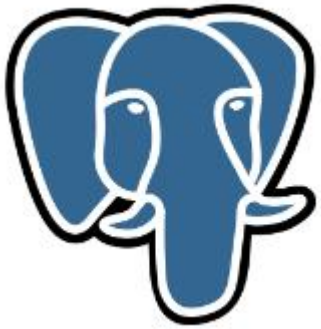


PG基础学习

CONTENTS

目录

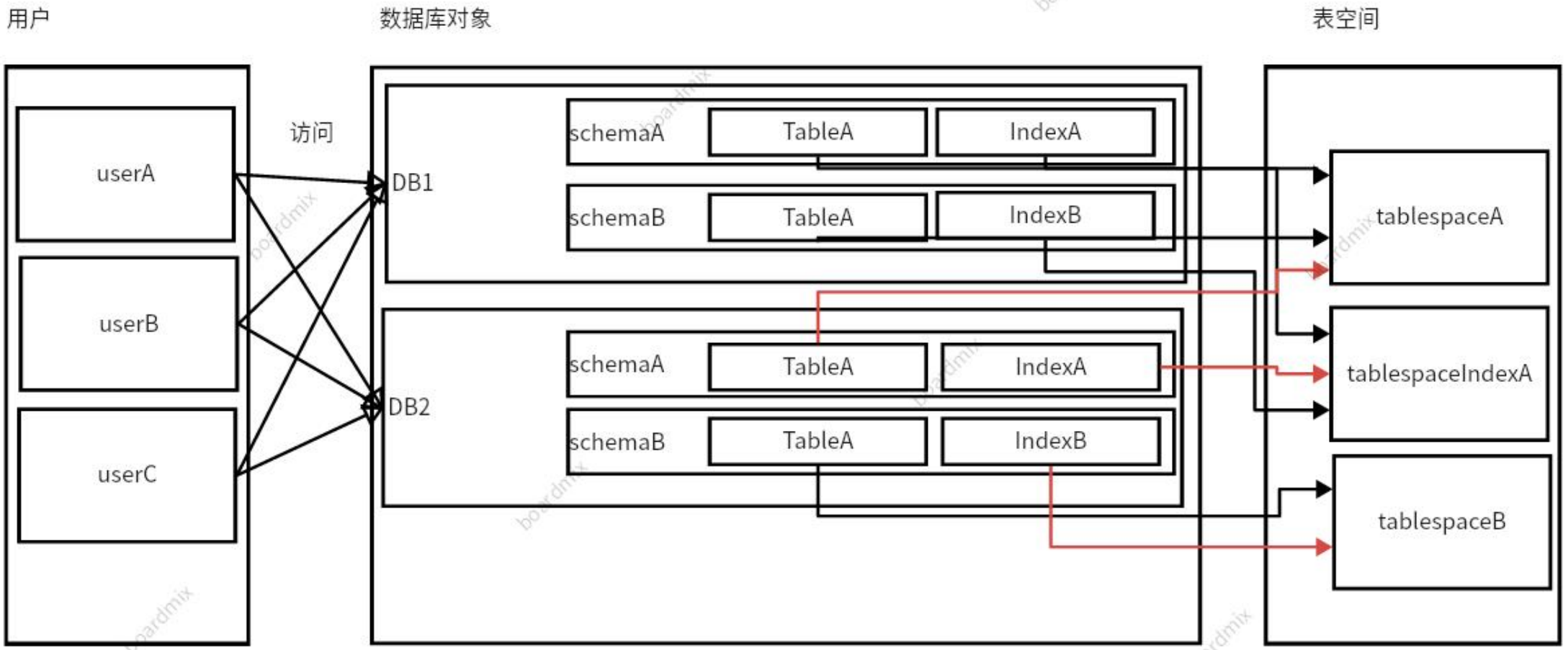
01

PART ONE

常见PG与Oracle差异

常见PG与Oracle差异

概念差异：



常见PG与Oracle差异

LOGO

PG JDBC示例：

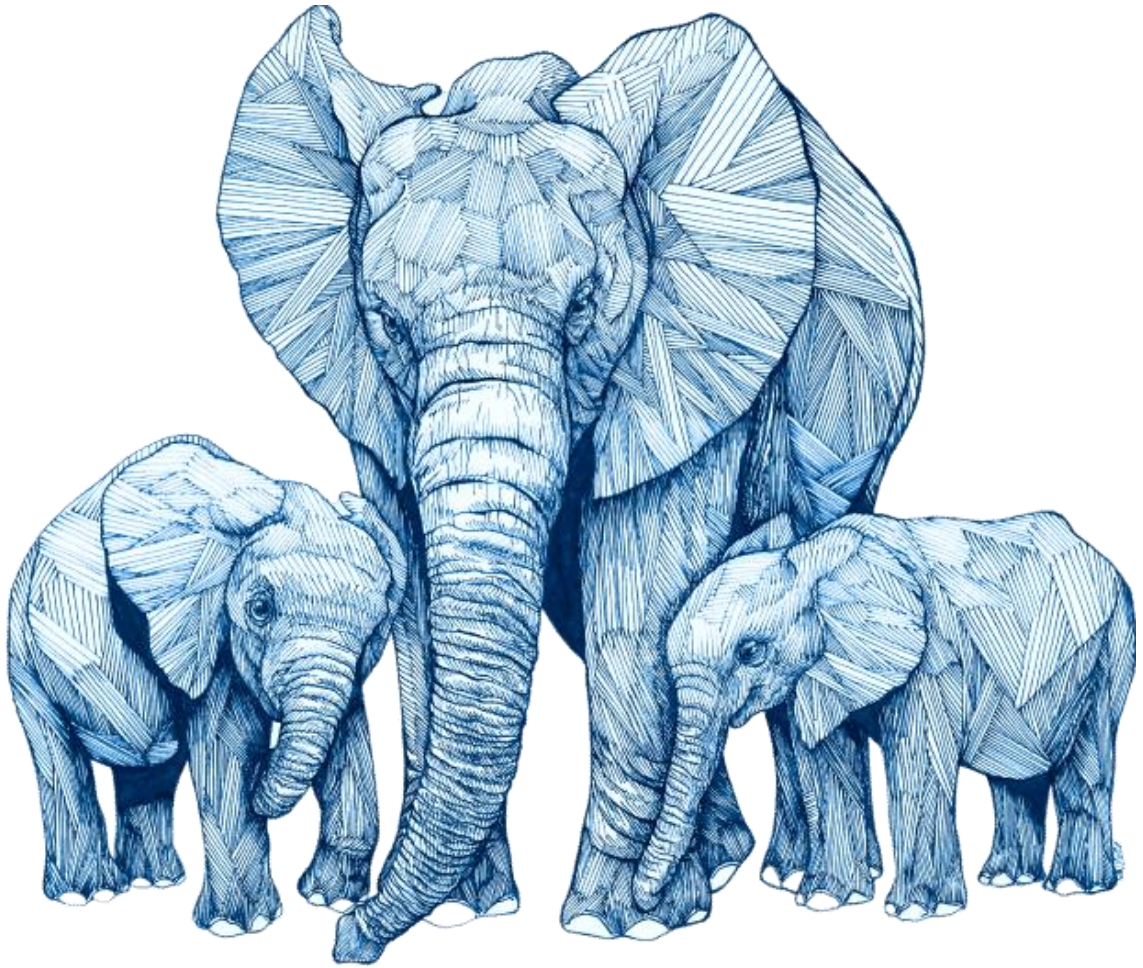
jdbc:postgresql://182.188.5.66:5432,182.188.5.67:5432,182.188.5.70:5432/devtenant1?autosave=always&targetServerType=primary&stringtype=unspecified

头部：jdbc:postgresql://
集群节点IP+端口：182.188.5.66:5432,182.188.5.67:5432,182.188.5.70:5432
数据库名：devtenant1
链接属性：

autosave：在autosave=always模式下，JDBC驱动程序在每个查询之前设置一个保存点，并在出现故障时回滚到该保存点。

targetServerType：在targetServerType=primary模式下，仅允许连接当前集群主节点。

stringtype：如果stringtype设置为unspecified，则参数将作为未类型化的值发送给服务器，服务器将尝试推断适当的类型。



常见PG与Oracle差异

语法差异：

当前时间：

ORACLE：

select sysdate from dual;

PG:

select current_timestamp;

select current_timestamp from dual;

dual语法属于ORACLE语法，但是PG通过兼容oracle插件（orafce）实现了该语法。

序列：

ORACLE：

序列使用序列名字+动作为序列取值。

select SEQNAME.NEXTVAL from dual;

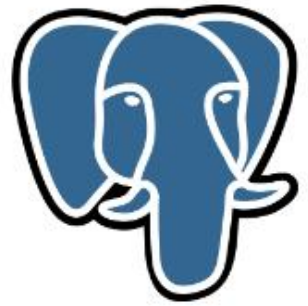
select SEQNAME.CURRVAL from dual;

PG:

select NEXTVAL('SEQNAME') from dual;

select CURRVAL('SEQNAME'）from dual;

常见PG与Oracle差异



语法差异：

同义词:
ORACLE：
select sysdate from dual;

PG:
不支持同义词

ROWNUM：
Oracle：
支持使用rownum作为查询条件限制返回行数。

Pg：
需要根据情况进行替换。
限制结果集数量，用于翻页等：
SELECT * FROM T ORDER BY C LIMIT 5 OFFSET 0

生成行号：
select T.*, ROW_NUMBER() OVER() rownum from T

注：不推荐使用第二种后面在SQL优化案例中有详细说明。

常见PG与Oracle差异

LOGO

语法差异：

分区表的主键及唯一索引：

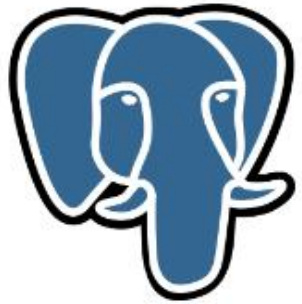
ORACLE：

支持全局不带分区键的唯一索引和主键

PG:

唯一索引和主键必须带上分区键，否则无法创建该唯一索引。如果应用业务上的唯一性不依赖分区键，则需要应用逻辑上来保证。

常见PG与Oracle差异



语法差异：

DDL：
Table:
NUMBER　替换　Numeric
大字段　　替换　Text

Oracle：
varchar2最大长度为4000

PG：
varchar 最大长度为1GB
text 最大长度为1GB

注：text不需要指定长度。

Sequence:
ORACLE:

PG:
NOCYCLE　删除　默认是不循环

02

PART TWO

PG工具演示

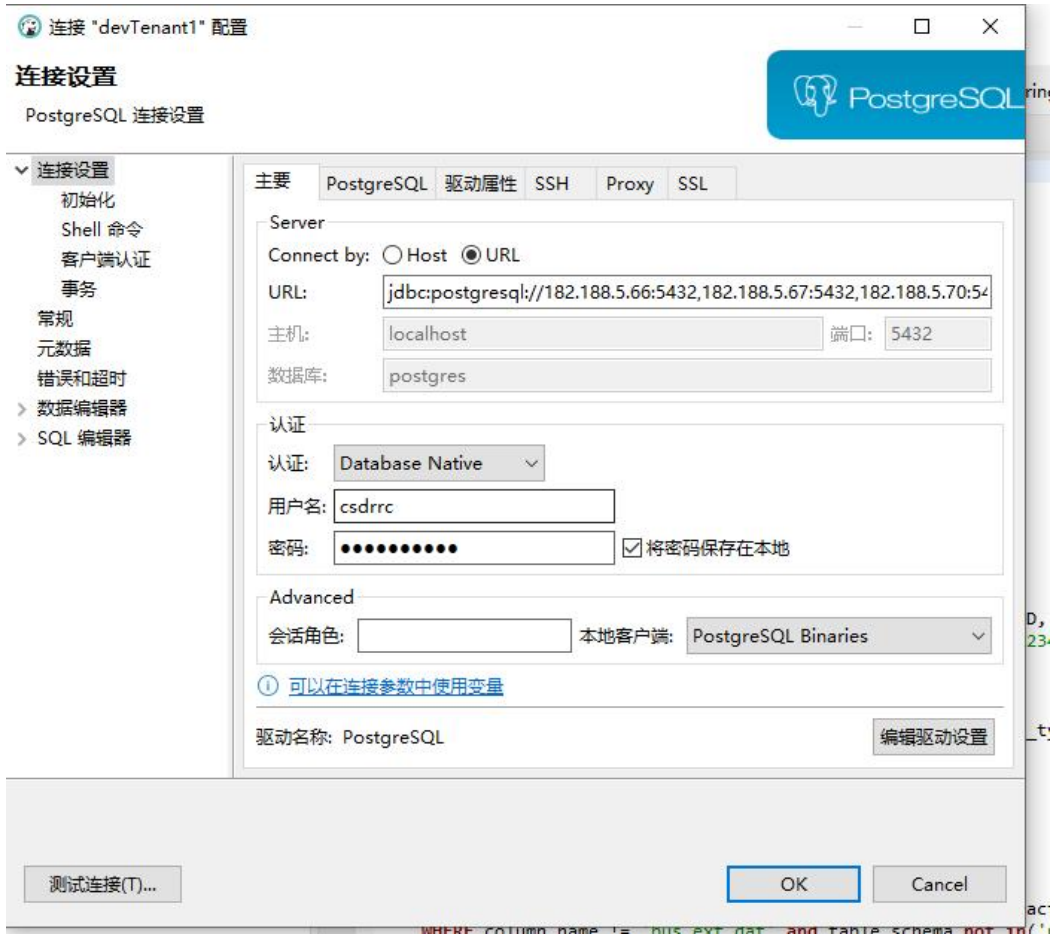
PG工具演示

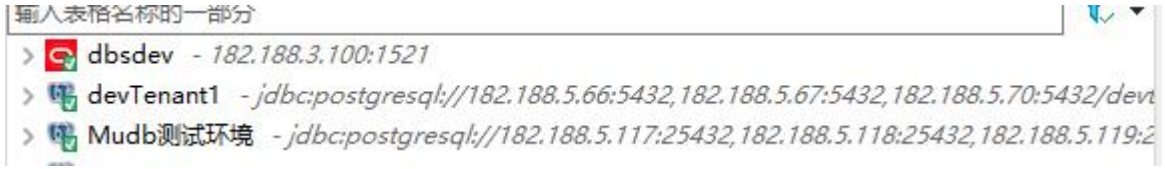
LOGO

图形化工具dbeaver：
通过JDBC访问数据库，基本可以模拟程序所遇到问题。
可视化操作。
基本满足日常需求。

命令行管理工具gsql：
PG源生
可以通过快捷命令迅速查询定义等
支持导出执行脚本等

PG工具演示-dbeaver





1.通过JDBC配置可以完全模拟程序，访问情况。因包含高可用配置，避免主节点变换而导致链接不可用。
2.可以同时管理多种数据库如：ORACLE,PG,MUDB

PG工具演示-psql

通过postgres用户登录后，使用psql登录数据库。
-h为指定IP
-d 为指定所连接数据库
-p 为指定端口
-U 为指定用户
-W 为指定密码

```
[08-15 15:28:05] postgres@pg-test-1:~
$ psql -h 182.188.5.66 -d postgres -p 5432 -U csdrrc -W
Password:
psql (14.8)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

csdrrc@pg-test-1:5432/postgres=>
```

通过\l 可以直接查看实例所包含所有数据库。

```
postgres@pg-test-1:5432/postgres=# \l
                                  List of databases
   Name    |    Owner    | Encoding | Collate |   Ctype    |    Access privileges
-----------+-------------+----------+---------+------------+-----------------------
 benchmark | postgres    | UTF8     | C       | en_US.UTF8 |
 devtenant1 | postgres    | UTF8     | C       | en_US.UTF8 | =Tc/postgres          +
           |             |          |         |            | postgres=CTc/postgres  +
           |             |          |         |            | devtenant1=CTc/postgres+
           |             |          |         |            | csdurm=CTc/postgres
 devtenant2 | postgres    | UTF8     | C       | en_US.UTF8 | =Tc/postgres          +
           |             |          |         |            | postgres=CTc/postgres  +
           |             |          |         |            | devtenant2=CTc/postgres+
           |             |          |         |            | csdurm=CTc/postgres    +
           |             |          |         |            | csdlon=C/postgres
 postgres  | postgres    | UTF8     | C       | en_US.UTF8 |
 template0 | postgres    | UTF8     | C       | en_US.UTF8 | =c/postgres           +
           |             |          |         |            | postgres=CTc/postgres
 template1 | postgres    | UTF8     | C       | en_US.UTF8 | =c/postgres           +
           |             |          |         |            | postgres=CTc/postgres
 test      | postgresmon | UTF8     | C       | en_US.UTF8 |
 test1     | postgresmon | UTF8     | C       | en_US.UTF8 |
 testoracle | postgres    | UTF8     | C       | en_US.UTF8 |
(9 rows)
```

PG工具演示-psql

通过\c+数据库名称切换到需要操作的数据库。

```
postgres@pg-test-1:5432/postgres=# \c devtenant1
You are now connected to database "devtenant1" as user "postgres".
postgres@pg-test-1:5432/devtenant1=#
```

通过\d+ +对象名可以查看该对象的定义如表类似于oracle desc命令。

通过\d 可以查看搜寻路径下所有对象如table，sequence等。

```
                List of relations
 Schema |          Name           |   Type   | Owner
--------+-------------------------+----------+----------
 csdrrc | s_cust                  | sequence | csdrrc
 csdrrc | s_inv                   | sequence | csdrrc
 csdrrc | s_lea                   | sequence | csdrrc
 csdrrc | s_mer                   | sequence | csdrrc
 csdrrc | s_mer                   | sequence | csdrrc
 csdrrc | s_mer                   | sequence | csdrrc
 csdrrc | s_mer                   | sequence | csdrrc
 csdrrc | s_me                    | sequence | csdrrc
 csdrrc | s_me                    | sequence | csdrrc
 csdrrc | s_me                    | sequence | csdrrc
```

```
csdrrc@pg-test-1:5432/devtenant1=> \d+ t_rrc_active
                                                  Table "csdrrc.t_rrc_active"
      Column      |         Type          | Collation | Nullable |       Default        | Storage  | Compression | Stats target | Description
------------------+-----------------------+-----------+----------+----------------------+----------+-------------+--------------+-------------
 usr              | character varying(13) |           | not null | ''::character varying | extended |             |              |
 lat              | character varying(8)  |           | not null | ''::character varying | extended |             |              |
 lat              | character varying(6)  |           | not null | ''::character varying | extended |             |              |
 nod              | character varying(64) |           | not null | ''::character varying | extended |             |              |
 tm_              | character varying(26) |           | not null | ''::character varying | extended |             |              |
 req              | character varying(64) |           | not null | ''::character varying | extended |             |              |
 ac_              | character varying(8)  |           |          | ''::character varying | extended |             |              |
 usr              | character varying(64) |           |          | ''::character varying | extended |             |              |
 usr              | character varying(16) |           |          | ''::character varying | extended |             |              |
 us               | character varying(1)  |           |          | ''::character varying | extended |             |              |
 co               | character varying(1)  |           |          | 'A'::character varying | extended |             |              |
 am               | character varying(1)  |           |          | 'A'::character varying | extended |             |              |
 in               | character varying(26) |           |          |                      | extended |             |              |
 ter              | character varying(35) |           | not null | ''::character varying | extended |             |              |
 tin              | character varying(20) |           | not null | ''::character varying | extended |             |              |
Indexes:
    "pk_t_rrc_active" PRIMARY KEY, btree (usr
Tablespace: "devtenant1_csdrrc_data"
Access method: heap
```

PG工具演示-psql

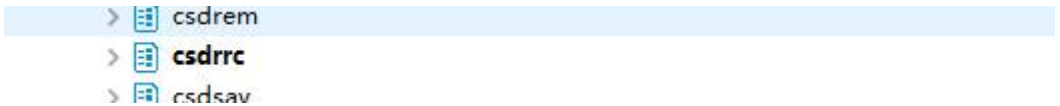
搜寻路径：
查看搜寻路径
show search_path

```
csdrrc@pg-test-1:5432/devtenant1=> show search_path;
                search_path
-------------------------------------------
 "$user", oracle, public, pg_catalog, monitor
(1 row)

Time: 0.392 ms
```

设置搜寻路径（该设置方式为会话级设置，断开链接后会还原默认值）
set search_path=schema1,schema2

注：dbeaver中也会以加粗标注



搜寻路径：
通过改变搜寻路径来达成访问不同schema下对象目的：

```
postgres@pg-test-1:5432/devtenant1=# set search_path=csdrrc;
SET
Time: 0.353 ms
postgres@pg-test-1:5432/devtenant1=# select * from t_rrc_active limit 1;
 usr
--------+----------+----------+--------+--------+--------+--------+--------+----------+----------+--------+--------------------+----------+--------
(0 rows)

Time: 17.961 ms
```

```
postgres@pg-test-1:5432/devtenant1=# set search_path=public;
SET
Time: 0.178 ms
postgres@pg-test-1:5432/devtenant1=# select * from t_rrc_active limit 1;
ERROR:  relation "t_rrc_active" does not exist
LINE 1: select * from t_rrc_active limit 1;
                      ^
Time: 0.333 ms
```

注：如在建表语句中未添加schema，会根据当前search_path创建到对应schema。

PG工具演示-psql

通过\du 查看用户及所属角色。

List of roles

Role name	Attributes	Member of
asdbas		{}
asdpas		{}
benchmark	Superuser	{}
bsdbui		{}
bsddas		{}
bsdrcl		{}
cgdcgw		{}
cgdgtw		{}

注：在PG中角色与用户为相同。

通过\dn查看schema及其owner

List of schemas

Name	Owner
asdbas	asdbas
asdpas	asdpas
bsdbui	bsdbui
bsddas	bsddas
bsdrcl	bsdrcl
cgdcgw	cgdcgw
cgdgtw	cgdgtw
cgdogw	cgdogw
csdact	csdact
csdbap	csdbap
csdlmt	csdlmt

注：在PG中schema名字可以与用户名字不一致，如果想直接访问需要修改search_path

PG工具演示-psql

导入导出数据工具\copy：

导出语句：

```
postgres@pg-test-1:5432/test=# \copy (select * from test limit 5) to '/pg/test.csv';
COPY 5
Time: 9.936 ms
```

导出文件示例：

```
[08-15 16:34:18] postgres@pg-test-1:~
$ cat /pg/test.csv
1       小明
2       小红
3       小张
4       小刘
4433333 32alksdfa
```

导入语句：

```
postgres@pg-test-3:5432/test=# truncate table test;
TRUNCATE TABLE
Time: 745.447 ms
postgres@pg-test-3:5432/test=# \copy test from '/pg/test.csv'
COPY 5
Time: 39.496 ms
```

PART THREE

执行计划分析

执行计划分析

扫描算子：
seq scan
index scan
index only scan
bitmap scan
tid scan

控制算子：
result
append
mergeappend
modifytable
recursiveUnion

物化算子：
sort
plain-aggrate
hash-aggrate
group-aggrate
unique
windowagg
lockrows

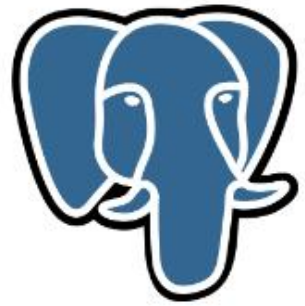
连接算子：
hash join
nest loop
merge join

执行计划分析

	QUERY PLAN
3	-> Nested Loop (cost=0.00..176.15 rows=1 width=168)
4	Join Filter: ((fezi.acc_no)::text = (acin.ac_no)::text)
5	-> Nested Loop (cost=0.00..26.30 rows=1 width=178)
6	-> Nested Loop (cost=0.00..18.02 rows=1 width=156)
7	-> Seq Scan on t_rcl_varlock fezi (cost=0.00..9.73 rows=1 width=123)
8	Filter: (((tx_typ)::text = ANY ('{72,7T}'::text[])) AND ((var_sts)::text = '2'::text))
9	-> Index Scan using pk_t_urm_pinf on t_urm_pinf pinf (cost=0.00..8.27 rows=1 width=42)
10	Index Cond: (usr_no = fezi.usr_no)
11	-> Index Scan using pk_t_sav_acbl on t_sav_acbl acbl (cost=0.00..8.27 rows=1 width=22)
12	Index Cond: ((ac_no)::text = (fezi.acc_no)::text)
13	-> Index Scan using pk_t_sav_acin on t_sav_acin acin (cost=0.00..0.44 rows=1 width=24)
14	Index Cond: ((ac_no)::text = (acbl.ac_no)::text)
15	SubPlan 1
16	-> Aggregate (cost=4.27..4.28 rows=1 width=8)
17	-> Index Scan using ni1_t_ntc_smsrec on t_ntc_smsrec (cost=0.00..4.27 rows=1 width=0)
18	Index Cond: ((snd_sts)::text = 'S'::text)
19	Filter: ((req_bus_no)::text = (fezi.req_bus_no)::text)
20	SubPlan 2
21	-> Aggregate (cost=73.33..73.34 rows=1 width=8)
22	-> Seq Scan on t_ntc_smsrec (cost=0.00..73.33 rows=1 width=0)
23	Filter: (((snd_sts)::text <> 'S'::text) AND ((req_bus_no)::text = (fezi.req_bus_no)::text))
24	SubPlan 3
25	-> Seq Scan on t_ntc_smsrec (cost=0.00..71.78 rows=1 width=131)
26	Filter: ((req_bus_no)::text = (fezi.req_bus_no)::text)

1.执行计划一般从下到上
2.根据缩进多会先执行
3.算子后括号内为优化器预估代价
cost为第一行数据需要花费...到最后一行数据需要花费时长
row为行数
width为宽度
4.explain不是真正执行该语句，其中所消费代价为预估值，来自于pg对表分析。存在一定的不准确性。

执行计划分析



如何消除不准确性？

explain 包含多个扩展属性：

ANALYZE
VERBOS
COSTS
SETTINGS
BUFFERS
WAL
TIMING
SUMMARY
FORMAT { TEXT | XML | JSON | YAML }

ANALYZE
执行命令并显示实际运行时间和其他统计信息。此参数默认为 。

VERBOSE
显示有关计划的其他信息。具体而言，包括计划树中每个节点的输出列列表、架构限定表和函数名称，始终使用范围表别名标记表达式中的变量，并始终打印显示统计信息的每个触发器的名称。如果已计算查询标识符，则还将显示查询标识符，有关详细信息。

BUFFERS
包括有关缓冲区使用情况的信息。具体而言，包括命中、读取、脏污和写入的共享块数，命中、读取、脏污和写入的本地块数，读取和写入的临时块数，以及读取和写入数据文件块和临时文件块所花费的时间（以毫秒为单位）（如果启用了track_io_timing）。

执行计划分析

```
postgres@pg-test-3:5432/test=# explain (analyze,verbose,buffers) (select * from test) union (select * from test) union (select * from test);
                                                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=3.45..3.60 rows=15 width=62) (actual time=0.024..0.025 rows=5 loops=1)
   Output: test.id, test.name
   Group Key: test.id, test.name
   Batches: 1  Memory Usage: 24kB
   Buffers: shared hit=3
   ->  Append  (cost=0.00..3.38 rows=15 width=62) (actual time=0.008..0.013 rows=15 loops=1)
         Buffers: shared hit=3
         ->  Seq Scan on public.test  (cost=0.00..1.05 rows=5 width=10) (actual time=0.007..0.008 rows=5 loops=1)
               Output: test.id, test.name
               Buffers: shared hit=1
         ->  Seq Scan on public.test test_1  (cost=0.00..1.05 rows=5 width=10) (actual time=0.001..0.001 rows=5 loops=1)
               Output: test_1.id, test_1.name
               Buffers: shared hit=1
         ->  Seq Scan on public.test test_2  (cost=0.00..1.05 rows=5 width=10) (actual time=0.001..0.001 rows=5 loops=1)
               Output: test_2.id, test_2.name
               Buffers: shared hit=1
 Query Identifier: -2239757414329093075
 Planning Time: 0.058 ms
 Execution Time: 0.058 ms
(19 rows)
```

1.算子后第二个括号中为analyze启用后所显示实际执行时间
time为第一行数据需要花费...到最后一行数据需要花费时长
　row为行数
　loops为循环次数
2.output部分为verbose启用后显示，可以看出该算子投影了哪些列。
3.Buffers部分为buffer 启用后显示，可以看出当前数据有多少条是从内存中获取多少条从磁盘读取。
注：在数据量较大情况下查询，会发现第二次查询时间要远小于第一次，这种情况就跟数据加载到内存中有关。会影响tps/qps。

04

PART FOUR

SQL优化案例

SQL优化案例

```
postgres@pg-test-3:5432/test=# explain (analyze,verbose,buffers) select * from test offset 1 limit 1;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.21..0.42 rows=1 width=10) (actual time=0.011..0.012 rows=1 loops=1)
   Output: id, name
   Buffers: shared hit=1
   ->  Seq Scan on public.test  (cost=0.00..1.05 rows=5 width=10) (actual time=0.010..0.010 rows=2 loops=1)
         Output: id, name
         Buffers: shared hit=1
 Query Identifier: -4778092353360888035
 Planning Time: 0.063 ms
 Execution Time: 0.033 ms
(9 rows)

Time: 0.479 ms
```

1.通过执行计划，可以看出如果使用窗口函数需要将全部数据排序标号后能存在行号。

2.使用limit offset好处是查询数据仅扫描了前两行。因此数据量会随翻页而逐渐上升。

```
postgres@pg-test-3:5432/test=# explain (analyze,verbose,buffers) select * from (select row_number() over() rn,* from test)testn where rn between 2 and 2;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Subquery Scan on testn  (cost=0.00..1.19 rows=1 width=18) (actual time=0.035..0.041 rows=1 loops=1)
   Output: testn.rn, testn.id, testn.name
   Filter: ((testn.rn >= 2) AND (testn.rn <= 2))
   Rows Removed by Filter: 4
   Buffers: shared hit=1
   ->  WindowAgg  (cost=0.00..1.11 rows=5 width=18) (actual time=0.031..0.037 rows=5 loops=1)
         Output: row_number() OVER (?), test.id, test.name
         Buffers: shared hit=1
         ->  Seq Scan on public.test  (cost=0.00..1.05 rows=5 width=10) (actual time=0.010..0.012 rows=5 loops=1)
               Output: test.id, test.name
               Buffers: shared hit=1
 Query Identifier: 6688428180531649918
 Planning Time: 0.086 ms
 Execution Time: 0.074 ms
(14 rows)
```

SQL优化案例-keyset pagination

表结构：

```
postgres@pg-test-3:5432/test=# \d ordtest
                    Table "oracle.ordtest"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 id     | integer               |           |          |
 name   | character varying(20) |           |          |
 endate | date                  |           |          |
```

初始SQL：

```
select *
 from ordtest
 where name='刘'
          and endate between '2008-03-01
              00:00' and '2012-03-04 12:00'
order by id
       offset 5 limit 5;
```

如何做到最优解：

1.展示几条条查询几条

2.没有循环情况

3.不再额外排序

SQL优化案例-keyset pagination

创建索引：create index keyset on ordtest(name,id,endate);
因索引默认使用btree,因此需按照以下顺序建立索引。如将endate范围查询提前将造成id索引失效，发生排序。

```
postgres@pg-test-3:5432/test=# explain (analyze,verbose,buffers) select * from ordtest where name='刘' and endate between '2008-03-01 00:00' and '2012-03-04 12:00'   order by id offset 5 limit 5;
                                                   QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.93..1.58 rows=5 width=15) (actual time=0.032..0.033 rows=5 loops=1)
   Output: id, name, endate
   Buffers: shared hit=1 read=2
   I/O Timings: read=0.013
   ->  Index Only Scan using keyset on oracle.ordtest  (cost=0.28..2.23 rows=15 width=15) (actual time=0.030..0.031 rows=10 loops=1)
         Output: id, name, endate
         Index Cond: ((ordtest.name = '刘'::text) AND (ordtest.endate >= '2008-03-01 00:00:00'::timestamp without time zone) AND (ordtest.endate <= '2012-03-04 12:00:00'::timestamp without time zone))
         Heap Fetches: 0
         Buffers: shared hit=1 read=2
         I/O Timings: read=0.013
 Query Identifier: 5896659971585719З
 Planning:
   Buffers: shared hit=33 read=1
   I/O Timings: read=0.035
 Planning Time: 0.235 ms
 Execution Time: 0.049 ms
(16 rows)

Time: 0.948 ms
```

SQL优化案例-keyset pagination

最后为了避免顺序翻页场景下offset带来的性能问题，需将上页order by最大值带入where条件代替offset，以降低数据查询总量。

```
postgres@pg-test-3:5432/test=# explain (analyze,verbose,buffers) select * from ordtest where name='刘' and endate between '2008-03-01 00:00' and '2012-03-04 12:00' and id>268   order by id limit 5;
                                                      QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.28..1.01 rows=5 width=15) (actual time=0.035..0.038 rows=5 loops=1)
   Output: id, name, endate
   Buffers: shared hit=3
   ->  Index Only Scan using keyset on oracle.ordtest  (cost=0.28..2.32 rows=14 width=15) (actual time=0.033..0.035 rows=5 loops=1)
         Output: id, name, endate
         Index Cond: ((ordtest.name = '刘'::text) AND (ordtest.id > 268) AND (ordtest.endate >= '2008-03-01 00:00:00'::timestamp without time zone) AND (ordtest.endate <= '2012-03-04 12:00:00'::timestamp without time zone))
         Heap Fetches: 0
         Buffers: shared hit=3
 Query Identifier: 5638644177116244590
 Planning:
   Buffers: shared hit=3
 Planning Time: 0.175 ms
 Execution Time: 0.066 ms
(13 rows)

Time: 1.052 ms
```

谢谢观看

THANK YOU