ПРОЦЕДУРНОЕ РАСШИРЕНИЕ SQL

Процедурный язык PSQL Firebird

ЦЕЛИ

- Совместное хранение данных и алгоритмов
- Программирование на стороне сервера
- Параметризация

ПРОБЛЕМА

- Нет стандарта
- Многие синтаксические конструкции пришли из ЯП, использовавших встроенный SQL

PSQL Позволяет использовать

- хранимые процедуры
- хранимые функции
- пакеты
- триггеры
- выполнимые блоки

Хранимые процедуры

Преимущества использования

- модульность
- упрощение поддержки приложений
- увеличение производительности

Типы

- выполнимые (executable)
- селективные (selectable)

Элементы PSQL

- модифицированные DML – операторы
- локальные переменные
- основные конструкции классических ЯП
- средства для работы с исключениями

!!! не допустимы операторы DDL, операторы старта и завершения транзакции, но

- есть возможность работы с любыми операторами, записанными в виде символьной строки

Заголовок и тело

```
{CREATE | RECREATE | ALTER | CREATE  OR  ALTER }

CREATE  OR  ALTER . . .
AS
. . .
BEGIN

END
```

Согласование синтаксиса

Конфликт использования «;» для разных целей

```
SET TERM ^ ;

{CREATE | RECREATE | ALTER } PROCEDURE <имя>
[(<список входных параметров>)]
[RETURNS (<список выходных параметров>)]
AS
[<объявления локальных переменных>]
BEGIN
<один или несколько блоков операторов>
END ^
COMMIT^

SET TERM ; ^
```

Согласование синтаксиса

```
// агенты, которые не работали со складом, заданным по названию

create or alter procedure P1 (  sNAME char(20) )
                                returns ( sAGENT char(20) )
as
begin
 for select  a.name_ag
             from agent a
             where not exists    (select 1
                         from operation op join  warehouse w using(id_wh)
                         where a.id_ag=op.id_ag and            w.name= :sNAME)
into :sAGENT do
 suspend;
end
```

Синтаксис (не полный)

```
CREATE PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
{<psql-body> | <external-body>}

<psql-body> ::=
  AS
     [<declarations>]
  BEGIN
     [<PSQL_statements>]
  END

<external-body> ::=
  EXTERNAL NAME '<extname>' ENGINE <engine>
  [AS <extbody>]
```

Объявление локальных переменных

DECLARE [VARIABLE] <*имя*> < *тип*> [NOT NULL]
[=<*начальное значение*>];

И для переменных, и для параметров возможно в качестве типа использовать домен или тип колонки существующей таблицы

```
CREATE OR ALTER procedure P1
        ( NAME type of column WAREHOUSE.NAME)
  returns ( AGENT type of column AGENT.NAME_AG)
```

Использование локальных переменных

Особенность в операторах DML
(select, insert, update, delete, execute procedure)
должны предваряться префиксом «:»

Это «наследие» встроенного SQL в языки программирования

Встроенный SQL

```
EXEC SQL     BEGIN DECLARE  SECTION;
      int emp_no;
      char [30] fname, [30]lname;
EXEC SQL     END DECLARE SECTION;
EXEC SQL
   CONNECT '. . .\employee.fdb'  USER 'STUDENT' PASSWORD '123abc';
emp_no=120;
EXEC SQL
    SELECT FIRSTNAME, LASTNAME
     FROM EMPLOYEE WHERE EMP_NO = :emp_no
     INTO :fname, :lname;
```

Использование локальных переменных

```
declare variable min_price numeric(15,2);
begin
  min_price = 0.1;
  update OPERATION
         set price=price*1.5
  where price< :min_price;
end
```

!!входные и выходные параметры, тоже локальные переменные

Структурные операторы PSQL

BEGIN ... END	Блок
<переменная> = <выражение>	Оператор присваивания
/*.....*/	Многострочный комментарий
-- ...	Комментарий до конца строки
IF (...) THEN ... [ELSE ...]	Оператор ветвления
WHILE (...) DO ...	Оператор цикла
EXCEPTION ...	Вызвать исключение

Структурные операторы PSQL

LEAVE [label]	Завершить цикл
CONTINUE [label]	Продолжить итерацию цикла
EXIT	Завершить процедуру
WHEN ….	Обработать исключение
SUSPEND	Для процедур выбора, передает одну строку результата, приостанавливая выполнение процедуры до тех пор, пока эту строку не получит вызвавшее приложение. Для выполнимых процедур эквивалентен EXIT
EXECUTE {STATEMENT| PROCEDURE}	Вызов процедуры или выполнение выражения, содержащего оператор DML

Однострочный оператор SELECT

SELECT

<*спецификация оператора* SELECT>

INTO

<*список локальных переменных*>;

Можно считать оператором присваивания для кортежа

Допустимо использовать в случае корректности

<var> = (select ... from ...);

Пример

```
SELECT SUM(BUDGET), AVG(BUDGET)
    FROM  DEPARTMENT
WHERE HEAD_DEPT = :head_dept
INTO :tot_budget, :avg_budget;

max_salary = (select max(salary)
                        from employee
                        where job_country= :cntr);
```

Многострочный оператор SELECT

- Курсорный цикл
- Реализует перебор строк, возвращаемых запросом

```
FOR
   <select_stmt>
   [INTO <variables>]
   [AS CURSOR cursorname]
DO <compound_statement>
```

FOR SELECT
    <*спецификация оператора* SELECT>
INTO <*список переменных*> DO
BEGIN
  <*блок обработки*>
 [SUSPEND;]
 [WHEN … <*обработка исключения* >]
END

Пример

```
sum =0;
for  select salary from employee
      where dept_no = :DN                    // DN – номер отдела
  into :SL  do
begin
  sum = sum +SL;
  -- suspend;  // возвращает сумму  «нарастающим итогом»
end
suspend;       //итоговая сумма
```

Оператор SUSPEND

- Передает значения выходных параметров в буфер
- Приостанавливает выполнение хранимой процедуры (блока) до тех пор, пока вызывающая сторона не получит содержимое буфера
- Выполнение продолжается с оператора, непосредственно следующего за SUSPEND
- Использование внутри цикла позволяет сформировать на выходе процедуры таблицу

Курсоры

Тоже понятие, пришедшее из «встроенного SQL»

- Курсор связывается с оператором SELECT
- Работа с курсором – OPEN, FETCH, CLOSE
- Оператор OPEN запрашивает у СУБД описание таблицы результатов запроса и помещает его в специальную область, которая используется в операции FETCH (SQLDA)

Встроенный SQL

```
EXEC SQL
    DECLARE to_be_hired CURSOR FOR
            SELECT D.DEPARTMENT, D.LOCATION,  P.DEPARTMENT
                    FROM DEPARTMENT D JOIN DEPARTMENT P
                    ON (AND D.HEAD_DEPT = P.DEPT_NO)
            WHERE D.MNGR_NO IS NULL;
EXEC SQL    OPEN to_be_hired;
EXEC SQL   FETCH to_be_hired     INTO :deptname, :lname, :head_deptname;
while (! sqlca.sqlcode ) {
   cout<< deptname <<" | "<< lname " | "<< head_deptname;
   EXEC SQL  FETCH to_be_hired INTO :deptname, :lname, :head_deptname;
}
EXEC SQL CLOSE to_be_hired;
```

Курсоры в PSQL

DECLARE <имя> CURSOR FOR
(<*спецификация оператора* SELECT>);

OPEN <имя>;

FETCH <имя>
INTO <*список переменных*>;

ROW_COUNT - 1/0 вернул ли FETCH строку

CLOSE <имя>;

Пример

```
create or alter procedure AG_LIST ( TOWN type of column AGENT.TOWN)
      returns ( NAME type of column AGENT.NAME_AG)
as
declare C_AG cursor for ( select AG.NAME_AG from AGENT AG
                                    where AG.TOWN = :TOWN);
begin
open C_AG; -- курсор нужно явно открыть
while (1=1) do begin
      fetch C_AG into :name;--извлечь строку по курсору
      if (row_count=0) then leave; --строк нет
      suspend;
end
close C_AG;--закрыть курсор
end
```

Редактирование по курсору

После выполнения операции FETCH курсор сохраняет адрес расположения результата, если курсор связан с запросом к одной таблице – это адрес расположения строки таблицы (ROWID)

```
UPDATE <имя таблицы>
    SET <список изменений>
    WHERE CURRENT OF <имя курсора>;

DELETE FROM <имя таблицы>
    WHERE CURRENT OF <имя курсора>;
```

```
open C_AG;
while (1=1) do
begin
     fetch C_AG into :name ;
     if (row_count=0) then leave;
      if (. . .) -- какое-то условие для агентов, например «каждый 5-й»
              then
     update agent  set phone=coalesce(phone,' ') || '+5'
           where current of C_AG;          --позиция курсора
suspend;
end
close C_AG;
```

Неявный курсор

```
for select AG.name_ag  from agent AG
            where AG.town=:town
            into :name
            as cursor CRU
do begin
 if (. . . ) then
      update agent  set phone=coalesce(phone,' ') || '+5'
            where current of CRU;
suspend;
end
```

Выполнить процедуру

EXECUTE PROCEDURE <процедура> [(<параметры>)] ;

Для процедур, возвращающих результат

SELECT <. . .>
FROM <процедура> [(<параметры>)]
[. . .]

Пример

```
CREATE PROCEDURE     T_KVADR (n
      INTEGER)
RETURNS (i INTEGER, kv INTEGER)
AS
BEGIN
  /* ТЕЛО ПРОЦЕДУРЫ */
  i=1;
  WHILE (i<=n) DO
  BEGIN
       kv=i*i;
      SUSPEND;
      i=i+1;
  END
END
```

Вызов

В командном окне

```
select * from  T_KVADR(10);
```

В другой процедуре

```
for select * from T_KVADR(10)
        into :x, :kvx
do
begin
. . .  -- что-то делаем с x и kvx
end
```

Примеры из БД employee

ADD_EMP_PROJ (emp_no SMALLINT, proj_id CHAR(5))

execute procedure ADD_EMP_PROJ (145,'GUIDE');

Обработка ошибки – выбрасывает исключение

DEPT_BUDGET (dno CHAR(3))

По номеру отдела определяет его суммарный бюджет (включая все подотделы). Рекурсивная.

select * from DEPT_BUDGET ('000');

Формирование и выполнение SQL-оператора

EXECUTE STATEMENT *<select-statement>*
INTO *<var>* [, *<var>* ...] ;

FOR EXECUTE STATEMENT *<select-statement>*
INTO *<var>* [, *<var>* ...]
DO *<psql-statement>;*

Позволяет также выполнять операторы DDL (create, alter, drop)

Формирование и выполнение SQL-оператора

```
create procedure TAB_RANG (name_t char(31))
                          returns      (KOL integer)
as
declare variable OPER varchar(200);
begin
    OPER = 'select count(*) from  ' || name_t;
    execute statement OPER into :KOL ;
    suspend;
end
```

Опасно при использовании строковых параметров!!!

Исключения

Исключение – объект схемы БД, создается/изменяется/удаляется командами CREATE/ALTER/DROP

CREATE EXCEPTION *<exception-name>* [*custom-message*];

В процедуре выбросить исключение:

EXCEPTION [*<exception-name>* [*message*]];

```
exception EX_BAD_ID     'Wrong id for input'|| emp_id;
```

Сообщение

Bad id exception Wrong id for input 1

Обработка ошибки/исключения

```
begin
. . .
when { <имя исключения>|
          SQLCODE  <код> |GDSCODE <код> | ANY}
          do <оператор>
[when… do <оператор>]
end
```

Обработка ошибки/исключения

```
begin
. . .
when any do begin
    in autonomous transaction
        insert into error_log (...) values (sqlcode, ...);
    exception;
end
end
```

Процедуры и транзакции

- Не допускается выполнения операций старта/фиксации/отмены транзакции
- Процедура выполняется в рамках какой-то транзакции
- Отдельный оператор/блок может быть выполнен в автономной транзакции, ее завершение не зависит от завершения/отката транзакции, в которой выполняется процедура

Автономные транзакции

- В случае успешного завершения фиксируется немедленно не дожидаясь завершения процедуры
- Выполняется с теми же параметрами, что и транзакция, в которой выполняется процедура
- Не зависит от родительской транзакции, поэтому может входить с ней в блокировку

Автономные транзакции

IN AUTONOMOUS TRANSACTION
DO *<psql-statement>;*

[FOR] EXECUTE STATEMENT
sql-statement
WITH {AUTONOMOUS|COMMON} TRANSACTION
[...other options...]
[INTO *<variables>*]
[DO *psql-statement*]

Использование общих табличных выражений

```
FOR
WITH  MY_AGENTS   AS ( SELECT * FROM AGENT
          WHERE TOWN = 'Ростов')
SELECT NAME_AG, PHONE
          FROM MY_AGENTS
INTO :ANAME, :APHONE
DO
BEGIN
...
END
```

Выполнимые блоки

Аналог хранимой процедуры, но не имеет имени и хранится на клиенте, может быть встроен в программный код на ЯВУ

Позволяет делать отладку будущих процедур «на лету»

Синтаксис

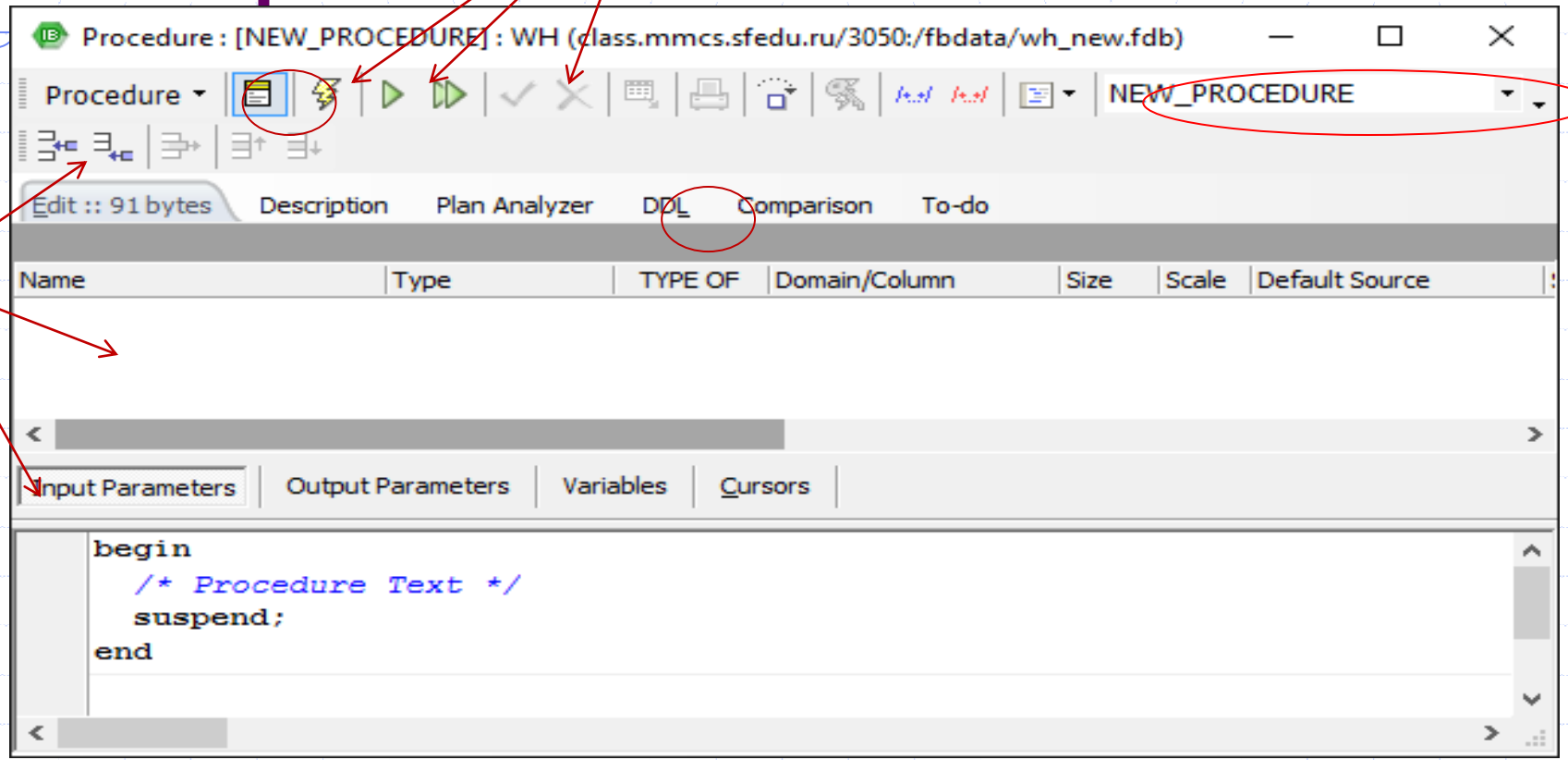
EXECUTE BLOCK [(*<inparams>*)]
[RETURNS (*<outparams>*)]
AS
[*<declarations>*]
BEGIN
[*<PSQL statements>*]
END

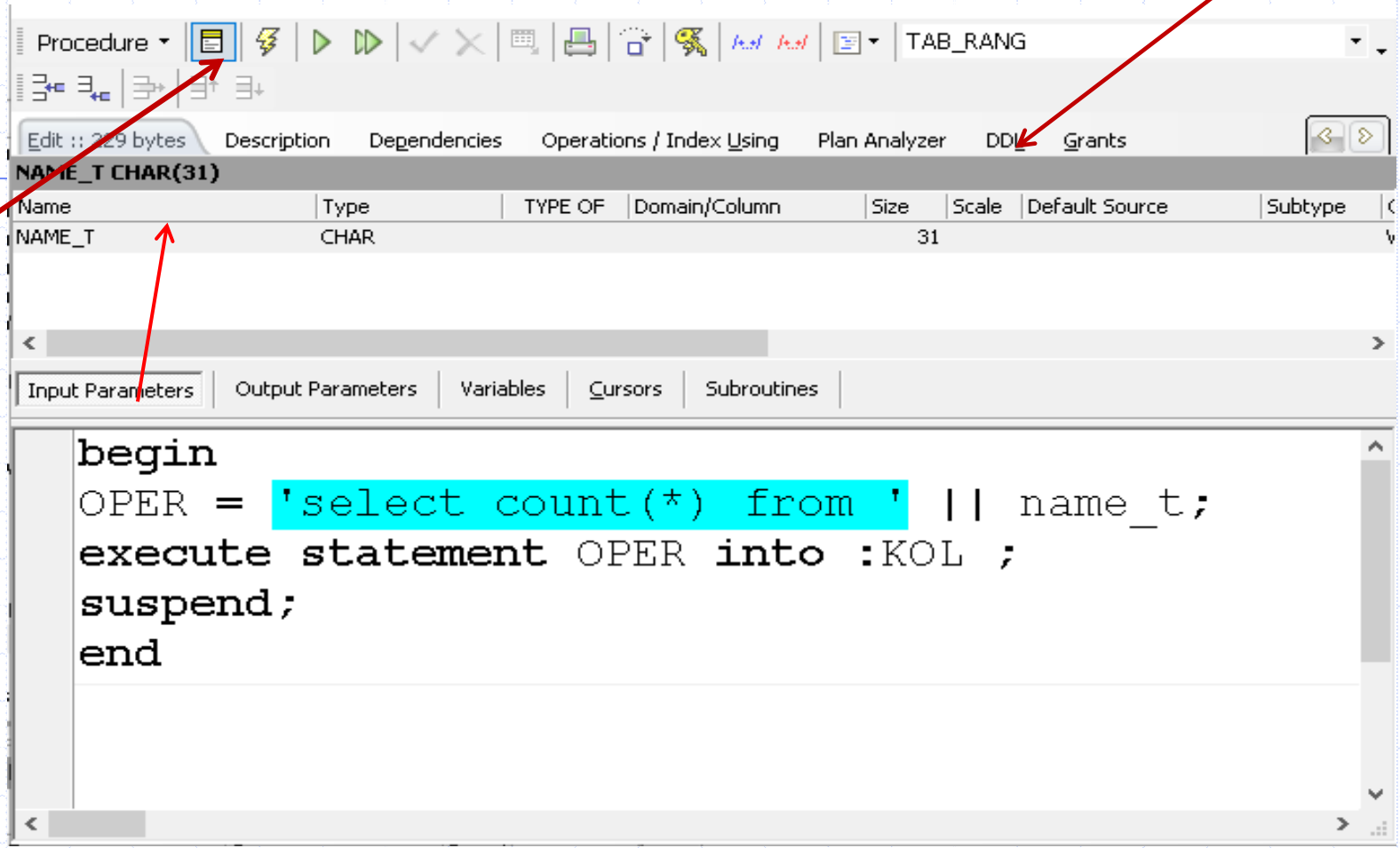
<inparams> ::= <name> <type> = ? [, <inparams>]
<outparams> ::= <name> <type> [, <outparams>]

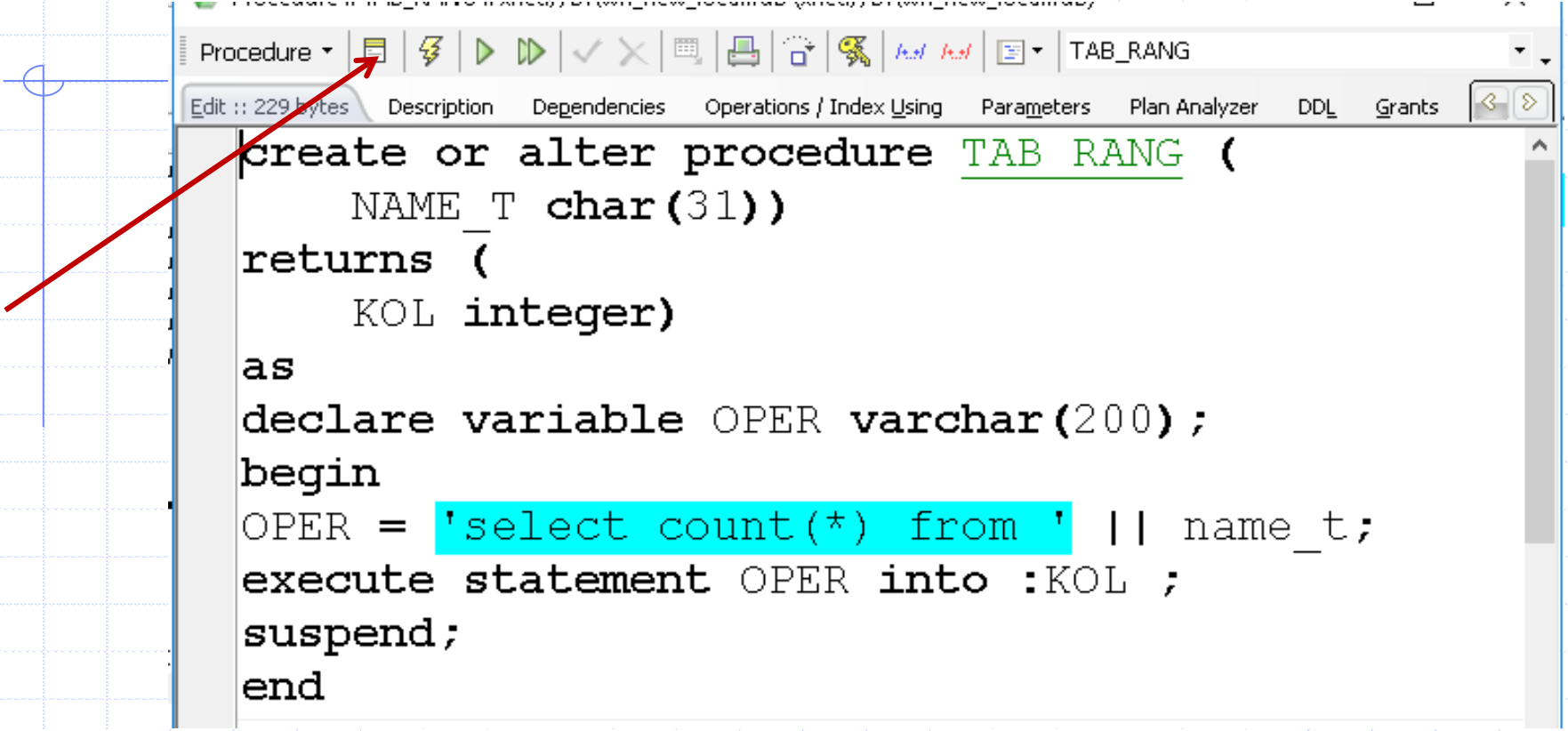
Пример

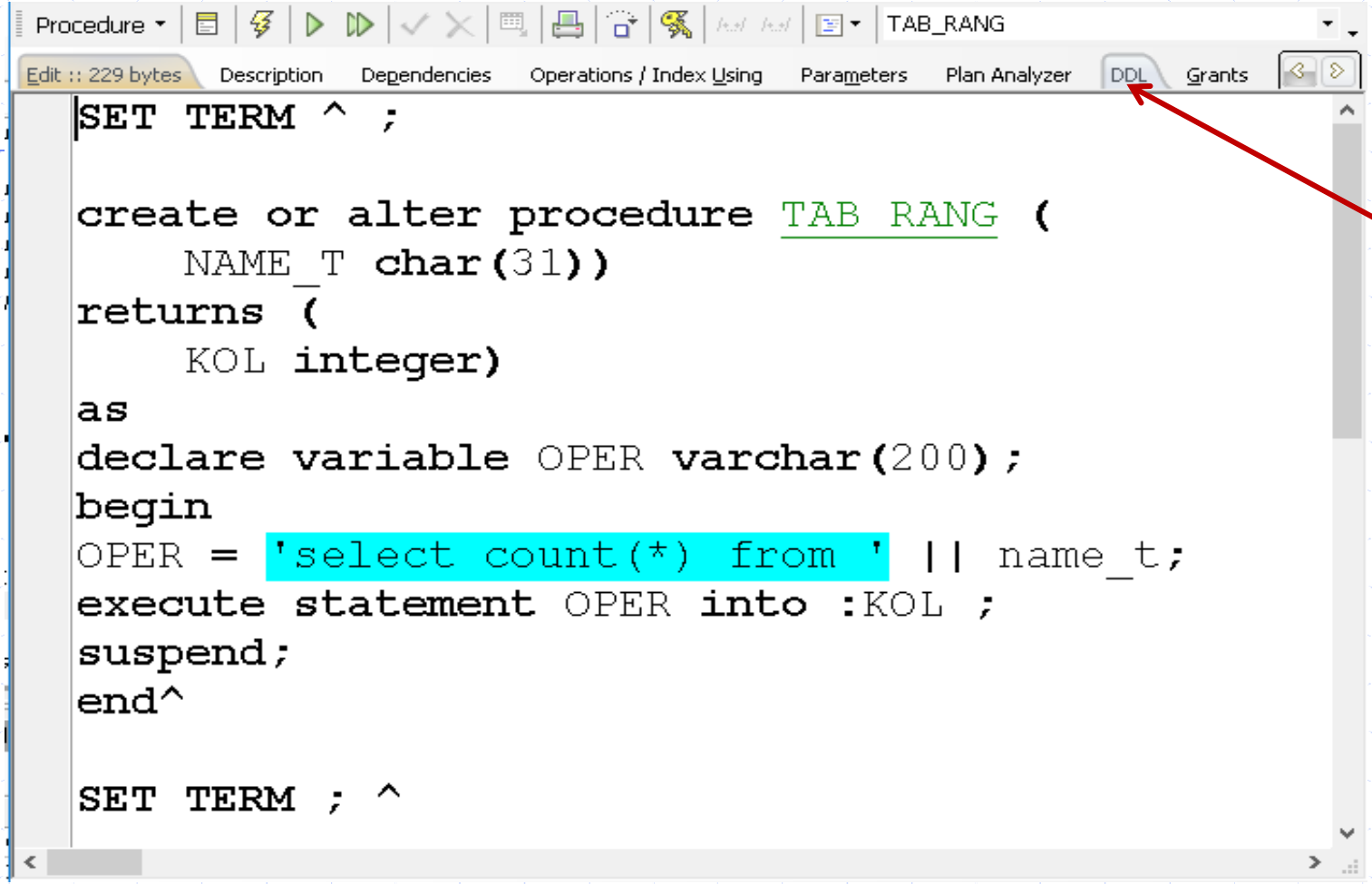
```
execute block
as
declare i int = 0;
begin
 while (i < 128) do
   begin
        insert into AsciiTable
          values (:i, ascii_char(:i)); i = i + 1;
    end
end
```

IBexpert


Procedure : [NEW_PROCEDURE] : WH (class.mmcs.sfedu.ru/3050:/fbdata/wh_new.fdb)
Procedure
NEW_PROCEDURE
Edit :: 91 bytes
Description
Plan Analyzer
DDL
Comparison
To-do
Name
Type
TYPE OF
Domain/Column
Size
Scale
Default Source
Input Parameters
Output Parameters
Variables
Cursors
begin
/* Procedure Text */
suspend;
end


Procedure
TAB_RANG
Edit :: 329 bytes
Description
Dependencies
Operations / Index Using
Plan Analyzer
DDL
Grants
NAME_T CHAR(31)
Name
Type
TYPE OF
Domain/Column
Size
Scale
Default Source
Subtype
NAME_T
CHAR
31
Input Parameters
Output Parameters
Variables
Cursors
Subroutines
begin
OPER = 'select count(*) from ' || name_t;
execute statement OPER into :KOL ;
suspend;
end


Procedure
TAB_RANG
Edit :: 229 bytes
Description
Dependencies
Operations / Index Using
Parameters
Plan Analyzer
DDL
Grants
create or alter procedure TAB_RANG (
NAME_T char(31))
returns (
KOL integer)
as
declare variable OPER varchar(200);
begin
OPER = 'select count(*) from ' || name_t;
execute statement OPER into :KOL ;
suspend;
end


Procedure
TAB_RANG
Edit :: 229 bytes
Description
Dependencies
Operations / Index Using
Parameters
Plan Analyzer
DDL
Grants
SET TERM ^ ;
create or alter procedure TAB_RANG (
NAME_T char(31))
returns (
KOL integer)
as
declare variable OPER varchar(200);
begin
OPER = 'select count(*) from ' || name_t;
execute statement OPER into :KOL ;
suspend;
end^
SET TERM ; ^