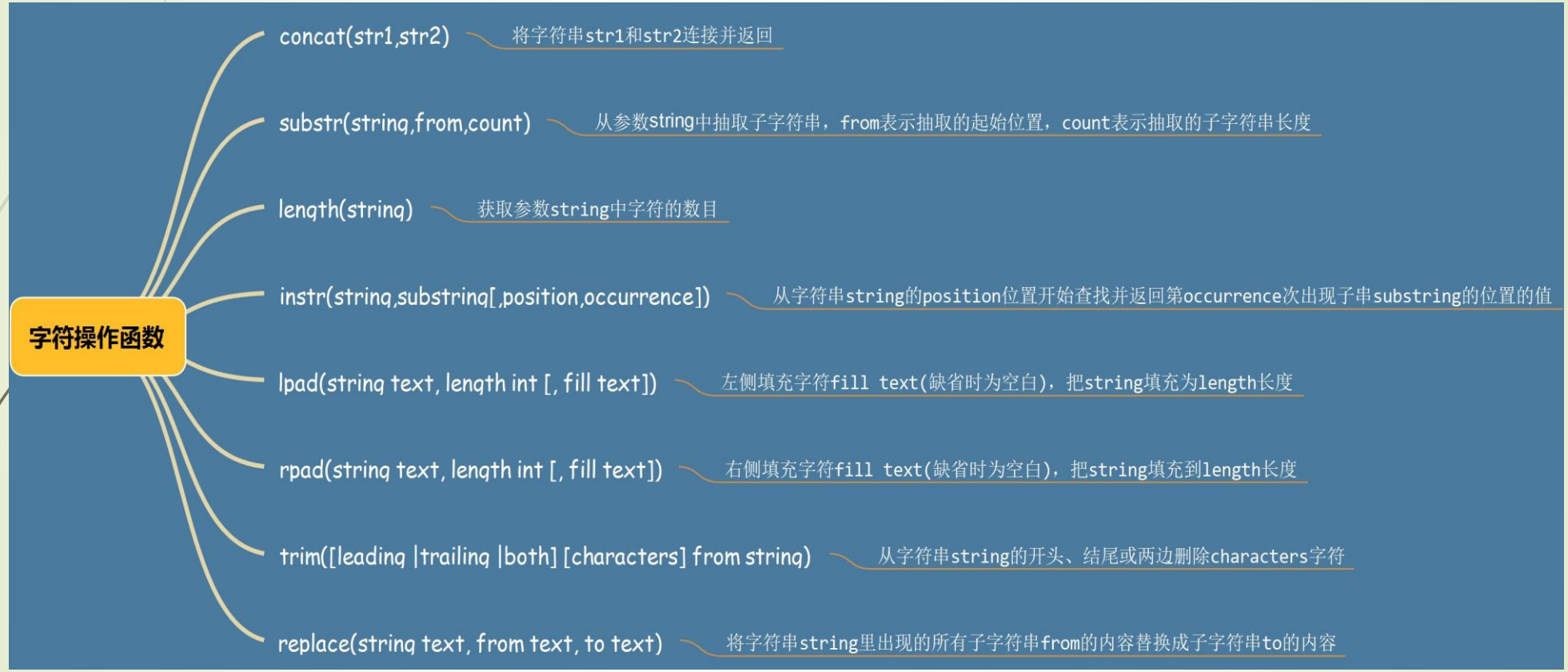
```
mydb=# select ename,lower(ename),upper(ename),initcap(ename) from emp where ename in ('Jerry','Tom');
 ename | lower | upper | initcap
-------+-------+-------+---------
 Jerry | jerry | JERRY | Jerry
 Tom   | tom   | TOM   | Tom
```

- initcap在单词分隔符语句中的示例

```
mydb=# select  initcap('hello,open%gauss!dba$this is _test');
              initcap
-------------------------------------
 Hello,Open%Gauss!Dba$This Is _Test
```

openGauss

单行函数(字符操作函数)



openGauss

单行函数(字符操作函数)示例1

- 【concat】连接字符串

```
mydb=# select concat(concat(ename,'的工作是'),job) from emp where ename='Jerry';
      concat
------------------
 Jerry的工作是CTO
```

- 【substr】截取子串

```
mydb=# select 'openGauss' as "char",substr('openGauss',1,3) as "substr(1,3)",substr('openGauss',3) as "substr(3)";
   char    | substr(1,3) | substr(3)
-----------+-------------+-----------
 openGauss | ope         | enGauss

mydb=# select 'openGauss' as "char",substr('openGauss',-3,2) as "substr(-3,2)",substr('openGauss',-5) as "substr(-5)";
   char    | substr(-3,2) | substr(-5)
-----------+--------------+------------
 openGauss | us           | Gauss
```

- 【length】获取字符串长度

```
mydb=# insert into t1 values(1,'Tom'),(2,''),(3,null),(4,' '),(5,'姓 名');
INSERT 0 5
mydb=# select id,name,nvl(length(name),0) from t1;
 id | name  | nvl
----+-------+-----
  1 | Tom   |   3
  2 |       |   0
  3 |       |   0
  4 |       |   1
  5 | 姓 名 |   3
```

- 【replace】替换

```
mydb=# select id,name,replace(name,' ','%') from t1;
 id | name  | replace
----+-------+---------
  1 | Tom   | Tom
  2 |       |
  3 |       |
  4 |       | %
  5 | 姓 名 | 姓%名
```

openGauss

单行函数(字符操作函数)示例2

- 【instr】子串在字符串第一次出现的位置

```
mydb=# select 'openGauss' "char",instr('openGauss','s') "default",instr('openGauss','s',1,2) "instr(1,2)";
   char    | default | instr(1,2)
-----------+---------+-----------
 openGauss |       8 |          9

mydb=# select 'openGauss' "char",instr('openGauss','s',-1) "instr(-1)",instr('openGauss','s',-3,2) "instr(-3,2)";
   char    | instr(-1) | instr(-3,2)
-----------+-----------+-------------
 openGauss |         9 |           0
```

倒数第3位往前逆向搜索

- 【lpad,rpad 】左补全，右补全

```
mydb=# select lpad('openGauss',12,'*') lpad12,rpad('openGauss',12,'*') lpad12,
              rpad('openGauss',8,'*')  rpad8 ,lpad('openGauss',8,'*')  lpad8;
    lpad12    |    lpad12    |  rpad8   |  lpad8
--------------+--------------+----------+----------
 ***openGauss | openGauss*** | openGaus | openGaus
```

如果string已经比length长则将其从尾部截断

- 【trim】去掉首尾空格或指定字符

```
mydb=# \pset expanded
Expanded display is on.
mydb=# select trim(leading '$' from '$openGauss$'),
mydb-#        trim(trailing '$' from '$openGauss$$$');
-[ RECORD 1 ]-----
ltrim | openGauss$
rtrim | $openGauss
```

```
mydb=# select trim('  openGauss  '),
mydb-#        trim('$' from '$$openGauss$$$$'),
mydb-#        trim(both '$' from '$$$$openGauss$');
-[ RECORD 1 ]----
btrim | openGauss
btrim | openGauss
btrim | openGauss
```

Tips：要去除的串 trim_character 只能是单个字符

openGauss

单行函数(数字操作函数)

- **ROUND：对指定的值进行四舍五入**

```
mydb=# select round(45.926,2) "round(45.926,2)",round(45.926,0) "round(45.926,0)",round(45.926,-1) "round(45.926,-1)";
 round(45.926,2) | round(45.926,0) | round(45.926,-1)
-----------------+-----------------+------------------
           45.93 |              46 |               50
```

- **TRUNC:对指定的值进行截断取整**

```
mydb=# select trunc(45.926,2) "trunc(45.926,2)",trunc(45.926,0) "trunc(45.926,0)",trunc(45.926,-1) "trunc(45.926,-1)";
 trunc(45.926,2) | trunc(45.926,0) | trunc(45.926,-1)
-----------------+-----------------+------------------
           45.92 |              45 |               40
```

- **MOD：返回除法计算后的余数**

```
mydb=# select 1600/300 "1600/300",trunc(1600/300,0) "trunc(1600/300,0)",mod(1600,300) "mod(1600,300)";
     1600/300      | trunc(1600/300,0) | mod(1600,300)
-------------------+-------------------+---------------
 5.33333333333333 |                 5 |           100
```

- **其他常用数字操作函数：**

```
mydb=# select 2+3 "2+3(加法)",2-3 "2-3(减法)",2*3 "2*3(乘法)",
      4/2 "4/2(除法)",|/ 25.0 "|/ 25.0(平方根)",||/ 27.0 "||/ 27.0(立方根)" ;
+----------+----------+----------+----------+----------------+-----------------+
| 2+3(加法) | 2-3(减法) | 2*3(乘法) | 4/2(除法) | |/ 25.0(平方根) | ||/ 27.0(立方根) |
+----------+----------+----------+----------+----------------+-----------------+
|        5 |       -1 |        6 |        2 |              5 |               3 |
+----------+----------+----------+----------+----------------+-----------------+
mydb=# select 5%4 "5%4(求余)",@ -5.0 "@ -5.0(绝对值)",5! as "5!(阶乘)",!!5 as "!!5(前缀阶乘)",
       2.0^3.0 "2.0^3.0(指数运算)",power(2.0,3.0) "power(2.0,3.0)(指数运算)",random() "random()(随机小数)";
+----------+----------------+---------+--------------+--------------------+--------------------------+--------------------+
| 5%4(求余) | @ -5.0(绝对值) | 5!(阶乘) | !!5(前缀阶乘) | 2.0^3.0(指数运算)  | power(2.0,3.0)(指数运算) | random()(随机小数) |
+----------+----------------+---------+--------------+--------------------+--------------------------+--------------------+
|        1 |            5.0 |     120 |          120 | 8.0000000000000000 |       8.0000000000000000 |   .366817091591656 |
+----------+----------------+---------+--------------+--------------------+--------------------------+--------------------+
```

openGauss

单行函数(日期/时间函数)示例1

- **查询当前时间**

```
mydb=# select sysdate,current_date "current_date",current_timestamp "current_timestamp",pg_systimestamp() "pg_systimestamp()";
+---------------------+--------------+-------------------------------+-------------------------------+
|       sysdate       | current_date |       current_timestamp       |       pg_systimestamp()       |
+---------------------+--------------+-------------------------------+-------------------------------+
| 2020-12-26 15:22:08 | 2020-12-26   | 2020-12-26 15:22:07.877317+08 | 2020-12-26 15:22:07.877317+08 |
+---------------------+--------------+-------------------------------+-------------------------------+

mydb=# select timenow() "timenow()",timeofday() "timeofday()",clock_timestamp() "clock_timestamp()",statement_timestamp() "statement_timestamp()";
+------------------------+-----------------------------------+-------------------------------+-------------------------------+
|       timenow()        |            timeofday()            |       clock_timestamp()       |     statement_timestamp()     |
+------------------------+-----------------------------------+-------------------------------+-------------------------------+
| 2020-12-26 15:22:09+08 | Sat Dec 26 15:22:09.793213 2020 CST | 2020-12-26 15:22:09.793221+08 | 2020-12-26 15:22:09.793174+08 |
+------------------------+-----------------------------------+-------------------------------+-------------------------------+
```

- **显示当前事务开始时间**

```
mydb=# select now() "now()",current_time "current_time",transaction_timestamp() "transaction_timestamp()";
+-------------------------------+--------------------+-------------------------------+
|             now()             |    current_time    |   transaction_timestamp()     |
+-------------------------------+--------------------+-------------------------------+
| 2020-12-26 15:24:13.29664+08 | 15:24:13.29664+08 | 2020-12-26 15:24:13.29664+08 |
+-------------------------------+--------------------+-------------------------------+
```

- **时间截断**

```
mydb=# select date_trunc('year',   timestamp '2001-02-16 20:38:40') "trunc_year",
mydb-#        date_trunc('month',  timestamp '2001-02-16 20:38:40') "trunc_month",
mydb-#        date_trunc('day',    timestamp '2001-02-16 20:38:40') "trunc_day";
+---------------------+---------------------+---------------------+
|     trunc_year      |     trunc_month     |      trunc_day      |
+---------------------+---------------------+---------------------+
| 2001-01-01 00:00:00 | 2001-02-01 00:00:00 | 2001-02-16 00:00:00 |
+---------------------+---------------------+---------------------+
```

```
mydb=# select date_trunc('hour',   timestamp '2001-02-16 20:38:40') "trunc_hour",
mydb-#        date_trunc('minute', timestamp '2001-02-16 20:38:40') "trunc_minute",
mydb-#        date_trunc('second', timestamp '2001-02-16 20:38:40') "trunc_second";
+---------------------+---------------------+---------------------+
|     trunc_hour      |     trunc_minute    |     trunc_second    |
+---------------------+---------------------+---------------------+
| 2001-02-16 20:00:00 | 2001-02-16 20:38:00 | 2001-02-16 20:38:40 |
+---------------------+---------------------+---------------------+
```

openGauss

单行函数(日期/时间函数)示例2

- **时间截取**

```
mydb=# select extract(year  from timestamp '2020-11-13 21:38:40') "year",
mydb-#        extract(month from timestamp '2020-11-13 21:38:40') "month",
mydb-#        extract(day   from timestamp '2020-11-13 21:38:40') "day",
mydb-#        extract(hour  from timestamp '2020-11-13 21:38:40') "hour",
mydb-#        extract(min   from timestamp '2020-11-13 21:38:40') "minutes",
mydb-#        extract(sec   from timestamp '2020-11-13 21:38:40') "seconds";
+------+-------+-----+------+---------+---------+
| year | month | day | hour | minutes | seconds |
+------+-------+-----+------+---------+---------+
| 2020 |    11 |  13 |   21 |      38 |      40 |
+------+-------+-----+------+---------+---------+
```

```
mydb=# select date_part('year',   timestamp '2020-11-13 21:38:40') "year",
mydb-#        date_part('month',  timestamp '2020-11-13 21:38:40') "month",
mydb-#        date_part('day',    timestamp '2020-11-13 21:38:40') "day",
mydb-#        date_part('hour',   timestamp '2020-11-13 21:38:40') "hour",
mydb-#        date_part('minute', timestamp '2020-11-13 21:38:40') "minutes",
mydb-#        date_part('second', timestamp '2020-11-13 21:38:40') "seconds";
+------+-------+-----+------+---------+---------+
| year | month | day | hour | minutes | seconds |
+------+-------+-----+------+---------+---------+
| 2020 |    11 |  13 |   21 |      38 |      40 |
+------+-------+-----+------+---------+---------+
```

- **时间计算**

```
mydb=# select age('2000-01-01 00:00:00'),age('2010-01-01 00:00:00','2000-10-01 00:00:00');
+--------------------------+----------------+
|           age            |      age       |
+--------------------------+----------------+
| 20 years 11 mons 25 days | 9 years 3 mons |
+--------------------------+----------------+

mydb=# select add_months(to_date('2000-06-01', 'yyyy-mm-dd'), 5) "months+5",add_months(to_date('2000-06-01', 'yyyy-mm-dd'), -5) "months-5";
+---------------------+---------------------+
|      months+5       |      months-5       |
+---------------------+---------------------+
| 2000-11-01 00:00:00 | 2000-01-01 00:00:00 |
+---------------------+---------------------+

mydb=# select sysdate,next_day(sysdate,'Tue') "Next Tue",last_day(sysdate);
+---------------------+---------------------+---------------------+
|       sysdate       |      Next Tue       |      last_day       |
+---------------------+---------------------+---------------------+
| 2020-12-26 15:57:10 | 2020-12-29 15:57:10 | 2020-12-31 15:57:10 |
+---------------------+---------------------+---------------------+
```

openGauss

日期计算相关示例

```
mydb=# select sysdate,sysdate+1 "sysdate+1",sysdate+1/24 "sysdate+1/24",sysdate+1/24/60 "sysdate+1/24/60";
+---------------------+---------------------+---------------------+---------------------+
|       sysdate       |      sysdate+1      |    sysdate+1/24     |   sysdate+1/24/60   |
+---------------------+---------------------+---------------------+---------------------+
| 2020-12-26 15:55:24 | 2020-12-27 15:55:24 | 2020-12-26 16:55:24 | 2020-12-26 15:56:24 |
+---------------------+---------------------+---------------------+---------------------+
mydb=# select timestamp '2000-01-01 10:00:00' + '7'          "+7 days",
mydb-#        timestamp '2000-01-01 10:00:00' + time '03:00' "+3:00",
mydb-#        timestamp '2000-01-01 10:00:00' + interval '6 day 5 hour 30min 10sec' "+6 days 5:30:10";
+---------------------+---------------------+---------------------+
|       +7 days       |        +3:00        |   +6 days 5:30:10   |
+---------------------+---------------------+---------------------+
| 2000-01-08 10:00:00 | 2000-01-01 13:00:00 | 2000-01-07 15:30:10 |
+---------------------+---------------------+---------------------+

mydb=# select interval '1 day' + interval '1 hour' "1day+1hour",
mydb-#        date '2001-10-01' - date '2001-09-28' "2001-10-01 - 2001-09-28",
mydb-#        time '05:00' - time '03:00' "05:00-03:00",
mydb-#        timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' "2001-09-29 03:00 - 2001-09-27 12:00";
+----------------+-------------------------+-------------+-------------------------------------+
|   1day+1hour   | 2001-10-01 - 2001-09-28 | 05:00-03:00 | 2001-09-29 03:00 - 2001-09-27 12:00 |
+----------------+-------------------------+-------------+-------------------------------------+
| 1 day 01:00:00 | 3 days                  | 02:00:00    | 1 day 15:00:00                      |
+----------------+-------------------------+-------------+-------------------------------------+

mydb=# select 900 * interval '1 second' "900*1s", interval '1 hour' /2 "1hour/2";
+----------+----------+
|  900*1s  | 1hour/2  |
+----------+----------+
| 00:15:00 | 00:30:00 |
+----------+----------+
```

openGauss

单行函数(数据类型转换)->隐式转换

数据库中允许有些数据类型进行隐式类型转换（赋值、函数调用的参数等），有些数据类型间不允许进行隐式数据类型转换，可尝试使用openGauss提供的类型转换函数，例如CAST进行数据类型强转。

openGauss数据库常见的隐式类型转换，见表：

原始数据类型	目标数据类型	备注	原始数据类型	目标数据类型	备注
CHAR	VARCHAR2	-	DATE	CHAR	-
CHAR	NUMBER	原数据必须由数字组成。	DATE	VARCHAR2	-
CHAR	DATE	原数据不能超出合法日期范围。	RAW	CHAR	-
CHAR	RAW	-	RAW	VARCHAR2	-
CHAR	CLOB	-	CLOB	CHAR	-
VARCHAR2	CHAR	-	CLOB	VARCHAR2	-
VARCHAR2	NUMBER	原数据必须由数字组成。	CLOB	NUMBER	原数据必须由数字组成。
VARCHAR2	DATE	原数据不能超出合法日期范围。	INT4	CHAR	-
VARCHAR2	CLOB	-	INT4	BOOLEAN	-
NUMBER	CHAR	-	INT4	CHAR	-
NUMBER	VARCHAR2	-	BOOLEAN	INT4	-

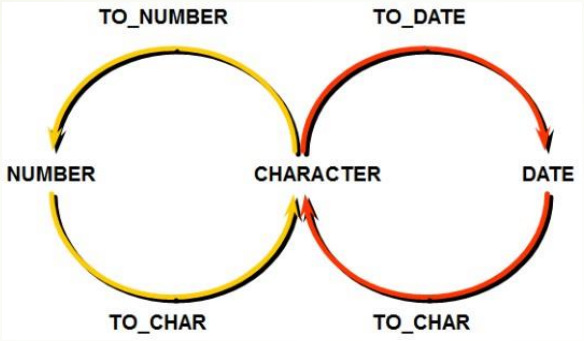
➢ **示例**

```
mydb=# select * from emp where empno='7788';
+-------+--------+--------+------+---------------------+---------+------+--------+
| empno | ename  |  job   | mgr  |      hiredate       |   sal   | comm | deptno |
+-------+--------+--------+------+---------------------+---------+------+--------+
|  7788 | 瘦羊羊 | 架构师 | 7566 | 2010-09-09 00:00:00 | 3000.00 |      |     20 |
+-------+--------+--------+------+---------------------+---------+------+--------+
```

隐式转换char-->number

单行函数(数据类型转换)->显示转换

- 常见的数据类型转换场景



- cast(x as y)

描述：类型转换函数，将x转换成y指定的类型。

```
mydb=# select cast('22-oct-1997' as timestamp);
+---------------------+
|      timestamp      |
+---------------------+
| 1997-10-22 00:00:00 |
+---------------------+

mydb=# select cast(44 as bit(8)) AS RESULT;
+----------+
|  result  |
+----------+
| 00101100 |
+----------+

mydb=# select '101100'::bit(6)::integer AS RESULT;
+--------+
| result |
+--------+
|     44 |
+--------+
```

- to_number(text, text)

描述：将字符串类型的值转换为指定格式的数字。

模式	描述
9	带有指定数值位数的值
0	带前导零的值
. (句点)	小数点
, (逗号)	分组（千）分隔符
PR	尖括号内负值
S	带符号的数值（使用区域设置）
L	货币符号（使用区域设置）

数值格式化模板

模式	描述
D	小数点（使用区域设置）
G	分组分隔符（使用区域设置）
MI	在指明的位置的负号（如果数字 < 0）
PL	在指明的位置的正号（如果数字 > 0）
SG	在指明的位置的正/负号
RN	罗马数字（输入在 1 和 3999 之间）
TH或th	序数后缀
V	移动指定位（小数）

```
mydb=# select to_number('12,454.8-','99G999D9S'),to_number('12,454.8' ,'99G999D9S');
+-----------+-----------+
| to_number | to_number |
+-----------+-----------+
|  -12454.8 |   12454.8 |
+-----------+-----------+
```

单行函数(数据类型转换)->char类型转换

➢ **数值格式化模板(to_char)**

模式	描述
9	带有指定数值位数的值
0	带前导零的值
.(句点)	小数点
,(逗号)	分组（千）分隔符
PR	尖括号内负值
S	带符号的数值（使用区域设置）
L	货币符号（使用区域设置）

模式	描述
D	小数点（使用区域设置）
G	分组分隔符（使用区域设置）
MI	在指明的位置的负号（如果数字 < 0）
PL	在指明的位置的正号（如果数字 > 0）
SG	在指明的位置的正/负号
RN	罗马数字（输入在 1 和 3999 之间）
TH或th	序数后缀
V	移动指定位（小数）

◆ **to_char (string)**

描述：将CHAR、VARCHAR、VARCHAR2、CLOB类型转换为VARCHAR类型。如果对CLOB类型进行转换，且待转换的CLOB值超出目标类型的范围(varchar存储空间为10MB)，则返回错误。

➢ **示例：**

```
mydb=# select to_char('01110');
+---------+
| to_char |
+---------+
| 01110   |
+---------+
```

◆ **to_char (numeric/smallint/integer/bigint/double precision/real[, fmt])**

描述：将一个整型/浮点类型/数字类型的值转换为指定格式的字符串。
模板可以有类似FM的修饰词抑制前导的零或尾随的空白，但FM不抑制由模板0指定而输出的0。
要将整型类型的值转换成对应16进制值的字符串，使用模板X或x。

示例：

```
mydb=# select to_char(1148.5,'9,999.999')  as "(1148.5,'9,999.999')",
mydb-#        to_char(1148.5,'909999.909') as "(1148.5,'909999.909')",
mydb-#        to_char(-1148.5,'9999D99S')  as "(-1148.5, '999D99S')",
mydb-#        to_char(1148.5,'XXX')        as "(1148.5,'XXX')";
+---------------------+---------------------+---------------------+---------------+
| (1148.5,'9,999.999') | (1148.5,'909999.909') | (-1148.5, '999D99S') | (1148.5,'XXX') |
+---------------------+---------------------+---------------------+---------------+
|  1,148.500          |   01148.500          | 1148.50-            |  47D          |
+---------------------+---------------------+---------------------+---------------+
```

```
mydb=# select to_char(sysdate,'Day,"the" ddth "of" Month,yyyy'),
mydb-#        to_char(sysdate,'FMDay,"the" ddth "of" FMMonth,yyyy'),
mydb-#        to_char(interval '15h 2m 12s', 'HH24:MI:SS');
+-------------------------------------+-------------------------------------+----------+
|              to_char                |              to_char                | to_char  |
+-------------------------------------+-------------------------------------+----------+
| Saturday ,the 26th of December ,2020 | Saturday,the 26th of December,2020 | 15:02:12 |
+-------------------------------------+-------------------------------------+----------+
```

openGauss

单行函数(数据类型转换)->日期类型转换

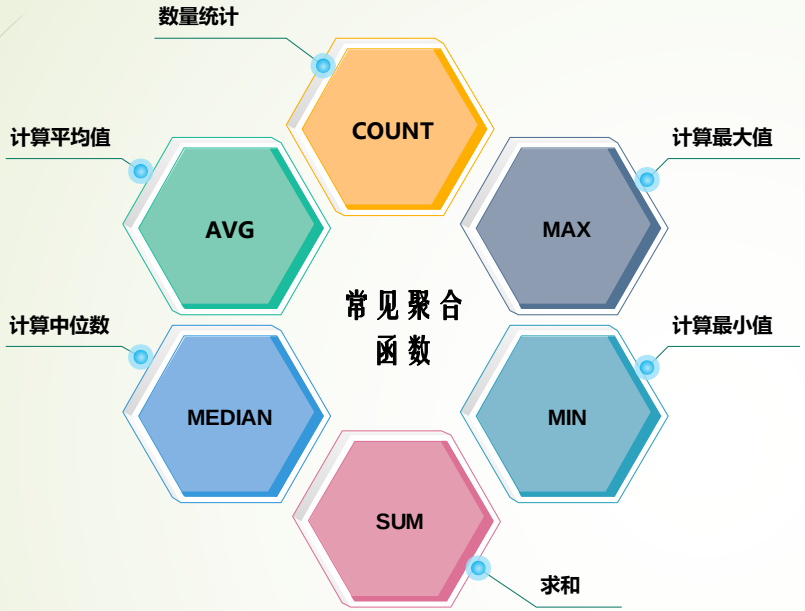
类别	模式	描述
小时	HH	一天的小时数（01-12）
	HH12	一天的小时数（01-12）
	HH24	一天的小时数（00-23）
分钟	MI	分钟（00-59）
秒	SS	秒（00-59）
	FF	微秒（000000-999999）
	SSSSS	午夜后的秒（0-86399）
上、下午	AM或A.M.	上午标识
	PM或P.M.	下午标识
年	Y,YYY	带逗号的年（4和更多位）
	SYYYY	公元前四位年
	YYYY	年（4和更多位）
	YYY	年的后三位
	YY	年的后两位
	Y	年的最后一位
	IYYY	ISO年（4位或更多位）
	IYY	ISO年的最后三位
	IY	ISO年的最后两位
	I	ISO年的最后一位
	RR	年的后两位（可在21世纪存储20世纪的年份）
	RRRR	可接收4位年或两位年。若是两位，则和RR的返回值相同，若是四位，则和YYYY相同。
	• BC或B.C. • AD或A.D.	纪元标识。BC（公元前），AD（公元后）。
月	MONTH	全长大写月份名（空白填充为9字符）
	MON	大写缩写月份名（3字符）
	MM	月份数（01-12）
	RM	罗马数字的月份（I-XII；I=JAN）（大写）
天	DAY	全长大写日期名（空白填充为9字符）
	DY	缩写大写日期名（3字符）
	DDD	一年里的日（001-366）
	DD	一个月里的日（01-31）
	D	一周里的日（1-7；周日是 1）
周	W	一个月里的周数（1-5）（第一周从该月第一天开始）
	WW	一年里的周数（1-53）（第一周从该年的第一天开始）
	IW	ISO一年里的周数（第一个星期四在第一周里）
世纪	CC	世纪（2位）（21 世纪从 2001-01-01 开始）
儒略日	J	儒略日（自公元前 4712 年 1 月 1 日来的天数）
季度	Q	季度

- **示例：**

```
mydb=# select to_timestamp('2020-11-13 12:00:00','YYYY-MM-DD HH24:MI:SS'),
mydb-#        to_timestamp('11/13/2020 12/00/00','MM/DD/YYYY HH24/MI/SS'),
mydb-#        to_date('11/13/2020','MM/DD/YYYY'),
mydb-#        cast ('28-dec-2020' as timestamp);
+---------------------+---------------------+---------------------+---------------------+
|    to_timestamp     |    to_timestamp     |       to_date       |      timestamp      |
+---------------------+---------------------+---------------------+---------------------+
| 2020-11-13 12:00:00 | 2020-11-13 12:00:00 | 2020-11-13 00:00:00 | 2020-12-28 00:00:00 |
+---------------------+---------------------+---------------------+---------------------+
```

openGauss

聚集函数



ERROR:
aggregate function calls cannot be nested!

➢ **计算总和**

•sum(expression)

返回类型:

对于SMALLINT或INT输入，输出类型为BIGINT。

对于BIGINT输入，输出类型为NUMBER 。

对于浮点数输入，输出类型为DOUBLE PRECISION。

否则和输入数据类型相同。

```
mydb=# select sum(sal) from emp where deptno=10;
+---------+
|   sum   |
+---------+
| 8750.00 |
+---------+
```

➢ **计算平均值**

•avg(expression)

返回类型:

对于任何整数类型输入，结果都是NUMBER类型。

对于任何浮点输入，结果都是DOUBLE PRECISION类型。

否则和输入数据类型相同。

```
mydb=# select avg(sal) from emp where deptno=10;
+-----------------------+
|          avg          |
+-----------------------+
| 2916.6666666666666667 |
+-----------------------+
```

openGauss

聚集函数示例

➢ **计算最大值和最小值**
- **max(expression)**
- **min(expression)**

参数类型：任意数组、数值、字符串、日期/时间类型
返回类型：与参数数据类型相同

```
mydb=# select max(sal),min(sal) from emp;
+---------+--------+
|   max   |  min   |
+---------+--------+
| 5000.00 | 800.00 |
+---------+--------+
```

➢ **统计行数**
- **count(*)** --> 返回表中的记录行数,包含空行
- **count(1)** --> 返回表中的记录行数,包含空行
- **count(expression)** --> 返回表中满足expression不为NULL的行数

```
mydb=# select count(*),count(1),count(comm) from emp;
+-------+-------+-------+
| count | count | count |
+-------+-------+-------+
|    14 |    14 |     4 |
+-------+-------+-------+
```

➢ **计算中位数并输出字符串集合**
- **median(expression)** --> 返回表达式的中位数，计算时NULL将会被median函数忽略
- **string_agg(expression, delimiter)** --> 将输入值连接成为一个字符串，用分隔符分开。

```
mydb=# select median(empno),string_agg(empno,',') from emp;
+--------+-------------------------------------------------------------------------+
| median |                               string_agg                                |
+--------+-------------------------------------------------------------------------+
|   7785 | 7369,7499,7566,7654,7698,7782,7788,7839,7844,7876,7900,7902,7934,7521 |
+--------+-------------------------------------------------------------------------+
```

排序后的中位数

group by子句

部门号	工作	薪资
10	CFO	1300.00
10	部门经理	2450.00
10	CEO	5000.00
20	架构师	3000.00
20	部门经理	2975.00
20	工程师	800.00
20	工程师	1100.00
20	CTO	3000.00
30	销售	1250.00
30	销售	1500.00
30	销售	1600.00
30	部门经理	2850.00
30	销售	1250.00
30	工程师	950.00

部门号	薪资
10	2917
20	2175
30	1567

计算每个部门的平均工资

部门号	工作	薪资
10	CEO	5000.00
10	CFO	1300.00
10	部门经理	2450.00
20	CTO	3000.00
20	工程师	1900.00
20	架构师	3000.00
20	部门经理	2975.00
30	工程师	950.00
30	部门经理	2850.00
30	销售	5600.00

查询每个部门中每种工作的工资总和

```
mydb=# select deptno "部门号",round(avg(sal)) "平均薪资" from emp
mydb-# group by deptno
mydb-# order by deptno;
+--------+----------+
| 部门号 | 平均薪资 |
+--------+----------+
|     10 |     2917 |
|     20 |     2175 |
|     30 |     1567 |
+--------+----------+
```

```
mydb=# select deptno "部门号",job "工作",sum(sal) "薪资总和" from emp
mydb-# group by deptno,job
mydb-# order by deptno;
+--------+----------+----------+
| 部门号 |   工作   | 薪资总和 |
+--------+----------+----------+
|     10 | CEO      |  5000.00 |
|     10 | CFO      |  1300.00 |
|     10 | 部门经理 |  2450.00 |
|     20 | CTO      |  3000.00 |
|     20 | 部门经理 |  2975.00 |
|     20 | 工程师   |  1900.00 |
|     20 | 架构师   |  3000.00 |
|     30 | 部门经理 |  2850.00 |
|     30 | 工程师   |   950.00 |
|     30 | 销售     |  5600.00 |
+--------+----------+----------+
```

Tips: 没有在聚集函数中出现的列必须在group by子句中出现

Tips: where、having与group by子句顺序有要求

```
SELECT  [column,]group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column]
[HAVING     condition]
;
```

- **查询平均薪资大于2000的部门**

```
mydb=# select deptno,round(avg(sal)) from emp
mydb-# group by deptno
mydb-# having avg(sal)>2000;
+--------+-------+
| deptno | round |
+--------+-------+
|     10 |  2917 |
|     20 |  2175 |
+--------+-------+
```

- **查询每个部门的平均工资(排除薪资小于2000的员工)**

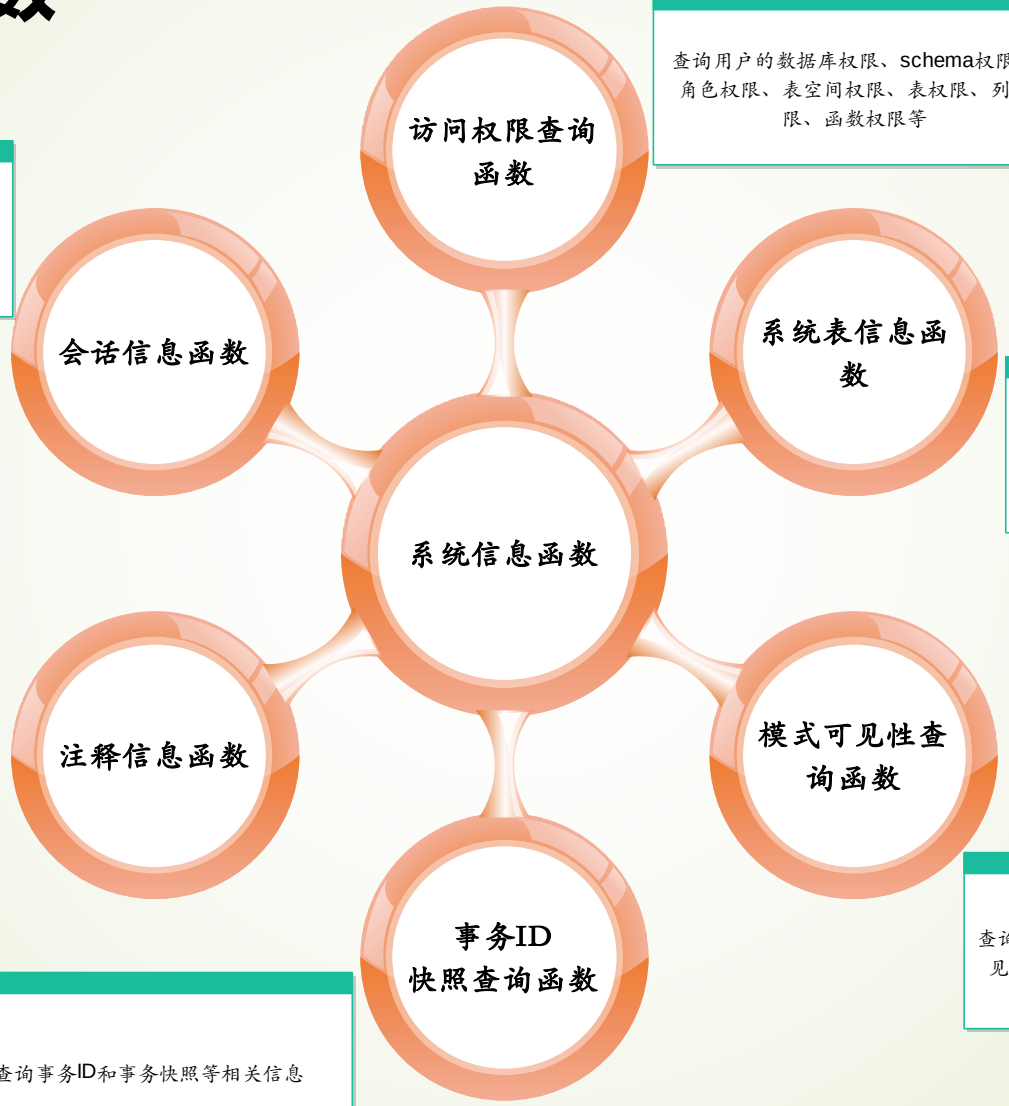
```
mydb=# select deptno,round(avg(sal)) from emp
mydb-# where sal>2000
mydb-# group by deptno;
+--------+-------+
| deptno | round |
+--------+-------+
|     10 |  3725 |
|     30 |  2850 |
|     20 |  2992 |
+--------+-------+
```

openGauss

PART 4

其他系统函数

系统信息函数



查询用户的数据库权限、schema权限、角色权限、表空间权限、表权限、列权限、函数权限等

查询当前会话的数据库信息、schema信息、用户信息、连接地址及端口信息、进程信息、操作系统信息等

查询表、视图、函数、序列等数据库对象的定义信息以及其他数据库系统表信息

查询表、字段以及其他数据库对象的注释信息

查询在搜索路径中表的可见性、视图的可见性、索引的可见性、视图的可见性等

查询事务ID和事务快照等相关信息

openGauss支持的常量和宏	
参数	描述
CURRENT_CATALOG	当前数据库
CURRENT_ROLE	当前用户
CURRENT_SCHEMA	当前数据库模式
CURRENT_USER	当前用户
LOCALTIMESTAMP	当前会话时间（无时区）
NULL	空值
SESSION_USER	当前系统用户
SYSDATE	当前系统日期
USER	当前用户，此用户为CURRENT_USER的别名

openGauss

系统信息函数示例1

- 查询数据库实例启动时间和当前数据库名称(在标准SQL中称"catalog")、主机名称

```
mydb=# select pg_postmaster_start_time(),current_catalog,current_database(),get_hostname();
   pg_postmaster_start_time    | current_database | current_database |   get_hostname
-------------------------------+------------------+------------------+------------------
 2020-12-22 16:53:43.417147+08 | mydb             | mydb             | db1.opengauss.com
```

- 查询当前schema和搜索路径中的schema

```
mydb=# select current_schema(),current_schemas(true);
 current_schema |   current_schemas
----------------+---------------------
 public         | {pg_catalog,public}
```

- 查询当前环境下的用户和用户oid

```
mydb=# select user,current_user,current_role,session_user,definer_current_user(),pg_current_userid();
 current_user | current_user | current_user | session_user | definer_current_user | pg_current_userid
--------------+--------------+--------------+--------------+----------------------+-------------------
 omm          | omm          | omm          | omm          | omm                  |                10
```

- 查询当前会话ID和线程ID

```
mydb=# select pg_current_sessid(),pg_current_sessionid(),pg_backend_pid();
 pg_current_sessid |    pg_current_sessionid     | pg_backend_pid
-------------------+-----------------------------+----------------
   140643875677952 | 1608980439.140643875677952 | 140643875677952
```

时间戳.会话ID

系统信息函数示例2

➢ 系统对象定义查询函数

```
mydb=# select pg_get_functiondef((select oid from pg_proc where proname='func_add_sql'));
                        pg_get_functiondef
------------------------------------------------------------------
 (4,"CREATE OR REPLACE FUNCTION public.func_add_sql(integer, integer)+
  RETURNS integer                                                   +
  LANGUAGE sql                                                      +
  IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE                         +
 AS $function$select $1 + $2;$function$                             +
 ")

mydb=# select pg_get_tabledef((select oid from pg_class where relname='emp'));
                           pg_get_tabledef
-------------------------------------------------------------------
 SET search_path = public;                                        +
 CREATE  TABLE emp (                                              +
         empno numeric(4,0) NOT NULL,                             +
         ename character varying(10),                             +
         job character varying(20),                               +
         mgr numeric(4,0),                                        +
         hiredate timestamp(0) without time zone,                 +
         sal numeric(7,2),                                        +
         comm numeric(7,2),                                       +
         deptno numeric(2,0),                                     +
     CONSTRAINT emp_fk_emp FOREIGN KEY (mgr) REFERENCES emp(empno),     +
     CONSTRAINT emp_fk_dept FOREIGN KEY (deptno) REFERENCES dept(deptno)+
 )                                                                +
 WITH (orientation=row, compression=no)                           +
 TABLESPACE tbs1;                                                 +
 ALTER TABLE emp ADD CONSTRAINT emp_pk PRIMARY KEY (empno);
```

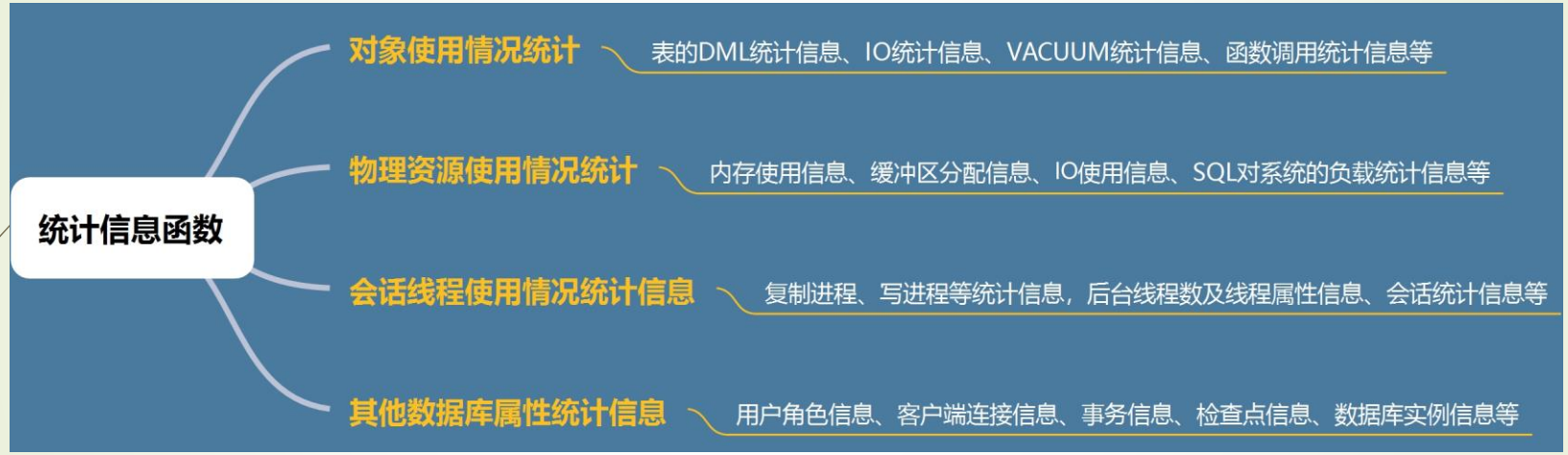
➢ 访问权限查询函数

```
mydb=# select has_database_privilege('jack','mydb','create');
 has_database_privilege
------------------------
 t

mydb=# select has_any_column_privilege('jack','emp','select');
 has_any_column_privilege
--------------------------
 t

mydb=# select has_table_privilege('jack','emp','update,select');
 has_table_privilege
---------------------
 t

mydb=# select has_sequence_privilege('jack','seq1','update');
 has_sequence_privilege
------------------------
 t

mydb=# select has_schema_privilege('alice','jack','usage');
 has_schema_privilege
----------------------
 f
```

统计信息函数

统计信息函数根据访问对象分为两种类型：

- 针对某个数据库进行访问的函数，以数据库中每个表或索引的OID作为参数，标识需要报告的数据库；
- 针对某个服务器进行访问的函数，以一个服务器线程号为参数，其范围从1到当前活跃服务器的数目。



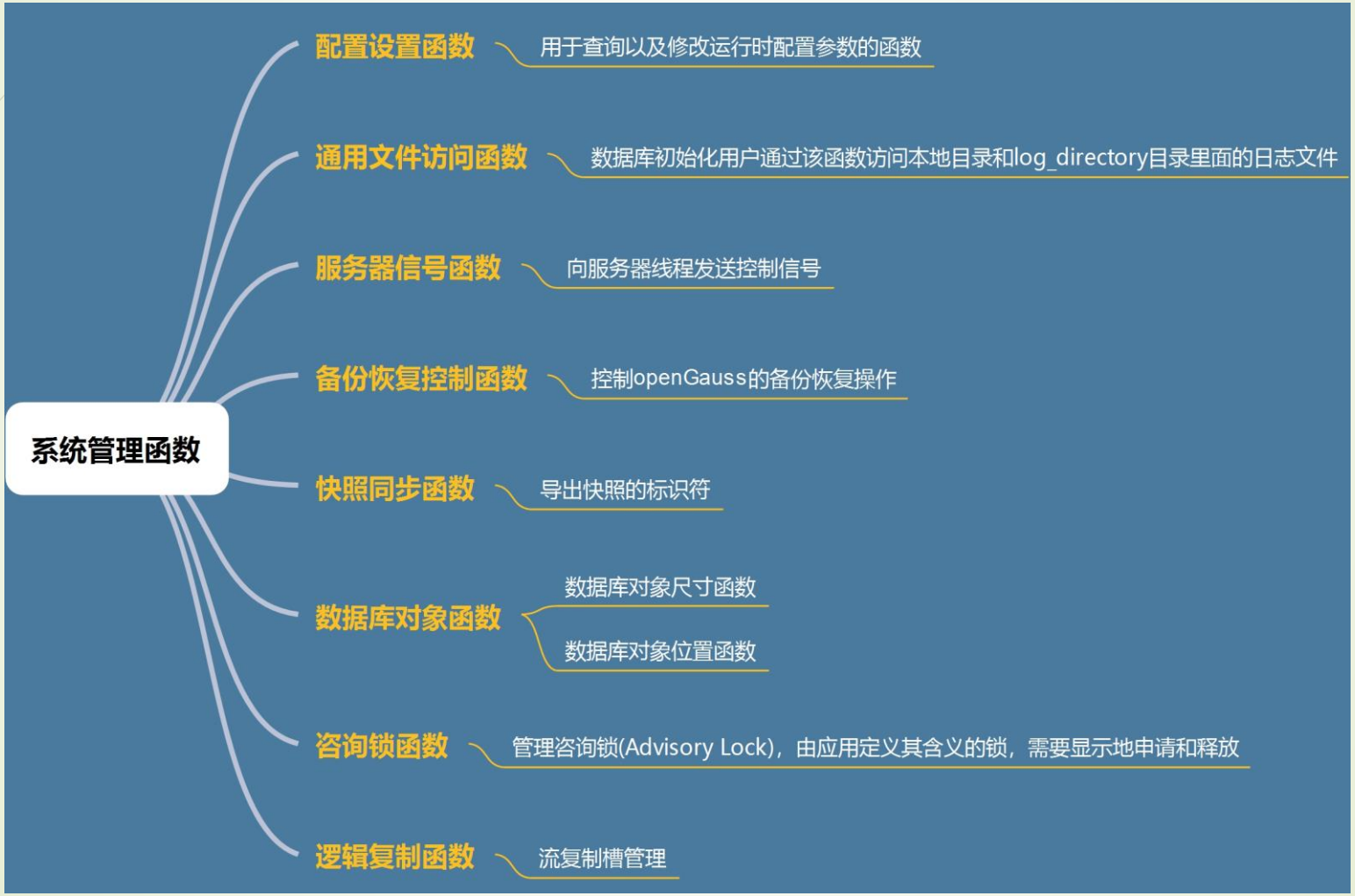
- **示例：**

```
mydb=# select pg_stat_get_tuples_inserted(16514) as inserted,
mydb-#        pg_stat_get_tuples_updated(16514)  as updated,
mydb-#        pg_stat_get_tuples_deleted(16514)  as deleted,
mydb-#        pg_stat_get_tuples_changed(16514)  as changed,
mydb-#        pg_stat_get_live_tuples(16514)     as live,
mydb-#        pg_stat_get_dead_tuples(16514)     as dead;
 inserted | updated | deleted | changed | live | dead
----------+---------+---------+---------+------+------
       14 |       2 |       2 |       0 |   12 |    4
```

```
mydb=# select * from pv_total_memory_detail() limit 8;
 nodename |      memorytype      | memorymbytes
----------+----------------------+--------------
 db1      | max_process_memory   |        12288
 db1      | process_used_memory  |          372
 db1      | max_dynamic_memory   |        11454
 db1      | dynamic_used_memory  |          258
 db1      | dynamic_peak_memory  |          265
 db1      | dynamic_used_shrctx  |           20
 db1      | dynamic_peak_shrctx  |           20
 db1      | max_shared_memory    |          321
```

Tips: pg_stat_get_tuples_changed → 2次analyze之间做的DML操作

系统管理函数


系统管理函数
配置设置函数
用于查询以及修改运行时配置参数的函数
通用文件访问函数
数据库初始化用户通过该函数访问本地目录和log_directory目录里面的日志文件
服务器信号函数
向服务器线程发送控制信号
备份恢复控制函数
控制openGauss的备份恢复操作
快照同步函数
导出快照的标识符
数据库对象函数
数据库对象尺寸函数
数据库对象位置函数
咨询锁函数
管理咨询锁(Advisory Lock)，由应用定义其含义的锁，需要显示地申请和释放
逻辑复制函数
流复制槽管理

openGauss

系统管理函数示例

- 读取外部二进制文件

```
mydb=# select convert_from(pg_read_binary_file('postmaster.pid'),'UTF-8');
    convert_from
--------------------
 12424              +
 /gauss/data/db1    +
 1608627223         +
 26000              +
 /gauss/tmp         +
 192.168.0.225      +
  26000001   9273344+
```

- 取消指定线程的SQL查询、手动中断会话线程

```
mydb=# select pg_cancel_backend('140643926013696'),pg_terminate_backend(140643926013696);
 pg_cancel_backend | pg_terminate_backend
-------------------+----------------------
 t                 | t
```

- 切换事务日志

```
mydb=# select pg_switch_xlog();
 pg_switch_xlog
----------------
 0/3000258
(1 row)
```

返回值是被切换的事务日志结束位置+1

- 查询当前事务日志的位置

```
mydb=# select pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/4000130
(1 row)
```

0 表示LSN的高32位
4 表示对应的xlog文件的最后1位
000130 表示当前的LSN在对应的xlog中的偏移字节地址

THANKS !

谢谢观看

Gauss松鼠会
公众号

Gauss松鼠会
小助手

墨天轮
openGauss专栏

技术交流圈子推荐：Gauss松鼠会
墨天轮

汇集数据库的爱好者和关注者，大家共同学习、探索、分享数据库前沿知识和技术，交流Gauss及其他数据库的使用心得和经验，互助解决问题，共建数据库技术交流圈。