

INGRES



Smedley's Guide To Database Procedures

What are Database Procedures?

- Code stored in the database
- Executed in the server
- Pre-compiled
- Executable code is a single transaction
- Composed of SQL statements
- Referenced objects must exist at create time
- Associated with database rules

Why Use Database Procedures?

- Performance
- Better than “repeated” SQL
- Reduced client/server communication
- Flexibility
- Easier to change
- Reusability, reduced coding time
- Integrity constraints, business rules

Why Use Database Procedures?

- Controls access to database objects
 - Only procedure owner requires privileges
 - Users only require execute on the procedure
 - Users do not require access to DBP objects
- Security is easier to manage
 - Data only changed via procedures
 - Less privileges
 - Easier still with roles and groups

What Not to Put in a DBP

- Unnecessary and complex code
- Unnecessary Variables
- Transaction control
 - Commit
 - Rollback
- Front-end work
 - Formatting data
 - Retry logic

Database Procedures Execution

- Called - 3GL or 4GL applications
- Executed - Interactive SQL
- Executed - database procedures
- Invoked - database rules

Parameter Passing

- By default Parameter passed BYVALUE
- Parameters can be passed BYREF
- Parameters can be passed
 - IN
 - The default mode
 - Equivalent to BYVALUE
 - OUT and INOUT
 - Equivalent to BYREF

Example in, out, inout

```
create procedure factorial_n
    ( inout n integer, inout fac integer)
as
begin
    if (n <= 0) then
        return;
    endif;
    fac = fac * n;
    n = n - 1;
    execute procedure factorial_n
        ( n = n, fac = fac);
    return;
end;
```

```
execute procedure factorial_n
    (n = 6, fac = 1);
```


Locking Strategy

- Locks taken when procedure starts
- Locks taken on all tables referenced
- No processing unless all locks in place
- Retakes locks after commit or rollback
- When an error rolls back to beginning of procedure

Debugging, Testing and Sizing

- Line numbers
- Sizing
- Message statement

Procedure: get_my_airlines

```
create procedure get_my_airlines(alname nvarchar(60) )
    result row( nchar(3) not null, nchar(3) not null,
    nvarchar(60) not null) as declare iatacode nchar(3);
icaocode nchar(3);
name nvarchar(60);
begin for select al_iatacode, al_icaocode, al_name into
    :iatacode, :icaocode, :name from "smejo01". airline where
    al_name like :alname do return row (:iatacode,
    :icaocode, :name);
endfor;
end;
```

```
2> create procedure get_my_airlines(alname nvarchar(60) ) result
row( nchar(3) not null, nchar(3) not null, nvarchar(60) not null) as
declare iatacode nchar(3);
icaocode nchar(3);
name nvarchar(60);
begin for seelct al_iatacode, al_icaocode, al_name into
:iatacode, :icaocode, :name from "smejo01". airline where al_name like
:alname do return row (:iatacode, :icaocode, :name);
endfor;
end
```

```
E_US0F23 line 4, Syntax error on 'seelct'. The correct syntax is:
FOR subselect DO
other procedure statements
ENDFOR;
(Sun Jun 03 16:43:22 2007)
```

End of Request

=====
Print(SH-F1) File(SH-F2) Help(F1) End(F10)

Start of Output

Line 1

```
1> help procedure get_my_airlines
```

```
Procedure:          get_my_airlines
Owner:             smejo01
Procedure Type:    native
Object type:       user object
Created:           03-jun-2007 16:45:02
```

Procedure Definition:

```
/* 1 */ create procedure get_my_airlines(alname nvarchar(60) )
result row( nchar(3) not null, nchar(3) not null, nvarchar(60) not
null) as declare iatacode nchar(3);
/* 2 */ icaocode nchar(3);
/* 3 */ name nvarchar(60);
/* 4 */ begin for select al_iatacode, al_icaocode, al_name into
:iatacode, :icaocode, :name from "smejo01". airline where al_name like
:alname do return row (:iatacode, :icaocode, :name);
/* 5 */ endfor;
/* 6 */ end
```

```
=====  
Print(SH-F1) File(SH-F2) Help(F1) End(F10)
```

Query Execution Plans

- Created when procedure is
 - Created
 - Loaded into server
- Stored in the Query Storage Facility
- QEP not based on full statistics
 - Max and Min statistics used in optimization

How Big is a QEP?

```
register table imp_qsf_dbp (  
server varchar(64) not null not default is 'SERVER',  
dbp_index integer4 not null not default is 'exp.qsf.qso.dbp.index',  
dbp_name varchar(60) not null not default is 'exp.qsf.qso.dbp.name',  
dbp_owner varchar(24) not null not default is 'exp.qsf.qso.dbp.owner',  
dbp_size integer4 not null not default is 'exp.qsf.qso.dbp.size',  
dbp_dbid integer4 not null not default is 'exp.qsf.qso.dbp.udbid',  
dbp_usage_count integer4 not null not default is 'exp.qsf.qso.dbp.usage'  
)  
as import from 'tables'  
with dbms = ima,  
structure = unique sortkeyed,  
key = (server, dbp_index)
```

Database Procedures in QSF for Server: SMEJ001-XP::/@R2\INGRES\7e0

Name	Owner	DB	Size	Usage
imp_complete_vnode	\$ingres	imadb	2656	2
imp_reset_domain	\$ingres	imadb	2604	2
imp_set_domain	\$ingres	imadb	2740	2
imp_del_domain	\$ingres	imadb	2700	0
get_my_airlines	smejo01	demodb	4579	0
imp_open_server	\$ingres	imadb	3474	0
imp_stop_server	\$ingres	imadb	3474	0
imp_remove_session	\$ingres	imadb	3972	0
imp_shut_server	\$ingres	imadb	3474	0
imp_close_server	\$ingres	imadb	3474	0

Flush(SH-F1) Refresh(SH-F2) End(F10)

Row Producing Procedure

- Must be called from an ESQL program
- Can return zero or more rows
- Row is defined by the Result Row clause
- The value returned in each “column” can be
 - A local variable
 - Parameter of the procedure
 - A constant

Example – Row Producing Procedure

```
Exec sql begin declare;  
    int                empid;  
    int                sales_rank;  
    float              sales_tot;  
    ...  
exec sql end declare;  
    ...  
exec sql execute procedure emp_sales_rank  
    result row (:sales_rank, :empid, :tot_sales);  
exec sql begin;  
    ...  
exec sql end;  
    ...
```

Example – Row Producing Procedure

```
create procedure emp_sales_rank result row (int, int, money) as
declare
    sales_tot          money;
    empid              int;
    sales_rank         int;
begin
    sales_rank = 0;
    for
        select e.empid, sum(s.sales) as sales_sum
        into :empid, :sales_tot
        from employee e, sales s
        where e.job = 'sales'
        and e.empid = s.empid
        group by e.empid
        order by sales_sum desc
    do
        sales_rank = sales_rank + 1;
    return row (:sales_rank, :empid, :tot_sales);
    endfor;
end
```

Error Behaviour

- Procedure is not terminated
- All statements in the procedure up to the point of the error are rolled back
- Error is returned and iierrornumber is set
- Continues execution with the statement following the statement that caused the error
- Parameters passed by reference are not updated
- The error is returned to the application in
 - SQLSTATE
 - SQLCODE
 - errorno

Row Counts and Errors

- `iirowcount`
 - Contains the number of rows affected by the last executed SQL statement
- `iierrornumber`
 - Contains the error number associated with the execution of a database procedure statement.
- Reflect the results of the preceding query
- Beware of resetting the value

Return

- Terminates the procedure
- Can return an integer

Message

- Use the SQL message statement to return messages to users and applications
- Messages from database procedures can be trapped using
 - whenever sqlmessage statement
 - set_sql(messagehandler) statement
- Messages from database procedures can return to the application before the database procedure has finished executing

Raise Error Statement

- Used to describe database errors and violations of business rules
- Generates an error
- The Server responds to this error exactly as it does to any other error
- Returns error number and customised message
 - `raise error errornumber [errortext]`
- `errornumber`
 - Is returned to `sqlerrd(1)`
 - Can be accessed using `inquire_sql(dbmserror)`

The Wrong Way to Error Check

...

```
update emp set ...;
```

```
rcount = irowcount;
```

```
enumber = ierrornumber;
```

...

The Right Way to Error Check

...

```
update emp set ...
```

```
select irowcount, ierrornumber  
into rcount, enumber;
```

...

Another Way to Error Check

```
update xyz . . . ;
```

```
if iierrornumber != 0 then
```

```
    rollback;
```

```
    message 17 'Update failed';
```

```
    return -1 ;
```

```
endif ;
```

The Wrong Way to Error Check

```
update xyz . . . ;  
if ierrornumber != 0 then  
    msg := 'Error updating xyz. '  
        + ' Error is '  
        + varchar(ierrornumber);  
    mno := 1701;  
    message :mno : msg ;  
    return ierrornumber ;  
endif ;
```

The Wrong Way to Error Check

```
if irowcount != 1 then
    msg := 'Bad rowcount updating table xyz'
        + ', expected 1 but got '
        + varchar(irowcount);
    mno := 1702 ;
    message :mno :msg ;
    return iierrornumber ; /* failure */
endif ;
return 0 ; / * success */
```

Error Handling

- Simple is best
- Just check `iierrornumber` and `iirowcount`
- Error check as first statement in a DBP
- Return status only
- Let front end translate, report, display, log

Select Loop

```
execute procedure news_sorter;
```

```
select * from news_sorted;
```

Select Loop

```
create procedure news_sorter
as
declare
    next_id      integer;  /* next_id is the next id in the sorted table */
    brand        integer;
    news_id      integer;
    headline     varchar(100);
    link         varchar(150);
    desc         varchar(100);
    entered      date;
begin
    next_id = 1;
    delete from news;
    for
```


Select Loop

```
select brand, news_id, headline, link, desc, entered
into :brand, :news_id, :headline, :link, :desc, :entered
from news_orig
where deleted = 0
order by brand, news_id

do

insert into news_sorted (news_id, brand, news_orig_id,
                        headline, link, desc, entered)
values (:next_id, :brand, :news_id, :headline, :link, :desc, :entered);
next_id = next_id + 1;

endfor;
end;
```

Session table

```
declare global temporary table session.news_sorted
```

```
  (news_order      integer,  
   brand           integer,  
   news_id        integer,  
   headline       varchar(100),  
   link           varchar (150),  
   desc           varchar( 100),  
   entered        date)
```

```
on commit preserve rows
```

```
with norecovery;
```

```
modify session.news_sorted to btree unique on news_order;
```

```
execute procedure news_sorter(my_news_sorted = session.news_sorted);
```

```
select * from session.news_sorted  
order by news_order;
```

Session table

```
create procedure news_sorter
(my_news_sorted = set of (
  news_order      integer,    /* next_id is the next id in the sorted table */
  brand           integer,
  news_id         integer,
  headline        varchar(100),
  link            varchar(150),
  desc           varchar(100),
  entered        date))
as
declare
  next_id         integer;    /* next_id is the next id in the sorted table */
  w_brand         integer;
  w_news_id       integer;
  w_headline      varchar(100);
  w_link          varchar(150);
  w_desc          varchar(100);
  w_entered       date;
```

Session table

```
begin
  next_id = 1;
  delete from my_news_sorted;
  for
    select brand, news_id, headline, link, desc, entered
    into :w_brand, :w_news_id, :w_headline, :w_link, :w_desc, :w_entered
    from news
    order by brand, news_id
  do
    insert into my_news_sorted
    values (:next_id, :w_brand, :w_news_id, :w_headline, :w_link, :w_desc, :w_entered);
    next_id = next_id + 1;
  endfor;
end;
```

INGRES



Database Rules

Comparison

- Advantages of rules and procedures over constraints and integrities -
 - More complex business rules
 - More control in handling violations
 - More control in handling errors
 - More flexibility in managing changes
 - Better use of resources

Things to avoid

- Multiple rules triggered by same action on a table
- Excessive rules cascading

Error Behaviour for DBP invoked by a rule

- Terminates immediately
- Returns an error
- Statements in the procedure which have been executed are rolled back
- The statement that fired the rule is rolled back
- NOT the same as an explicit rollback
 - just rolls back the statement, not the transactions
- The error is returned to the application in
 - SQLSTATE
 - SQLCODE
 - errorno

Rules

- Set [no]rules statement
- Rules are not fired by
 - Copy
 - Modify
- Set [no]prinrules

Example of Before Rule

```
create rule state_ins before insert into employee
    execute procedure state_ins (
        stcode = state_code,
        stname = state_name);
```

```
create procedure state_ins( in stcode char(2), out stname char(12))
as
begin
    select st_name into :stname from state_codes
        where st_code = :stcode;

    return;
end;
```

Questions & Answers