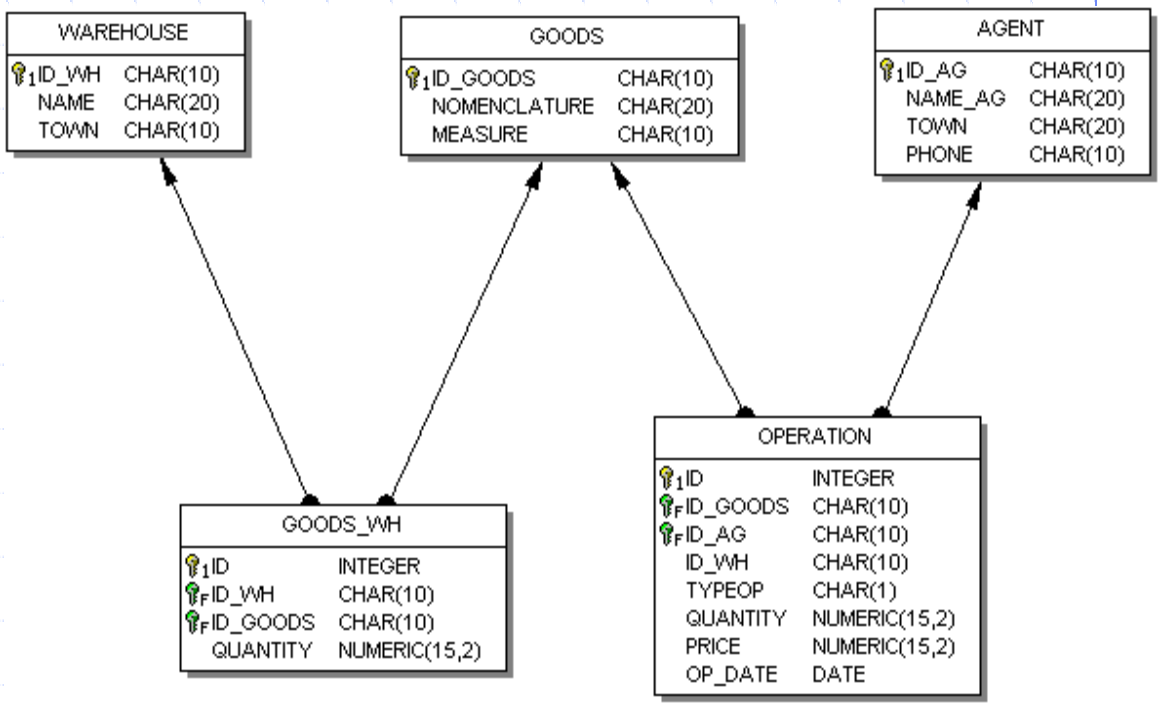
ПРОЦЕДУРЫ И ТИГГЕРЫ

ЦЕЛИ

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




```
select  a.name_ag
from agent a
where not exists
    (select 1
        from operation op join  warehouse w
                    on op.id_wh=w.id_wh
        where a.id_ag=op.id_ag
                and w.name= 'Склад 1')
```

```
select  a.name_ag
from agent a
where not exists
    (select 1
        from operation op join  warehouse w
                    on op.id_wh=w.id_wh
        where a.id_ag=op.id_ag and
  w.naimen= ? )
```

```
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
                           on op.id_wh=w.id_wh
                  where a.id_ag=op.id_ag and
                           w.name= :sNAME)
into :sAGENT do
 suspend;
end
```

Особенности синтаксиса

```
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 ; ^
```

Смешанный синтаксис

{CREATE | RECREATE | ALTER }
CREATE OR ALTER PROCEDURE

Объявление локальных переменных

DECLARE VARIABLE <*имя*> < *тип*>
[=<*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 «:».

Пример

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

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	Завершить цикл
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 выполняемых процедур эквивалентен EXIT

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

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

```
sum =0;
for  select salary from employee
       where dept_no = :DN
  into :SL  do
begin
  sum = sum +SL;
  -- suspend;
end
```

Курсоры

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 PROCEDURE SHOW_TABL_NAMES
            RETURNS ( RNAME CHAR(31)) AS
DECLARE C CURSOR FOR
( SELECT  RDB$RELATION_NAME  FROM  RDB$RELATIONS );
BEGIN
   OPEN C;
   WHILE (1 = 1) DO
   BEGIN
         FETCH C INTO :RNAME;
         IF (ROW_COUNT = 0) THEN  LEAVE;
         SUSPEND;
   END
   CLOSE C;
END
```

Редактирование по курсору

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

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

Выполнить оператор/процедуру

EXECUTE STATEMENT <строка>;
EXECUTE PROCEDURE <процедура> ;

Для процедур, возвращающих результат

SELECT <. . .>
FROM <процедура>
[. . .]

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 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!!!

Возможный синтаксис

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)

Исключения

EXCEPTION [*<exception-name>* [*custom-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
```

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

Выполнимые блоки

Начиная с версии 2.0

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

Позволяет отлаживать будущие процедуры «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
```

Пример

```
execute block (x double precision = ?,
                              y double precision = ?)
        returns (gmean double precision)
as
begin
gmean = sqrt(x*y);
suspend;
end
```

Процедуры и транзакции

- Не допускается выполнения операций старта/фиксации/отмены транзакции
- Процедура выполняется в рамках какой-то транзакции

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*]

Триггеры

```
CREATE [OR ALTER] TRIGGER имя FOR таблица
    {ACTIVE|INACTIVE}
    {BEFORE|AFTER}{DELETE|INSERT|UPDATE}
    [POSITION число]
    AS
    [DECLARE локальные переменные
    . . .]
    BEGIN
    операторы
    END^
```

Триггеры

Совместимость с SQL 2003

```
CREATE TRIGGER имя
    {ACTIVE|INACTIVE}
    {BEFORE|AFTER}
    {DELETE|INSERT|UPDATE}
    [POSITION число]
    ON таблица
AS
. . .
```

Атрибут состояния триггера

ACTIVE | INACTIVE

Перевод триггера из одного состояния в другое осуществляется командой ALTER TRIGGER.

```
alter trigger TR_DU for TABLE_1 inactive;
```

Операции, активирующие триггер

INSERT | UPDATE | DELETE

Несколько операций

```
create trigger TR_DU for TABLE_1
      before DELETE OR UPDATE
```

логические контекстные переменные для проверки

INSERTING, UPDATING, DELETING

Фазы выполнения

BEFORE | AFTER

Объединение фаз BEFORE и AFTER недопустимо, т.к. триггеры разных фаз имеют разные возможности

Контекстные переменные NEW и OLD

Содержат состояние строки, подвергающейся модификации оператором INSERT, UPDATE или DELETE (там, где это имеет смысл)

OLD - до выполнения операции модификации

NEW - после

В триггерах фазы BEFORE контекстная переменная NEW для операций INSERT и UPDATE может быть подвергнута изменению.

Именно измененное значение будет использовано для завершения выполнения операции.

Индикатор последовательности запуска

POSITION n

Процесс выполнения операции

- Формирование данных и передача серверу (посылка)
- При получении посылки активируются триггеры фазы BEFORE
- Если триггеры завершились успешно, проверка ограничений (CONSTRAINTS)
- Если все ограничения проверены успешно, активируются триггеры фазы AFTER

Процесс выполнения операции

- Если триггеры завершились успешно, сохранение версии данных
- Для фиксации версии данных COMMIT
- Если на любом этапе триггеры или ограничения вызывают исключение транзакция помечается как ROLLED BACK

Использование триггеров

- для проверки и исправления данных, нарушающих ограничение целостности;
- для отмены операций модификации, противоречащих ограничениям целостности в виде требований деловых правил;
- для замены операций модификации данных в таблицах вызовом хранимых процедур, проверяющих и поддерживающих деловые правила;
- для протоколирования событий

Пример – автоматическое исправление

```
create trigger BIU_AGENT for AGENT
    before insert  or update position 0
as
begin
    NEW.NAME_AG = UPPER(NEW.NAME_AG);
end
```

Пример – запрет ошибочных действий

```
create trigger AU_EMP for EMPLOYEE
    after update position 0
as
begin
    if (NEW.SALARY < OLD.SALARY) then
            exception ERROR_PAY;
end^
```

Пример –автоматическая нумерация

```
create trigger BI_OPERATION for OPERATION
   before insert
as
begin
   if (NEW.ID is NULL) then
       NEW.ID=GEN_ID(GEN_OPID,1);
end^
```

Изменение другой таблицы

```
create trigger BI_OPER for OPERATION
    before insert position 0
as
begin
 if (NEW.TYPEOP='R')
  update GOODS_WH  T
       set  T.MEASURE  = T.MEASURE – NEW.MEASURE
   where       T.ID_WH        = NEW.ID_WH and
               T.ID_GOODS = NEW.ID_GOODS
end^
```

Журнализация событий

```
create trigger AI_GOODS for GOODS
ACTIVE AFTER INSERT
POSITION 1
as
declare variable tid integer;
begin
  tid = gen_id(log_table_gen,1);
  insert into log_table (id, operation, date_time,
    user_name)
  values (:tid, ’Добавлен новый товар', 'NOW',
                                  user);
end^
```

Триггеры базы данных

- Начиная с версии 2.1
- События базы данных
  - CONNECT
  - DISCONNECT
  - TRANSACTION START
  - TRANSACTION COMMIT
  - TRANSACTION ROLLBACK
- Права – только SYSDBA или владелец базы

```
CREATE TRIGGER  имя
[ACTIVE | INACTIVE]
ON событиеБД
[POSITION number]
AS
[<declarations>]
BEGIN
    [<statements>]
END
```

Пример

```
create trigger tr_connect active
on connect
as
begin
    insert into dblog
            values (current_user,
                        current_timestamp,
                    'connect');
end
```