飞象数据 PostgresData

Sybase向PG迁移实践

赖伟

神州飞象（北京）数据科技有限公司

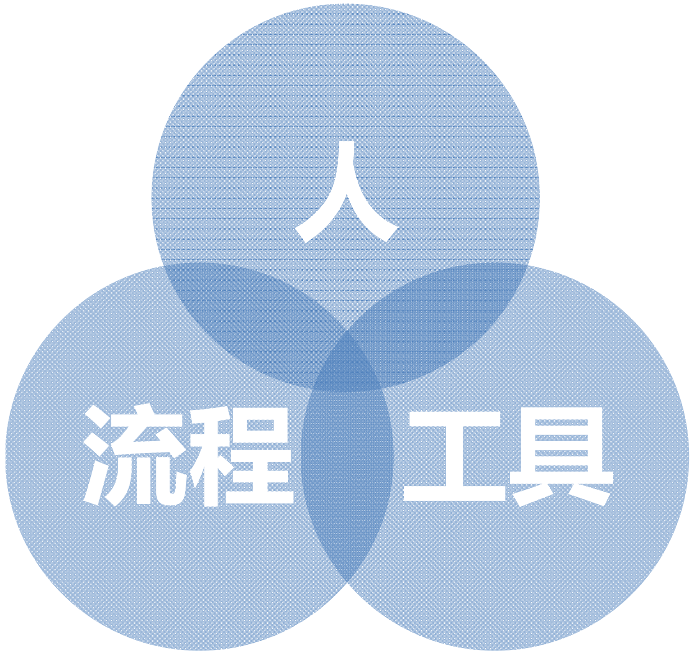
PostgreSQL

目录

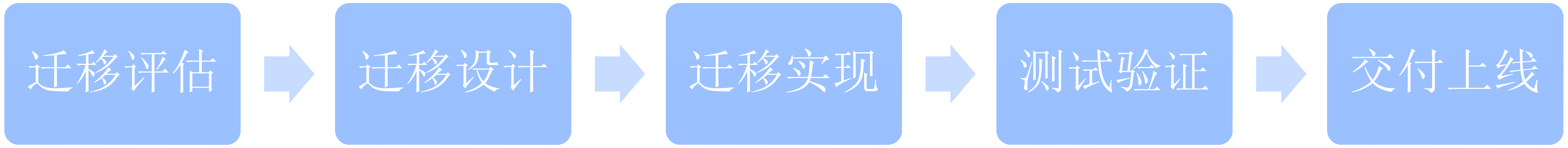
01
迁移方法介绍

迁移方法

- 架构主导
- 测试驱动
- 标准化
- 自动化



迁移流程



02

SYBASE迁移PG项目介绍

项目介绍

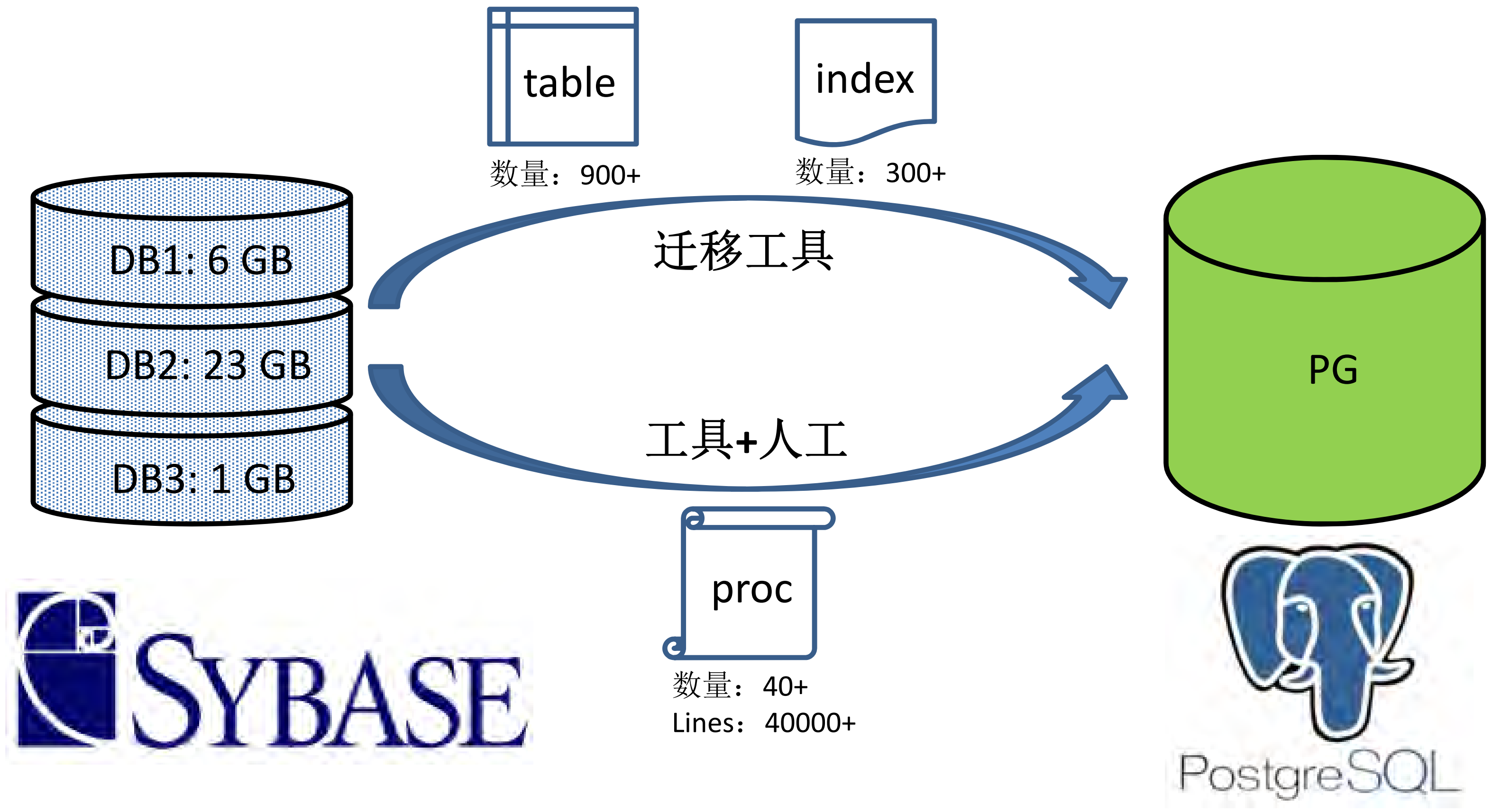
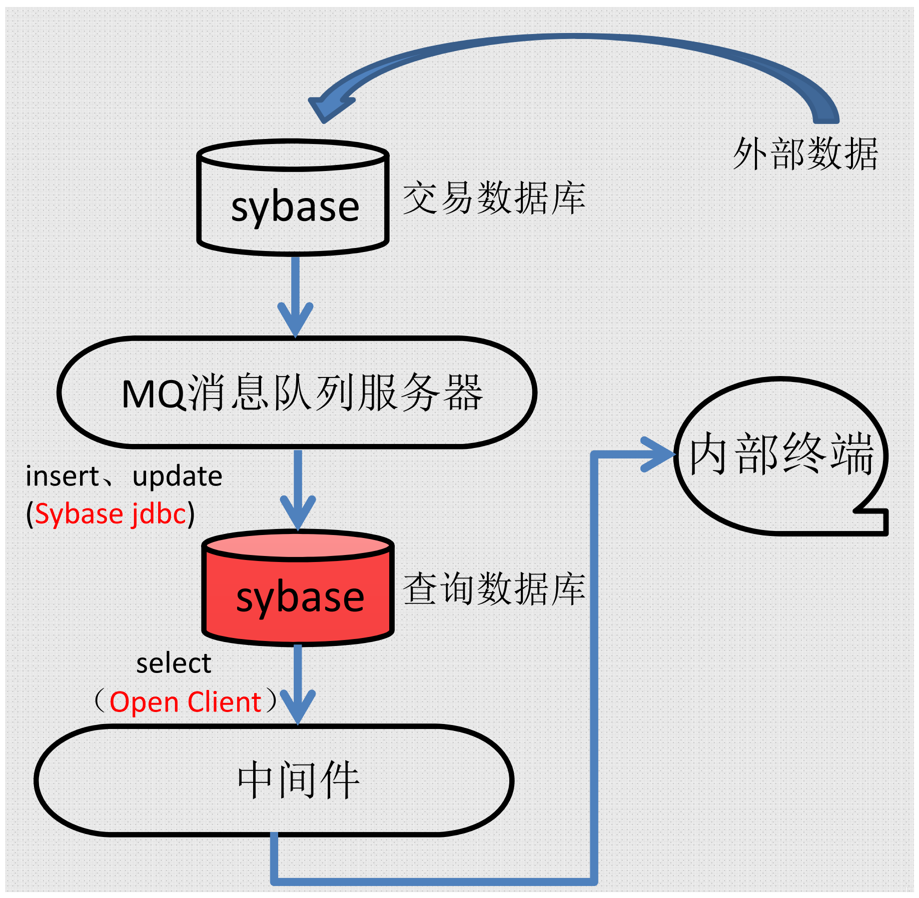
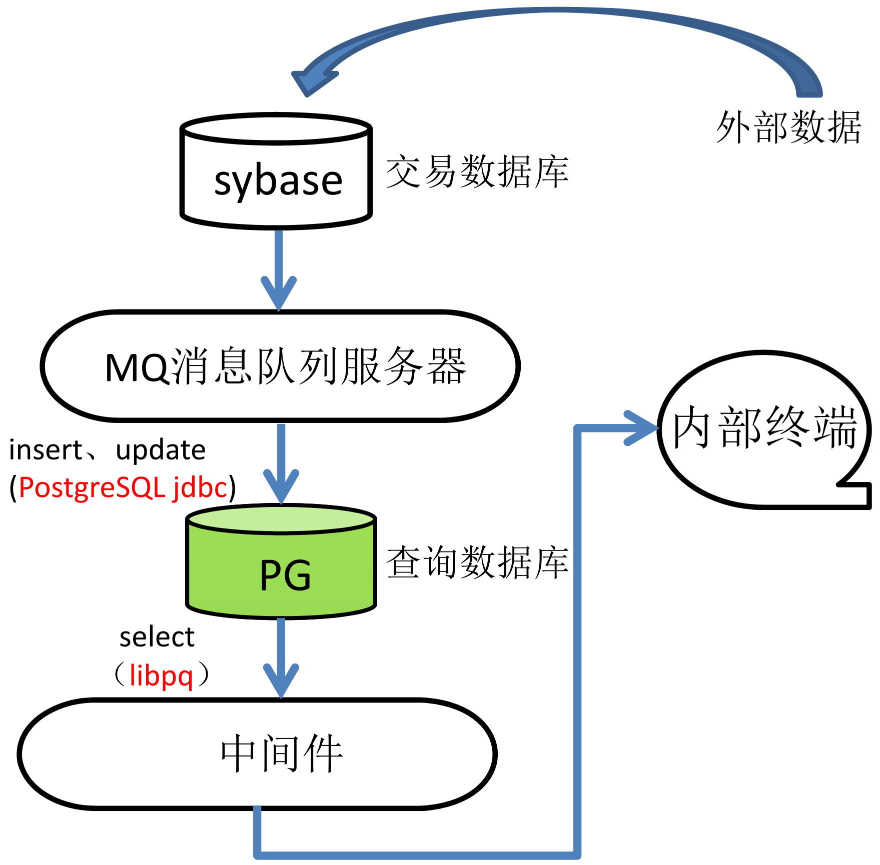


table
数量：900+
index
数量：300+
DB1: 6 GB
DB2: 23 GB
DB3: 1 GB
迁移工具
工具+人工
PG
proc
数量：40+
Lines：40000+
SYBASE
PostgreSQL

项目介绍



外部数据
sybase
交易数据库
MQ消息队列服务器
内部终端
insert、update
(Sybase jdbc)
sybase
查询数据库
select
（Open Client）
中间件



外部数据
sybase
交易数据库
MQ消息队列服务器
内部终端
insert、update
(PostgreSQL jdbc)
PG
查询数据库
select
（libpq）
中间件

迁移评估

名称	SYBASE	PostgreSQL	匹配情况	备注
JDBC驱动接口	SYBASE JDBC	PostgreSQL JDBC	匹配	消息队列调用
C驱动接口	Open Client	libpq	匹配	中间件调用
数据表	数据表	数据表	匹配	
索引	索引	索引	匹配	
临时表	临时表	临时表	匹配	
存储过程	存储过程	函数	匹配	
数据类型	tinyint	同名domain	匹配	
数据类型	datetime	同名domain	匹配	

迁移评估

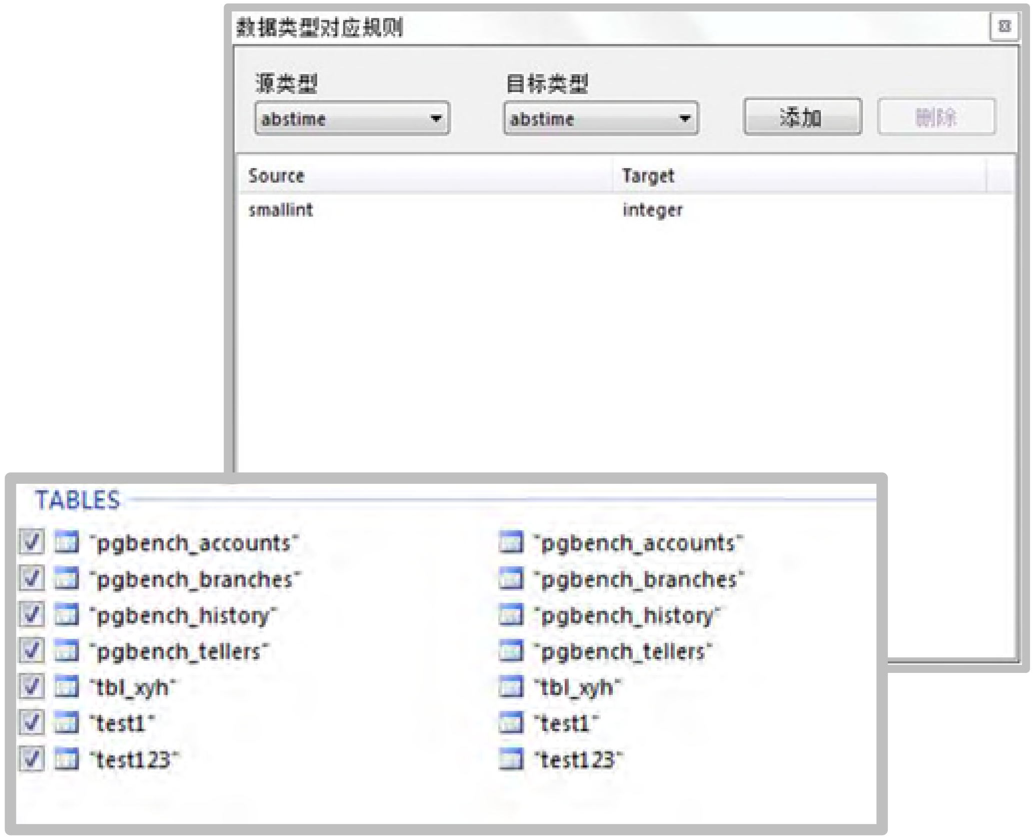
名称	SYBASE	PostgreSQL	匹配情况	备注
数据类型	CHAR	CHAR	匹配	
数据类型	INTEGER	INTEGER	匹配	
数据类型	VARCHAR	VARCHAR	匹配	
数据类型	SMALLINT	SMALLINT	匹配	
数据类型	DECIMAL	DECIMAL	匹配	

03

SYBASE to PG迁移实践分享

自动化、工具化

- 图像化操作界面，可自由筛选迁移对象。
- 支持数据类型对应规则。
- 迁移报错询问，避免中途回滚。



数据类型对应规则

源类型: abstime
目标类型: abstime

Source	Target
smallint	integer

TABLES

✓ "pgbench_accounts"	"pgbench_accounts"
✓ "pgbench_branches"	"pgbench_branches"
✓ "pgbench_history"	"pgbench_history"
✓ "pgbench_tellers"	"pgbench_tellers"
✓ "tbl_xyh"	"tbl_xyh"
✓ "test1"	"test1"
✓ "test123"	"test123"

自动化、工具化

块语句结尾关键词补全

```
IF @ex1 = 1
BEGIN
   select @time_1=getdate()
   select @word = "hellow,world"
END
```

```
IF p_ex1=1 THEN
  BEGIN
    v_time_1
=timeofday()::timestamp ;
    v_word = 'hellow,world';
  END;
END IF;
```

自动化、工具化

特殊语句标记

```
BEGIN
  select @time_1=getdate()
  GOTO A
END
PRINT @word
A:
PRINT @time_1
```

```
BEGIN
   v_time_1 =
timeofday()::timestamp ;
! GOTO A
 END;
RAISE NOTICE '%', v_word;
! A :
RAISE NOTICE '%', v_time_1;
```

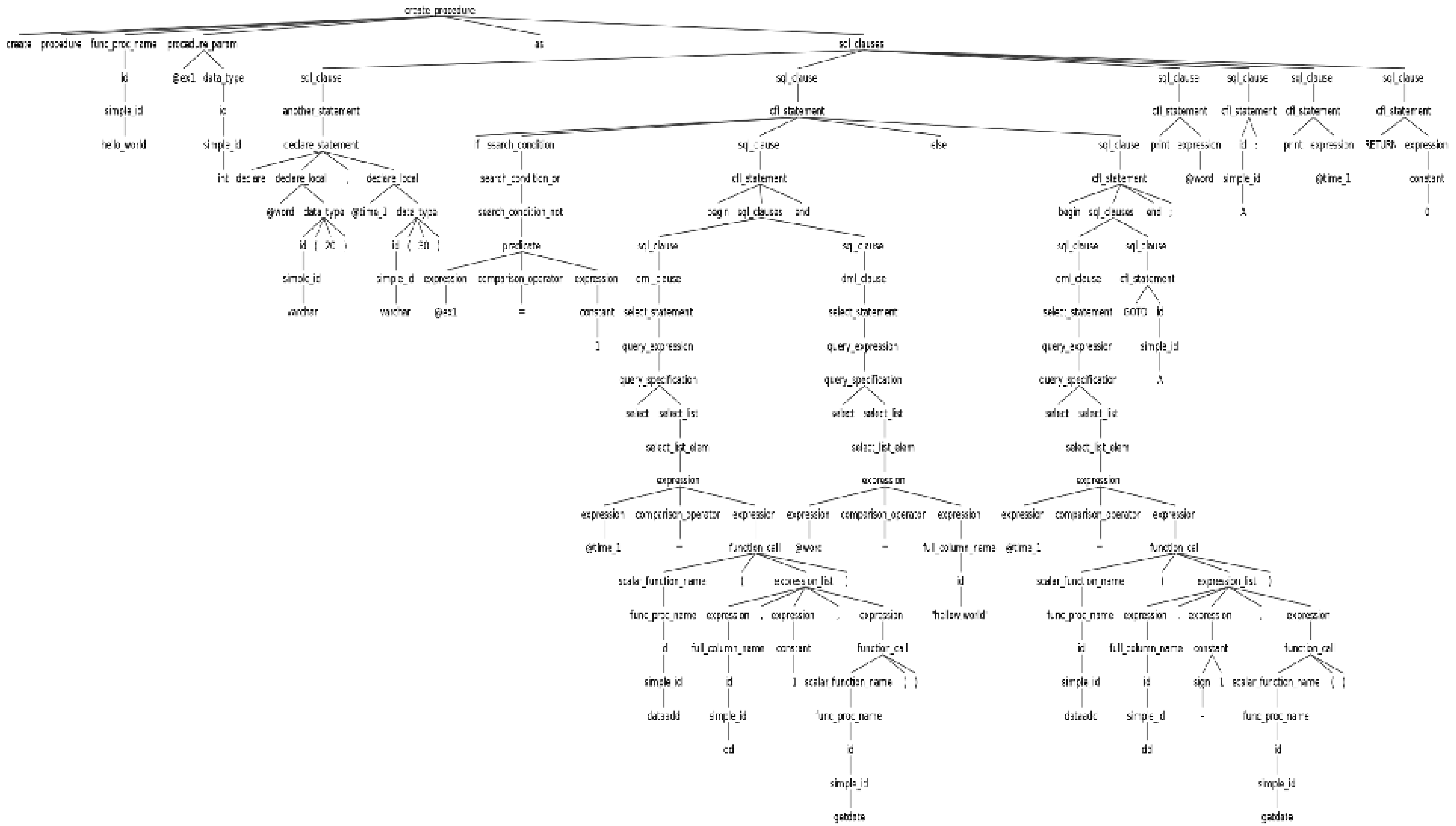
自动化、工具化

差异语句处理

```
IF @a = 1
BEGIN
   select @time_2 = time_clock
from p_info_list
PRINT @word
END
```

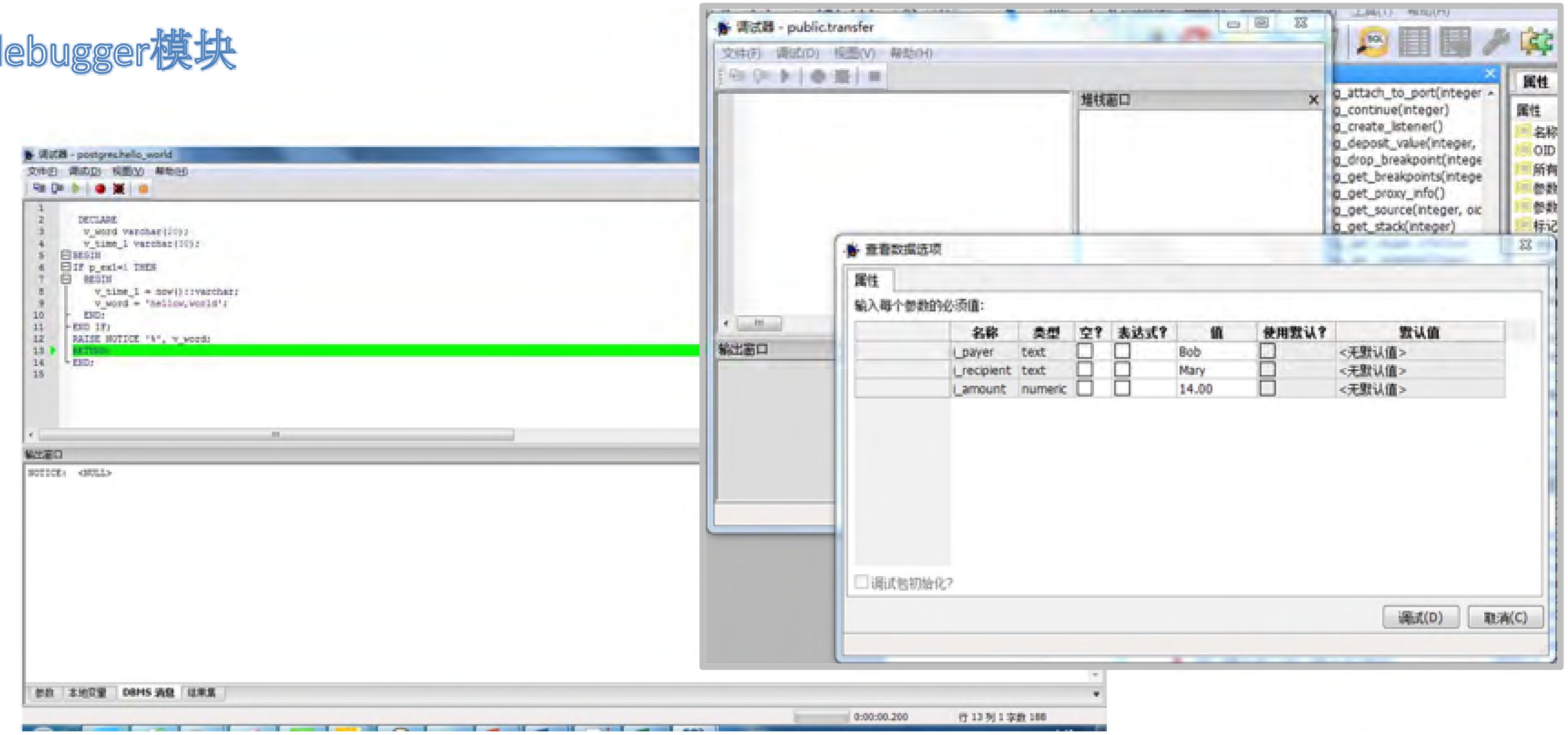
```
IF v_a =1 then
BEGIN
    select time_clock into v_time
from p_info_list ;
RAISE NOTICE '%', v_word;
 END;
END IF;
```

自动化、工具化


create_procedure
create procedure func_proc_name procedure_param as sql_clauses
id @ex1 data_type sql_clause sql_clause sql_clause sql_clause sql_clause sql_clause
simple_id id another_statement cfl_statement cfl_statement cfl_statement cfl_statement cfl_statement
hello_world simple_id declare_statement if search_condition sql_clause else sql_clause print expression id : print expression RETURN expression
int declare declare_local , declare_local search_condition_or cfl_statement cfl_statement @word simple_id @time_1 constant
@word data_type @time_1 data_type search_condition_not begin sql_clauses end begin sql_clauses end ; A 0
id (20) id (50) predicate sql_clause sql_clause sql_clause sql_clause
simple_id simple_id expression comparison_operator expression cfl_clause dml_clause dml_clause cfl_statement
varchar varchar @ex1 = constant select_statement select_statement select_statement GOTO id
1 query_expression query_expression query_expression simple_id
query_specification query_specification query_specification A
select select_list select select_list select select_list
select_list_elem select_list_elem select_list_elem
expression expression expression
expression comparison_operator expression expression comparison_operator expression expression comparison_operator expression
@time_1 = function_call @word = full_column_name @time_1 = function_call
scalar_function_name (expression_list) id scalar_function_name (expression_list)
func_proc_name expression , expression , expression 'hello world' func_proc_name expression , expression , expression
id full_column_name constant function_call id full_column_name constant function_call
simple_id id 1 scalar_function_name ( ) simple_id id sign 1 scalar_function_name ( )
dateadd simple_id func_proc_name dateadd simple_id - func_proc_name
dd id dd id
simple_id simple_id
getdate getdate

自动化、工具化

Pldebugger模块



调试器 - postgres.hello_world
文件(F) 调试(D) 视图(V) 帮助(H)
输出窗口
参数 本地变量 DBMS 消息 结果集
0:00:00.200
行 13 列 1 字数 188
调试器 - public.transfer
文件(F) 调试(D) 视图(V) 帮助(H)
堆栈窗口
输出窗口
属性
名称
OID
所有
参数
参数
标记

```
  DECLARE
   v_word varchar(20);
   v_time_1 varchar(30);
BEGIN
IF p_ex1=1 THEN
  BEGIN
    v_time_1 = now()::varchar;
    v_word = 'hellow,world';
  END;
END IF;
RAISE NOTICE '%', v_word;
RETURN;
END;
```

```
NOTICE:  <NULL>
```

```
g_attach_to_port(integer
g_continue(integer)
g_create_listener()
g_deposit_value(integer,
g_drop_breakpoint(intege
g_get_breakpoints(intege
g_get_proxy_info()
g_get_source(integer, oic
g_get_stack(integer)
```

查看数据选项

属性

输入每个参数的必须值:

名称	类型	空?	表达式?	值	使用默认?	默认值
i_payer	text	☐	☐	Bob	☐	<无默认值>
i_recipient	text	☐	☐	Mary	☐	<无默认值>
i_amount	numeric	☐	☐	14.00	☐	<无默认值>

☐ 调试包初始化?

调试(D) 取消(C)

利用PG可扩展性

datediff()

```
1> select datediff(hh,'2015-06-06
14:21:23','2015-06-07 12:20:23')
2> go
 -----------
         21

1> select
datediff(ms,'20150606','20150607')
2> go
 -----------
   86400000
```

```
postgres=# SELECT EXTRACT(DAY
FROM('2015-06-07 12:20:23'::timestamp-
'2015-06-06 14:21:23'::timestamp))*24
+EXTRACT(HOUR FROM ('2015-06-07
12:20:23'::timestamp-'2015-06-06
14:21:23'::timestamp));
 ?column?
----------
      21

postgres=# SELECT TRUNC(EXTRACT(EPOCH
FROM('20150607'::timestamp-
'20150606'::timestamp)))*1000;
 ?column?
----------
 86400000
```

利用PG可扩展性

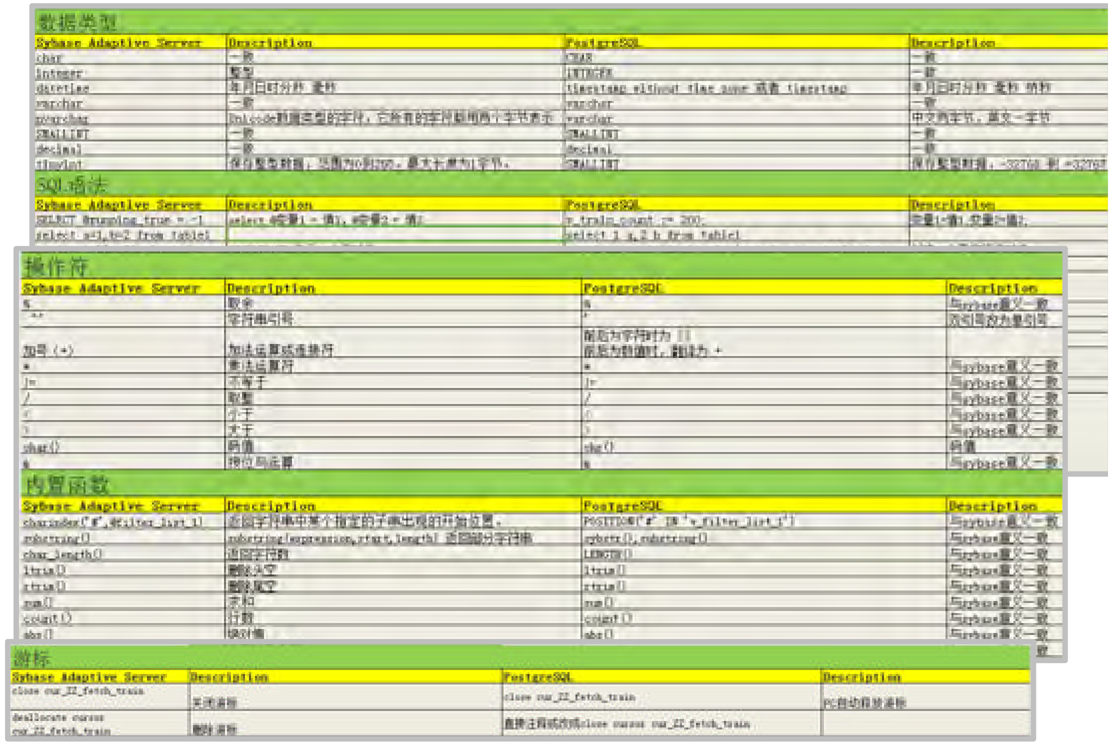
datediff()

```
1> select datediff(hh,'2015-06-06
14:21:23','2015-06-07 12:20:23')
2> go
 -----------
        21

1> select
datediff(ms,'20150606','20150607')
2> go
 -----------
   86400000
```

```
postgres=# select datediff('hh','2015-
06-06 14:21:23','2015-06-07 12:20:23');
 datediff
----------
      21

postgres=# select
datediff('ms','20150606','20150607');
 datediff
----------
 86400000
```

标准化

- 便于备忘查阅。
- 便于统一人工翻译格式。
- 便于提供追溯依据。



数据类型

Sybase Adaptive Server	Description	PostgreSQL	Description
char	一致	CHAR	一致
integer	整型	INTEGER	一致
datetime	年月日时分秒 毫秒	timestamp without time zone 或者 timestamp	年月日时分秒 毫秒 纳秒
varchar	一致	varchar	一致
nvarchar	Unicode数据类型的字符，它所有的字符都用两个字节表示	varchar	中文两字节，英文一字节
SMALLINT	一致	SMALLINT	一致
decimal	一致	decimal	一致
tinyint	保存整型数据，范围为0到255，最大长度为1字节。	SMALLINT	保存整型数据，-32768 到 +32767

SQL语法

Sybase Adaptive Server	Description	PostgreSQL	Description
SELECT @running_true = -1	select @变量1 = 值1, @变量2 = 值2;	v_train_count := 200;	变量1:=值1,变量2:=值2;
select a=1,b=2 from table1		select 1 a,2 b from table1	

操作符

Sybase Adaptive Server	Description	PostgreSQL	Description
%	取余	%	与sybase意义一致
""	字符串引号	'	双引号改为单引号
加号（+）	加法运算或连接符	前后为字符时为 || 前后为数值时，翻译为 +	
*	乘法运算符	*	与sybase意义一致
!=	不等于	!=	与sybase意义一致
/	取整	/	与sybase意义一致
<	小于	<	与sybase意义一致
>	大于	>	与sybase意义一致
char()	码值	chr()	码值
&	按位与运算	&	与sybase意义一致

内置函数

Sybase Adaptive Server	Description	PostgreSQL	Description
charindex('#',@filter_list_1)	返回字符串中某个指定的子串出现的开始位置。	POSITION('#' IN 'v_filter_list_1')	与sybase意义一致
substring()	substring(expression,start,length) 返回部分字符串	substr(), substring()	与sybase意义一致
char_length()	返回字符数	LENGTH()	与sybase意义一致
ltrim()	删除头空	ltrim()	与sybase意义一致
rtrim()	删除尾空	rtrim()	与sybase意义一致
sum()	求和	sum()	与sybase意义一致
count()	行数	count()	与sybase意义一致
abs()	绝对值	abs()	与sybase意义一致

游标

Sybase Adaptive Server	Description	PostgreSQL	Description
close cur_ZZ_fetch_train	关闭游标	close cur_ZZ_fetch_train	PG自动释放游标
deallocate cursor cur_ZZ_fetch_train	删除游标	直接注释或改成close cursor cur_ZZ_fetch_train	

标准化

案例1：跳转

```
A:
    SELECT ...
    IF ...  GOTO A
    INSERT ...
    RETURN
```

```
<<A>>
LOOP
    SELECT ...
    IF ... THEN CONTINUE A;
END IF;
    INSERT ...
    EXIT A;
END LOOP;
RETURN;
```

SYBASE

PostgreSQL

标准化

```
  SELECT ...
   IF ...  GOTO B
   INSERT ...
B:
  RETURN
```

```
SELECT ...
<<B>>
LOOP
    IF ... THEN EXIT B;
END IF;
    INSERT ...
    EXIT B;
END LOOP;
RETURN;
```

标准化

案例2：LIKE+char

```
1> CREATE TABLE TEST_8(a1
CHAR(10))
2> go
1> INSERT INTO TEST_8 SELECT ' A '
2> go
(1 row affected)
```

```
1> SELECT * FROM TEST_8
WHERE a1 like ' A       '
2> go
 a1
 ----------
  A
(1 row affected)
1> SELECT * FROM TEST_8
WHERE ' A       ' like a1
2> go
 a1
 ----------
  A
```

标准化

```
postgres=# CREATE TABLE
TEST_8(a1 CHAR(10));
CREATE TABLE

postgres=# INSERT INTO TEST_8
SELECT ' A ';
INSERT 0 1
```

```
postgres=# SELECT * FROM
TEST_8 WHERE a1 like ' A        ';
    a1
------------
  A
(1 row)

postgres=# SELECT * FROM
TEST_8 WHERE ' A        ' like a1;
 a1
----
(0 rows)
```

标准化

```
postgres=# SELECT * FROM TEST_8 WHERE a1 like ' A     '::char(10);
 a1
----
(0 rows)

postgres=# SELECT * FROM TEST_8 WHERE rtrim(a1) like ' A     '::char(10);
     a1
------------
  A
(1 row)
```

标准化

案例3：**rtrim()**

```
1> select rtrim('')
2> go


 -
 NULL
(1 row affected)

1> select rtrim(null)
2> go

(1 row affected)
```

```
postgres=# select
rtrim('')||'1';
 ?column?
----------
 1
(1 row)

postgres=# select
rtrim(null)||'1';
 ?column?
----------

(1 row)
```

标准化

案例4：取数据并赋值

```
 CREATE PORC test_pro8
@a  VARCHAR(10)  = "HELLO"
AS
BEGIN
    select @a=a1 from testgg
where 1=2
    print @a
END
```

```
1> CREATE TABLE testgg(a1
varchar(10))
2> go

1> EXEC test_pro8
2> go
HELLO
(return status = 0)
```

标准化

```
CREATE FUNCTION test_pro8
(a varchar(10)='HELLO')
RETURNS void AS
$$
BEGIN
    SELECT a1 INTO a FROM testgg
WHERE 1=2;
    RASIE NOTICE '%',a;
END;
$$
LANGUAGE plpgsql ;
```

```
postgres=# create table
testgg(a1 varchar(10));
CREATE TABLE

postgres=# select * from
test_pro8();
NOTICE:  <NULL>
 test_pro8
-----------

(1 row)
```

标准化

```
CREATE FUNCTION test_pro8
(a varchar(10)='HELLO')
RETURNS void AS
$$
DECLARE
cur REFCURSOR;
BEGIN
   OPEN cur FOR SELECT a ;
   SELECT a1 INTO a FROM testgg WHERE 1=2;
   IF NOT FOUND THEN
    FETCH cur INTO a;
   END IF;
   RAISE NOTICE '%',a;
END;
$$
LANGUAGE plpgsql ;
```

```
postgres=# create table testgg(a1
varchar(10));
CREATE TABLE

postgres=# select test_pro8();
NOTICE:  HELLO
 test_pro8
-----------


(1 row)
```

标准化

案例5：子过程调用

```
CREATE proc proc10
@b int output
AS
SELECT @b=@b+1
RETURN 3

CREATE proc proc11
AS
DECLARE
@a INT,@c INT
SELECT @c=5
EXEC @a=proc10 @b=@c OUTPUT
select @a AS a,@c AS c
```

```
1> EXEC proc11
2> GO
 a           c
 ----------- -----------
           3           6

(1 row affected)
(return status = 0)
```

标准化

```
CREATE proc proc10
@b int output
AS
SELECT @b=@b+1
RETURN 3
```

```
CREATE FUNCTION proc10(INOUT
b INT,INOUT v_ret INT=null)
AS $$
BEGIN
        $1=$1+1;
        $2=3;
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

标准化

```
CREATE proc proc11
AS
DECLARE
@a INT,@c INT
SELECT @c=5
EXEC @a=proc10 @b=@c
OUTPUT
select @a AS a,@c AS c
```

```
CREATE FUNCTION proc11()
RETURNS VOID AS $$
DECLARE
 a INT;c INT ;
BEGIN
    c=5;
    SELECT p.v_ret,p.b INTO
    a,c  from  proc10(b:=c) as p;
            RAISE NOTICE '%',a||' '||c;
END;
$$
LANGUAGE plpgsql;
```

标准化

```
postgres=# select proc11();
NOTICE:  3 6
 proc11
--------

(1 row)
```

Thanks!

Q & A