Stored Procedures Languages – Comparación

Objetivo

Este apunte tiene como objeto describir y comparar las sentencias principales de los diferentes lenguajes de stored procedures para los siguientes motores de BD:

- Informix – SPL Stored Procedure Language
- Oracle – PL/SQL Procedure Language / SQL
- SqlServer – Transact SQL
- DB2 – SQL Procedure Language

Contenido

Introducción a los Stored Procedures ... 3
Que es un Stored Procedure? ... 3
Ventajas de los Stored Procedures ... 3
Desventajas ... 3
Diccionario de Datos Tablas Involucradas ... 4
Creación, borrado y ejecución de Stored Procedures ... 5
Creación de Stored Procedures ... 5
Borrado de Stored Procedures ... 8
Ejecución de Stored Procedures ... 9
Invocación de Stored Procedures desde otro SP ... 10
Manejo de Parámetros de Entrada y Salida. ... 12
Creación de Stored Procedures con Parámetros. ... 12
Ejecución de Stored Procedures con parámetros. ... 14
Retorno de variables de salida ... 16
Ejecución de Stored Procedures desde sentencias SQL. ... 18
Sentencias del Lenguaje de Stored Procedures ... 20
Definición de variables ... 20
Asignación de valores a variables ... 22
Sentencias de Manejo de Bloques y Labels ... 24
Sentencias Condicionales ... 26
Sentencias de Cíclicas ... 28
Ejecución de comandos del Sistema Operativo. ... 30
Manejo de Cursores. ... 31
Otras Sentencias. ... 33
Manejo de Transacciones ... 36
Manejo de excepciones para los Stored Procedures ... 37
Ejemplos de uso de stored procedures con transacciones ... 40
Ejemplos de uso de stored procedures con cursores ... 41

Introducción a los Stored Procedures

(REVISAR LA INTRO PARA OTROS MOTORES, ESTÁ BASADA EN INFORMIX)

Que es un Stored Procedure?

Características:

1. Incluyen sentencias de SQL y sentencias de lenguaje propias. Lenguaje SPL.
    - Pueden contener cualquier Sentencia SQL excepto:
        - CREATE PROCEDURES
        - CREATE DATABASE
        - DATABASE
        - CLOSE DATABASE
2. Son almacenados en la base de Datos
    - Solo las sentencias del lenguaje de programación de Stored Procedure son permitidas además de las de SQL
3. Se guarda la sentencia SQL ya parseada y optimizada
    - Antes de ser almacenada en la base de datos las sentencias SQL son parseadas y optimizadas. Cuando el stored procedure es ejecutado puede que no sea necesario su optimización, en caso contrario se optimiza la sentencia antes de ejecutarse.
    - El stored procedure almacenado es optimizado cuando se ejecuta la instrucción UPDATE STATISTICS adecuada.
4. Disponibles en SE y en ON-Line

Ventajas de los Stored Procedures

- Pueden reducir la complejidad en la programación. Creando SP con las funciones más usadas.
- Pueden ganar perfomance en algunos casos
- Otorgan un nivel de seguridad extra
- Pueden definirse ciertas reglas de negocio independientemente de las aplicaciones.
- Diferentes aplicaciones acceden al mismo código ya compilado y optimizado.
- En un ambiente cliente servidor, no sería necesario tener distribuido el código de la aplicación
- En proyectos donde el código puede ser ejecutados desde diferentes interfaces, Ud. mantiene un solo tipo de código.
- Menor tráfico en el PIPE / SOCKET, no en la cantidad de bytes que viajan sino en los ciclos que debo ejecutar una instrucción.

Desventajas

- Los procedures por primera vez deben ser recuperados desde disco (ojo se mantienen aquello SP de más reciente Uso)
- Deben ser convertidos desde la representación ASCII a código Binario
- La decisión de reoptimizar SI/NO siempre debe ser hecha.

Diccionario de Datos Tablas Involucradas

INFORMIX

1. **SYSPROCEDURES**
   - Aquí se almacena la cabecera del procedure con datos como:
     - El nombre
     - El dueño del procedure
     - El procid
     - Tamaño (en bytes) del Procedure
     - Nro. de Argumentos
2. **SYSPROCBODY**
   - Aquí se almacena el cuerpo del procedure con datos como:
     - El procid
     - el Datakey: Char(1) que nos indica el tipo de dato almacenado en el cuerpo
       - D=Documento
       - T=Codigo Fuente
       - P=P-Code interprete
     - un nro.seq.
     - El cuerpo del Procedure
3. **SYSPROCPLAN**
   - Aquí se almacena el plan de query y la lista de dependencias con datos como:
     - El procid
     - El planid
     - El datakey con valores = D = Lista de dependencias; Q = Plan de Ejecución
     - Fecha de creación
     - Data: Datos compilados del plan
4. **SYSPROCAUTH**
   - Aquí se almacena la lista de usuarios y sus permisos sobre los SP, con datos como:
     - grantor (Otorgador)
     - grantee (A quien se le otorgo el permiso)
     - El procid
     - La autorización: e = Ejecución; E = ejecución. Habilitado para otorgar permisos.

ORACLE

DB2

1. **SYSCAT.PROCEDURES**
   - Aquí se almacena la información acerca de los stored procedures creados en la BD.:

     Buscar datos de la tabla
2. **SYSCAT.PACKAGES**
   - Aquí se almacena la información acerca de los packages creados en la BD.:
   -

Buscar datos de la tabla

3. **SYSCAT.FUNCTIONS**
   - Aquí se almacena la información acerca de las funciones creadas en la BD.:

Buscar datos de la tabla

SQLSERVER

Creación, borrado y ejecución de Stored Procedures

Creación de Stored Procedures

En esta sección se describe la forma de crear un stored procedure para distintos motores de BD.

INFORMIX

Sintaxis:

CREATE PROCEDURE nombre_proc(parametros)

sentencias SPL y/o SQL (Cada sentencia al final con ;)

END PROCEDURE
[document "Este procedimiento tiene por finalidad… "**]**
[with listing in "warning_file.lst"]

"[ ]" Cláusulas opcionales en cada definición se ponen entre corchetes.

- La cláusula **DOCUMENT** sirve para indicar una documentación acerca del procedure que queda almacenada en la BD y no afecta al funcionamiento del mismo.
- La opción **WITH LISTING** in, permite trapear los warnigs que se puedan producir en la compilación del procedure (no se pueden trapear de otra manera).

Para manejo de Comentarios dentro del código se pueden usar para una sola línea se usan dos menos al principio de la línea "-- línea" y para comentar una serie de líneas se pondrán entre llaves " {xxxx}".

Ejemplo

```
CREATE PROCEDURE borra_orden (n_orden INT)
        XXXXXXXXXXXXX
        XXXXXXXXXXXXX
        XXXXXXXXXXXXX
        XXXXXXXXXXXXX
END PROCEDURE
DOCUMENT "Texto de documentación del SP";
```

ORACLE

En el caso de Oracle, es importante aclarar, que además de permitir el uso de procedures, también posee el objeto Package que permite agrupar varios procedures lógicamente, compartiendo variables globales entre ellos de ser necesario.

Sintaxis:

CREATE OR REPLACE PROCEDURE nombre_proc (parámetros de entrada o salida o ambos)
[SPECIFIC nom_var**]**
IS
BEGIN
sentencias SPL y/o SQL (Cada sentencia al final con ;)

END;

Ejemplos:

CREATE OR REPLACE PROCEDURE nombre_proc (n_orden IN INT)
IS
BEGIN
XXXXXXXXXXXXX
XXXXXXXXXXXXX
XXXXXXXXXXXXX
XXXXXXXXXXXXX

END;

CREATE PROCEDURE borra_orden (**IN** n_orden INTEGER)
LANGUAGE SQL
BEGIN
XXXXXXXXXXXXX
XXXXXXXXXXXXX
XXXXXXXXXXXXX
XXXXXXXXXXXXX
END

DB2

Sintaxis:

CREATE PROCEDURE nombre_proc (parámetros de entrada o salida o ambos)
[SPECIFIC nom_var**]**
[LANGUAGE SQL]
BEGIN
sentencias SPL y/o SQL (Cada sentencia al final con ;)

END

"[ ]" Cláusulas opcionales en cada definición se ponen entre corchetes.

La cláusula **SPECIFIC** es una cláusula opcional que definie un nombre único para un procedimiento. DB2 permite tener un procedimiento definido varias veces con

por ejemplo distinta cantidad de parámetros. Para que eso sea posible es necesario especificar un nombre SPECIFIC único para cada versión del mismo procedimiento.

La cláusula **LANGUAGE SQL** identifica al procedimiento como un procedimiento SQL e indica que el cuerpo del procedimiento será especificado en el cuerpo del la sentencia CREATE PROCEDURE.

Los Stored Procedures en DB2 tienen otras cláusulas opcionales posibles de utilizar como DYNAMIC RESULT SETS, MODIFIES SQL DATA, CONTAINS SQL, READS SQL DATA, etc. Para más información consultar la documentación de referencia de IBM.

Para manejo de Comentarios dentro del código se pueden usar para una sola línea se usan dos menos al principio de la línea "-- línea" y para comentar una serie de líneas se pondrán entre "/*" y "*/", ejemplo: " **/***xxxxxxxxx***/**".

Ejemplo 1:

```
CREATE PROCEDURE borra_orden (IN n_orden INTEGER)
LANGUAGE SQL
        BEGIN
        XXXXXXXXXXXXXX
        XXXXXXXXXXXXXX
        XXXXXXXXXXXXXX
        XXXXXXXXXXXXXX
        END
```

Ejemplo 2: Un procedure con dos versiones diferentes

```
CREATE PROCEDURE suma (IN var1   INTEGER, IN var2 INTEGER, OUT var3
INTEGER)
SPECIFIC suma_2
LANGUAGE SQL
        BEGIN
                SET var3 = va1 + var2;
        END


CREATE PROCEDURE suma (IN var1   INTEGER, IN var2 INTEGER, IN var3
INTEGER, OUT var4 INTEGER)
SPECIFIC suma_3
LANGUAGE SQL
        BEGIN
                SET var4 = va1 + var2 + var3;
        END
```

Como se observa las versiones del procedure suma son dos, la primera cuyo nombre específico es suma_2 tiene 3 parámetros y la segunda cuyo nombre específico es suma_3 tiene 4 parámetros.

SQLSERVER

Sintaxis:

```
CREATE PROCEDURE [nombre_proc]
AS
```

sentencias SPL y/o SQL (Cada sentencia al final con ;)
GO

"[ ]" Cláusulas opcionales en cada definición se ponen entre corchetes.

Ejemplo

```
CREATE PROCEDURE [suma]
AS
DECLARE @var1    INTEGER = 15,
DECLARE @var2    INTEGER = 18
        SET @var3 = @va1 + @var2;
GO
```

Borrado de Stored Procedures

INFORMIX

Sintaxis:

DROP PROCEDURE nombre_proc
NOTA: No es posible modificar un procedure. Hay que borrarlo y luego crearlo nuevamente.

Ejemplo:

DROP PROCEDURE borra_orden

ORACLE

Sintaxis:

DROP PROCEDURE nombre_proc

Ejemplo:

DROP PROCEDURE borra_orden

DB2

Sintaxis:

DROP PROCEDURE nombre_proc
NOTA: No es posible modificar un procedure. Hay que borrarlo y luego crearlo nuevamente.

Cuando un procedure tiene varias versiones no se podrá utilizar el DROP PROCEDURE, sino que habrá que usar la siguiente instrucción:

DROP SPECIFIC PROCEDURE nom_specific

ó utilizar la siguiente instrucción

DROP PROCEDURE nom_proc (datatype, datatype …)

Se borraría el procedure indicando con la cantidad de parámetros y tipo de datos la versión del procedure a borrar.

Ejemplo:

DROP PROCEDURE borra_orden

DROP PROCEDURE suma
Este drop fallaría debido a que el procedure suma es ambiguo debido a que tiene dos versiones.

Lo correcto sería realizarlo de cualquiera de estas dos formas:

DROP SPECIFIC PROCEDURE suma_2
Ó
DROP PROCEDURE suma (INTEGER, INTEGER, INTEGER)

SQLSERVER

Sintaxis:

DROP PROCEDURE nombre_proc
NOTA: No es posible modificar un procedure. Hay que borrarlo y luego crearlo nuevamente.

Ejemplo:

DROP PROCEDURE suma

Ejecución de Stored Procedures

Vemos en esta sección la forma de ejecutar un stored procedure para distintos motores de BD.

INFORMIX

Sintaxis:

EXECUTE PROCEDURE [nom_db@nom_server] nombre_proc(param1);

Ejemplos:
EXECUTE PROCEDURE prueba(param1, param2, param3)

EXECUTE PROCEDURE dbase1@servidor22:prueba(1)

1. Es posible llamar a procedures en bases de datos remotas.
2. No se puede usar un sinónimo para el nombre de un procedure
3. No se puede llamar a otros procedures remotos desde un procedure remoto
4. Si se pueden mencionar tablas en servers remotos

ORACLE

Sintaxis:

DECLARE
param1 VARCHAR2(32767);

```
param2 VARCHAR2(32767);
BEGIN
   param1:= valor1;
   param1:= valor2;

   nombre_proc(param1,param2);

END;
```

Ejemplos:

```
DECLARE
param1 VARCHAR2(32767);
param2 VARCHAR2(32767);
BEGIN
   param1:= 8;
   param1:= 20;

   prueba(param1,param2);

END;
```

DB2

Sintaxis:

CALL nombre_proc(param1,…..);

Ejemplos:
CALL suma(param1, param2, param3,?)

Se deberán pasar los valores para los parámetros IN, para los parámetros OUT se deberá poner el carácter ?.

SQLSERVER

Sintaxis:

execute nombre_proc param1, param2

Ejemplos:

execute Suma 15, 13

Invocación de Stored Procedures desde otro SP

Vemos en esta sección la forma de ejecutar un SP desde otro Stored Procedure.

INFORMIX

Sintaxis:

CALL PROCEDURE nombre_proc(param1)
RETURNING var1, var2;

Si el procedure retorna valores los mismos serán asignados a las variables que siguen a la cláusula RETURNING. Dichas variables deberán estar definidas a principio del procedure con la instrucción DEFINE.

Ejemplos:

```
CREATE PROCEDURE otorgar_descuento (p_customer_num integer)
        DEFINE p_order_num integer;

        CALL busca_mayor_orden(p_customer_num)
        RETURNING p_order_num;

        ……

END PROCEDURE;

CREATE PROCEDURE busca_mayor_orden (p_customer_num integer)
        DEFINE p_order_num integer;

        SELECT MAX(order_num)          -- Devuelve sólo una fila
        INTO p_order_num
        FROM orders
        WHERE customer_num =p_customer_num;

        RETURN p_order_num;

END PROCEDURE;
```

ORACLE

Sintaxis:

```
nombre_proc(param1, param2);
```

Ejemplos:

```
suma(15,13);
```

DB2

Sintaxis:

```
CALL nombre_proc(param1,…..);
```

Ejemplos:

```
CREATE PROCEDURE otorgar_descuento (IN p_customer_num integer)
        SPECIFIC otorgar_descuento
        LANGUAGE SQL
        BEGIN
                DECLARE p_order_num INTEGER;

                CALL busca_mayor_orden(p_customer_num, p_order_num);

        ……

        END

CREATE PROCEDURE busca_mayor_orden (IN p_customer_num integer,
                                    OUT p_oder_num INTEGER)
```

```
SPECIFIC busca_mayor_orden
LANGUAGE SQL
BEGIN

    SELECT MAX(order_num)        -- Devuelve sólo una fila
    INTO p_order_num
    FROM orders
    WHERE customer_num =p_customer_num;

END
```

SQLSERVER

Sintaxis:

execute nombre_proc param1, param2

Ejemplo:

```
CREATE PROCEDURE [otorgar_descuento]
    @p_customer_num NUMERIC
    AS

    Execute busca_mayor_orden @p_customer_num
    ……
```

END PROCEDURE;

En este caso se invoca de la misma forma que si se estuviera llamando externamente.

Manejo de Parámetros de Entrada y Salida.

En esta sección veremos la forma de pasar parámetros al stored procedure y de devolver valores.

Creación de Stored Procedures con Parámetros.

INFORMIX

Sintaxis:

```
CREATE PROCEDURE nombre_proc (nombre_param datatype
                                [default 0] , ...)
...
...

END PROCEDURE
```

En la definición del procedure se deberá poner entre paréntesis los parámetros con su nombre y su tipo de datos. (*En otras palabras la definición del parámetro de entrada se realiza entre estos paréntesis)*

De manera opcional se podrá definir el valor default en el caso que el parámetro no sea enviado.

Ejemplos:

```
CREATE PROCEDURE borra_orden (n_orden int default 0)
        ...
        ...
END PROCEDURE


CREATE PROCEDURE modifica_fecha (n_orden INT, f_orden DATE)
        ...
        ...
END PROCEDURE
```

ORACLE

Sintaxis:

```
CREATE OR REPLACE PROCEDURE nombre_proc (param1 IN datatype, param2
OUT datatype)
[SPECIFIC nom_var]
IS
BEGIN
        sentencias SPL y/o SQL (Cada sentencia al final con ;)

END;
```

En la definición del procedure se deberá poner entre paréntesis los parámetros con su nombre y su tipo de datos. (*En otras palabras la definición del parámetro de entrada se realiza entre estos paréntesis)*

De manera opcional se podrá definir el valor default en el caso que el parámetro no sea enviado.

Ejemplos:

```
CREATE OR REPLACE PROCEDURE suma (valor1 IN INT, valor2 IN INT,
resultado OUT INT)
IS
BEGIN
        resultado := valor1 + valor2;
END;
```

DB2

Sintaxis:

```
CREATE PROCEDURE nombre_proc (IN nombre_param datatype,
                              OUT nombre_param datatype,
                              INOUT nombre_param datatype)
LANGUAGE SQL
BEGIN
...
...

END
```

En la definición del procedure se deberá poner entre paréntesis los parámetros de entrada, salida o entrada/salida con su nombre y su tipo de datos. (*En otras palabras la definición del parámetro de entrada/salida se realiza entre estos paréntesis)*

Ejemplos:

```
CREATE PROCEDURE borra_orden (IN n_orden integer)
```

```
LANGUAGE SQL
BEGIN
        ...
        ...
END


CREATE PROCEDURE modifica_fecha (IN n_orden INTEGER, IN    f_orden DATE)
LANGUAGE SQL
BEGIN
        ...
        ...
END

CREATE PROCEDURE consulta_ctacte (IN c_cliente INTEGER, IN    f_movim DATE,
                                                    OUT saldo DECIMAL(12,2))
LANGUAGE SQL
BEGIN
        ...
        ...
END
```

SQLSERVER

Sintaxis:

```
CREATE PROCEDURE [nombre_proc] @param1,
                                        @param2

AS
...
GO
```

Ejemplos:

```
CREATE PROCEDURE [suma]
@var1    INTEGER,
@var2 INTEGER
AS
        SET @var3 = @va1 + @var2;
GO
```

Ejecución de Stored Procedures con parámetros.

INFORMIX

Sintaxis:

EXECUTE PROCEDURE nombre_proc(valor) o

EXECUTE PROCEDURE nombre_proc(nombre_param =valor)

El nombre de parámetro debe ser igual al nombre del parámetro definido en la sentencia CREATE PROCEDURE.

Ejemplos:

```
EXECUTE PROCEDURE borra_orden (1001)

EXECUTE PROCEDURE borra_orden (n_orden =1001)
```

EXECUTE PROCEDURE modifica_fecha **(1001,"23/08/2006")**
...
...
END PROCEDURE

ORACLE

Sintaxis:

```
DECLARE
param1 VARCHAR2(32767);
param2 VARCHAR2(32767);
BEGIN
   param1:= valor1;
   param1:= valor2;

   nombre_proc(param1,param2);

END;
```

Ejemplos:

```
DECLARE
param1 VARCHAR2(32767);
param2 VARCHAR2(32767);
BEGIN
   param1:= 8;
   param1:= 20;

   prueba(param1,param2);

END;
```

DB2

Sintaxis:

CALL nombre_proc(valor) o

Ejemplos:

CALL borra_orden **(1001)**

CALL consulta_ctacte (107,"23/08/2007",?)
El tercer parámetro es de Salida (OUT).

SQLSERVER

Sintaxis:

execute nombre_proc valor1, valor2

Ejemplos:

execute Suma 15, 13

Retorno de variables de salida

INFORMIX

Sintaxis:

CREATE PROCEDURE nom_proc (nom_param datatype [default valor])

RETURNING datatype,datatype;

***DEFINE nombre_var datatype**;*
...
...
RETURN nom_param, nombre_var

END PROCEDURE

En la cláusula RETURNING se deberán declarar tantos tipos de datos como variables distintas retorne el Stored Procedure.
Toda variable que se retornará desde el Stored Procedure se deberá declarar con la instrucción DEFINE.

Sintaxis

DEFINE nombre_var datatype;

Si un parámetro de entrada es a su vez un valor de salida del stored procedure el mismo NO se deberá volver a declarar.

La cláusula RETURN sirve para retornar los valores de las variables que la antecedan, dichas variables deberán estar definidas y sus datatypes deberán estar referenciados en la cláusula RETURNING.
Los valores a retornar pueden ser parámetros de entrada o variables.

Para retornar múltiples valores para las mismas variables se deberá utilizar un cursores.
Ver la sección de definición de cursores.

Ejemplos:

Retorno de una variable.

CREATE PROCEDURE consulta_item(n_orden int default null)
RETURNING integer

***DEFINE n_item SMALLINT**;*
....

RETURN n_item
…

END PROCEDURE

Retorno de un parámetro de entrada y una variable.

CREATE PROCEDURE borra_orden(n_orden int default null)
RETURNING integer, smallint;

***DEFINE n_item SMALLINT**;*
....

RETURN n_orden, n_item

END PROCEDURE

Retorno de dos variables

CREATE PROCEDURE consulta_orden (n_orden int)
RETURNING int,smallint;

DEFINE c_articulo SMALLINT;
DEFINE i_neto DEC(12,2);
...
...
RETURN c_articulo, i_neto;
END PROCEDURE

ORACLE

En el caso de Oracle, para asignar un valor a un parámetro de salida, se realiza de la misma manera que cualquier asignación a variable.

Sintaxis:

param_salida:= valor;

Ejemplo:

resultado := 15;

DB2

Sintaxis:

CREATE PROCEDURE nom_proc (OUT nom_param datatype)
LANGUAGE SQL
BEGIN

...
SET nom_param = valor;

END

Para devolver uno o varios parámetros ó parámetros con tipos de datos distintos de INTEGER, se deberán declarar los mismos con la cláusula OUT ó INOUT entre paréntesis en el CREATE PROCEDURE y deben ser actualizados con un valor antes que se complete la ejecución del mismo.

Existe además la cláusula **RETURN** pero la misma sirve para retornar un único valor INTEGER que sólo es utilizado para indicar si el procedimiento se ejecutó correctamente o no, o para indicar algún estado particular de la ejecución del procedure de acuerdo a una regla de negocio.

Sintaxis:

RETURN {valor entero | variable con datatype INTEGER}

Los valores entre “{ }”

Ejemplo:

Retorno de uno o varios parámetros.

```
CREATE PROCEDURE borra_orden(OUT n_orden INTEGER,
                             OUT n_item SMALLINT)
LANGUAGE SQL
BEGIN
....
        SET n_orden = 9999999;
        SET n_item = 99;
...

END
```

Sólo se requiere que antes de finalizar el procedure se asigne un valor a las variables.

Utilización de la instrucción RETURN

```
CREATE PROCEDURE valida_seccion (IN v_n_empleado int)
LANGUAGE SQL
BEGIN
        DECLARE     v_seccion VARCHAR(5);

        SELECT n_seccion
        INTO v_seccion
        FROM empleados
        WHERE n_empleado = v_n_empleado;


...     IF v_seccion = "XXXXX" THEN
            RETURN 1;
        ELSE
            RETURN -1;
        END IF;
END
```

SQLSERVER

Sintaxis:

```
CREATE PROCEDURE [nombre_proc]
@param1,
@param2
AS
        …
        SELECT @varsalida AS ID
GO
```

Ejemplo:

```
CREATE PROCEDURE [suma]
@var1   INTEGER,
@var2 INTEGER
AS
        SET @var3 = @va1 + @var2;
        SELECT @var3 AS ID
GO
```

Ejecución de Stored Procedures desde sentencias SQL.

Los Stored Procedures pueden ser ejecutados desde sentencias sql de distintas formas.

INFORMIX

Sintaxis:
INVOCACIÓN COMO CAMPO EN UN SELECT

```
SELECT campo, nombre_proc(param)
    FROM nom_tabla
```

El procedure deberá devolver sólo un valor.

INVOCACIÓN COMO CAMPO EN LA CLAUSULA WHERE

```
SELECT campo, campo2
    FROM nom_tabla
WHERE campo3 = nombre_proc(param)
```

El procedure deberá devolver sólo un valor.

Ejemplos:

```
SELECT * from ordenes
WHERE n_orden = consulta_mayor_num();
```

Toma el procedimiento consulta_mayor_num y lo ejecuta. Tener en cuenta que el mismo debe retornar un solo valor.

```
SELECT n_orden, calcula_fecha_fin(f_orden)
FROM ordenes
WHERE n_orden = 114;
```

Por cada orden que devuelva el select, ejecutará el procedure "calcula_fecha_fin" en base a la fecha de la orden listada. Si el Select devuelve 10 filas, el procedure se ejecutará 10 veces uno por cada fila.

ORACLE

En oracle no es posible ejecutar un stored procedure desde un sql, pero si una función, la cual debe retornar un único valor

Sintaxis:
INVOCACIÓN COMO CAMPO EN UN SELECT

```
SELECT campo, nombre_func(param)
    FROM nom_tabla
```

INVOCACIÓN COMO CAMPO EN LA CLAUSULA WHERE

```
SELECT campo, campo2
    FROM nom_tabla
WHERE campo3 = nombre_func(param)
```

Ejemplos:

```
SELECT * from ordenes
WHERE n_orden = consulta_mayor_num();

SELECT n_orden, calcula_fecha_fin(f_orden)
FROM ordenes
```

WHERE n_orden = 114;

DB2

Investigar un poco más

SQLSERVER

No está permitido en sql server.

Sentencias del Lenguaje de Stored Procedures

Definición de variables

INFORMIX

Sintaxis:

DEFINE nombre_var datatype;
DEFINE nombre_var2 LIKE nom_tabla.nom_campo;
DEFINE GLOBAL nombre_glob_var default valor;

VARIABLES GLOBALES:

1. Son variables que pueden ser usadas por distintos procedures ejecutados en una misma sesión. Esta variable guarda su valor y puede ser actualizada y/o utilizada en todos los procedures que la tengan definida.
2. Es obligatorio definirle un valor default a una variable global.
3. El contenido de la variable será almacenado en la sesión en la que se ejecutó cada procedure.

Ejemplos:

Create procedure XX (p_order_num int) → Las variables pasadas son definidas en los parámetros

DEFINE p_order_date date;
DEFINE p_customer-num like orders.customer_num;
DEFINE GLOBAL glob_var int default 1;

...

... Variables que serán usadas dentro del procedure y las que se retornarán

end procedure

Ejemplo de variables globales entre dos procedures ejecutados en una misma sesión.

```
CREATE PROCEDURE proc1()
        DEFINE GLOBAL var_global    int DEFAULT 1.

                LET var_global = var_global + 1
END PROCEDURE

CREATE PROCEDURE proc2()
```

```
        DEFINE GLOBAL var_global int default 2.

                LET var_global = var_global + 1
    END PROCEDURE
```

Si en una misma sesión se ejecutan los procedimientos en este orden:

```
execute procedure proc1()
execute procedure proc2()
```

El resultado final de la variable var_global = 3

Si en una misma sesión se ejecutan los procedimientos en este orden:

```
execute procedure proc2()
execute procedure proc1()
```

El resultado final de la variable var_global = 4

ORACLE

Sintaxis:

```
CREATE OR REPLACE PROCEDURE nombre_proc (parámetros de entrada o
salida o ambos)
IS
DECLARE nombre_var;
BEGIN
        sentencias SPL y/o SQL
END;
```

Ejemplos:

```
CREATE OR REPLACE PROCEDURE ejemplo (n_orden IN INT)
IS
DECLARE nombre_var;
BEGIN

        sentencias SPL y/o SQL
END;
```

En este ejemplo podemos observar 2 maneras de asignarle un valor a una variable, la primera es a través de una asignación directa, la segunda es a través del resultado de una consulta sql.

DB2

Sintaxis:

```
        DECLARE nombre_var datatype;
```

Ejemplos:

```
        Create procedure XX (IN    p_order_num int)
        LANGUAGE SQL
        BEGIN
                DECLARE p_order_date DATE;
        ...


        END
```

SQLSERVER

Sintaxis:

```
CREATE procedure [nombre_proc]
AS
        DECLARE @nombre_var datatype
        ...
GO
```

Ejemplos:

```
CREATE procedure [XX]
        AS
                DECLARE @ p_order_date DATE
                ...
        GO
```

Asignación de valores a variables

En los Stored procedures las variables que no son inicializadas, son tratadas de manera diferentes a las variables con NULL.

Es recomendable inicializar las variables que serán usadas en el procedure.

(Validar esto para otros motores)

INFORMIX

Sintaxis:

```
LET nom_var1= valor1; LET nom_var2= valor2
LET nom_var= FUNCION_MOTOR;
LET nom_var1, nom_var2 = valor1, valor2;
LET nom_var1, nom_var2 = (SELECT campo1, campo2 FROM tabla WHERE
                                campo= valor);
                                -- Este select debería devolver sólo 1 fila
LET nom_var = nom_procedure();
LET nom_var3 = nom_var1||nom_var2;
        -- El doble pipe "||" sirve para concatenar valores de variables.
```

Ejemplos:

```
LET c=10 ; LET d=7
LET p_oder_date = today;
LET a,b = 10, 10+c
LET a,b = (select nro, tot from tab1 where clave=10);
LET a = proc_num1()
LET a = c||d
```

ORACLE

Sintaxis:

```
CREATE OR REPLACE PROCEDURE nombre_proc (parámetros de entrada o
salida o ambos)
IS
DECLARE nombre_var;
BEGIN
        nombre_var := valor;
END;
```

Ejemplos:

```
CREATE OR REPLACE PROCEDURE ejemplo (n_orden IN INT)
IS
DECLARE nombre_var;
BEGIN

        nombre_var := 4;

        SELECT INTO nombre_var FROM tabla1 WHERE campo1 = id AND
rownum < 2;
END;
```

En este ejemplo podemos observar 2 maneras de asignarle un valor a una variable, la primera es a través de una asignación directa, la segunda es a través del resultado de una consulta sql.

DB2

Sintaxis:

```
SET nom_var1= valor1;
  SET nom_var= FUNCION_MOTOR;
SET nom_var3 = nom_var1||nom_var2;
        -- El doble pipe "||" sirve para concatenar valores de variables.
```

Ejemplos:

```
SET c=10 ;
SET a = c||d
```

SQLSERVER

Sintaxis:

```
SET nom_var1= valor1
SET nom_var= FUNCION_MOTOR
SET nom_var3 = nom_var1|nom_var2
        -- El pipe "|" sirve para concatenar valores de variables.
```

Ejemplos:

```
SET c = 10 ;
```

```
SET d = getdate()
SET a = c|d
```

Sentencias de Manejo de Bloques y Labels

INFORMIX

1. Cuando se crea un procedure existe al menos un bloque con un BEGIN y END implícitos.
2. Dentro de las sentencias BEGIN y END las variables allí definidas no modifican variables con igual nombre definidas fuera del bloque primero, ni tienen valor fuera del mismo.
3. Se pueden definir dos variables con el mismo nombre en dos bloques distintos del mismo procedure.

Sintaxis:

BEGIN Inicia Bloque

END Finaliza Bloque

Ejemplos:

```
CREATE PROCEDURE proc1 ()
        RETURNING integer;

                -- Bloque implícito
        DEFINE var1 integer;

        LET var1 = 10

        BEGIN     -- Bloque explícito
                DEFINE var1 integer DEFAULT 1;    Esta es una variable de bloque
                let var1 = 40

        END     -- Fin Bloque explícito

        Aquí el valor de var1 es 10

                -- Fin Bloque implícito
END PROCEDURE
```

ORACLE

En el caso de Oracle, no existen los bloques implícitos.

Sintaxis:

BEGIN Inicia Bloque

END Finaliza Bloque

```
CREATE OR REPLACE PROCEDURE nombre_proc (parámetros de entrada o
salida o ambos)
IS
```

DECLARE nombre_var;
BEGIN

Sentencias pl/Sql

BEGIN
Sentencias Pl/Sql
END;

END;

DB2

Manejo de Bloques

Un procedure debe tener al menos un bloque explícito
En un procedure pueden existir varios bloques.

Sintaxis:
BEGIN Inicia Bloque

END Finaliza Bloque

Manejo de Labels

En un procedure podrían existir varios bloques etiquetados con un Label

Sintaxis:
Nom_Label: BEGIN --Inicia Bloque y Label

END [Nom_Label] --Finaliza Bloque y Label

Ó

Nom_Label: -- Label sin bloque definido, en general se pone al final de procedure

Los

Ejemplos:
CREATE PROCEDURE proc1 ()
LANGUAGE SQL
Label1: BEGIN

Label2: BEGIN

GOTO Fin
…...
END Label2

Fin: -- Label sin bloque explícito
….. sentencias de fin de procedure

END -- Este End cierra el Begin de Label2

SQLSERVER

1. Cuando se crea un procedure existe al menos un bloque con un BEGIN y END implícitos.

Sintaxis:

BEGIN Inicia Bloque

END Finaliza Bloque

Ejemplos:

```
CREATE PROCEDURE proc1 ()

        -- Bloque implícito
        DECLARE @var1 integer;

        @var1 = 10

        BEGIN     -- Bloque explícito
                @var1 = 40

        END     -- Fin Bloque explícito

END PROCEDURE
```

Sentencias Condicionales

INFORMIX

Sintaxis:

```
IF condición1 THEN
        Sentencia1

[ELIF condición2 THEN
        Sentencia2      ]

[ELIF condición3 THEN
        Sentencia3      ]

[ELSE
        Sentencia4      ]

END IF
```

Ejemplo:

```
IF a > b THEN      -- si la condición no se cumple evalúa el próximo ELIF
        .........
ELIF c > d THEN    -- si la condición no se cumple evalúa el próximo ELIF
        .........
ELIF j = u THEN    -- si la condición no se cumple ejecuta la acción el ELSE
```

```
        ........
ELSE            -- si ninguna condición se cumplió ejecuta la acción del ELSE
        ........
END IF
```

EXPRESIONES en UNA SENTENCIA IF

```
IF EXIST(Select num_order from Orders) THEN
                ......
END IF

IF p_total_price > ALL (select total_price from orders)

        ......
END IF

IF p_customer_name MATCHES "A*" THEN
        .....
END IF
```

Las expresiones que se pueden poner son similares a los que se pueden usar en una cláusula WHERE de la sentencia SELECT.

ORACLE

Sintaxis:

```
IF condición1 THEN
        Sentencia1;

ELSIF condición2 THEN
        Sentencia2;

ELSIF condición3 THEN
        Sentencia3;

ELSE
        Sentencia4;

END IF;
```

Ejemplo:

```
IF a > b THEN     -- si la condición no se cumple evalúa el próximo ELIF
        .........
ELSIF c > d THEN    -- si la condición no se cumple evalúa el próximo ELIF
        .........
ELSIF j = u THEN    -- si la condición no se cumple ejecuta la acción el ELSE
        ........
ELSE            -- si ninguna condición se cumplió ejecuta la acción del ELSE
        ........
END IF;
```

DB2

SQLSERVER

Sintaxis:

IF condición1 THEN
Sentencia1

ELSE
Sentencia2

END IF

Ejemplo:

```
IF a > b THEN     -- si la condición no se cumple evalúa el próximo ELIF
        ..........
ELSE              -- si condición no se cumplió ejecuta la acción del ELSE
        ........
END IF
```

EXPRESIONES en UNA SENTENCIA IF

```
IF EXIST(Select num_order from Orders) THEN
                .....
END IF
```

Sentencias de Cíclicas

INFORMIX

Sintaxis:

WHILE Loop

WHILE condición
……
[Exit while]

[Continue While]
END WHILE

Exit While – Cuando esta cláusula es ejecutada dentro de un While el programa abandona el mismo y ejecuta la próxima instrucción siguiente al End While.

Continue While – Cuando esta cláusula es ejecutada dentro de un While el programa abandona vuelve al principio del While para evaluar la condición nuevamente.

FOR Loop

FOR variable = 1 to n [step m]
……
[Exit For]

[Continue For]

END WHILE

Exit For – Cuando esta cláusula es ejecutada dentro de un For el programa abandona el mismo y ejecuta la próxima instrucción siguiente al End For.

Continue For – Cuando esta cláusula es ejecutada dentro de un For el programa abandona vuelve al principio del For para ejecutar la próxima iteración actualizando en el 1 o en valor del step la variable.

Otras expresiones pueden ser

For i IN (val1, val2, …)

For i IN (1 to n, m to w)

For i IN (valor1, valor2, 1 to n)

Ejemplos:

WHILE Loop

Las expresiones son evaluadas antes de realizar el loop.
Let x=1

while x < 100 -- WHILE (1=1) = Infinito hasta el fin natural
insert into contador values(x)
let x=x+1
end while

FOR Loop

Las expresiones son evaluadas luego realizar el loop.

for i=1 to 10 step 3 ó for i in (1,4,6,8,10) ó for i in(1 to 20, 40 to 120)
.....

end for end for end for

Terminando o continuando un Loop:

```
for i=1 to 10 step 3
        if i = 9 and x=20 then
                exit for
        end If
        if i = 5 then
                continue for
        end If

end for
```

ORACLE

Sintaxis:

WHILE condición **LOOP**
……
END LOOP;

Ejemplos:

Las expresiones son evaluadas antes de realizar el loop.
DECLARE x:=1 INT;

WHILE x < 100 **LOOP**
insert into contador values(x);
x:=x+1;
END LOOP;

DB2

SQLSERVER

Sintaxis:

WHILE condición
……
END WHILE

Ejemplos:

Las expresiones son evaluadas antes de realizar el loop.
DECLARE x=1 INT

while x < 100
insert into contador values(x)
set @x=@x+1
end

Ejecución de comandos del Sistema Operativo.

INFORMIX

Sintaxis:

SYSTEM “comando”

Ejemplos:

System “echo “” Fila borrada “” | mail judy”

System “mail -s violation ”|| usr1||” “||usr2

- El motor del online esperará a que termine la ejecución de la sentencia.
- No es posible retornar valores desde un Fyle System.
- Es posible controlar el status de la ejecución del comando. Un valor diferente a cero es Retornado cuando el comando falla. (ver sqlerrorcode) ISAM error.

ORACLE

DB2

SQLSERVER

Manejo de Cursores.

El concepto de cursor está asociado a una sentencia cuyo objetivo es de de recuperar y tratar más de una fila a partir del resultado de un Select asociado a él. El cursor va levantando a estructuras de memoria las filas obtenidas y permite asignarlas a un variables con el fin de operar con ellas.

INFORMIX

Sintaxis:

```
FOREACH     [nom_cursor  FOR]   -- sólo par a cursores de UPDATE
            {SELECT cpo1, cpo2….
               INTO var1, var2
               FROM tab1, tab2,….
               WHERE condiciones y joins     |
             EXECUTE PROCEDURE sproc INTO var1, var2…}

            [EXIT FOREACH]
            [CONTINUE FOREACH]
            [RETURN var1,var2…. ] [WITH RESUME]

END FOREACH
```

Las cláusulas entre corchetes “[ ]”son opcionales.
Las cláusulas entre llaves “{}”y separadas por “|” pueden ser ejecutadas de manera excluyente, o una u otra.

Cuando se tiene que retornar múltiples filas desde el procedimiento se debe usar la opción “WITH RESUME” para el caso de que retorne todo el set de datos. Sino termina el FOREACH en el primer ”RETURN”.

Ejemplos:

```
FOREACH
        SELECT c_cliente
        INTO var_c_cliente
          FROM clientes
        WHERE estado=10

……….

END FOREACH;

FOREACH
        EXECUTE PROCEDURE prueba() INTO aux_num
……..

END FOREACH;
```

```
CREATE PROCEDURE borra_orden ()
        DEFINE num integer default NULL;
        BEGIN WORK;

        FOREACH cur1 FOR      -- cursores de update deben ser
                                 nombrados
                SELECT num
                  INTO num
                FROM customers
                WHERE estado=10

                IF num IS NOT NULL THEN
                        DELETE FROM orders WHERE current of cur1;
                                -- la cláusula CURRENT OF cur1 permite
                                tomar la fila que el cursor está apuntando
                                para modificar o borrar.
                        LET num = NULL;
                END IF;
        END FOREACH;

        COMMIT WORK;
END PROCEDURE
```

Ejemplo de procedure que devuelve múltiples filas con múltiples valores, en el caso que el cursor devuelva 100 filas el procedure irá devolviéndolas de a una a la aplicación.

```
CREATE PROCEDURE get_items ()
        RETURNING integer, char (3),integer;
        DEFINE p_stock_no integer;
        DEFINE p_manu_code char (3);
        DEFINE p_quantity integer;

        FOREACH
                SELECT stock_num, manu_code, quantity
                INTO p_stock_no, p_manu_code, p_quantity
                FROM items

                RETURN p_stock_no,p_manu_code,p_quantity
                WITH RESUME;
        END FOREACH;
END PROCEDURE;
```

Nota: Los puntos y comas deberían ser puestos después de cada sentencia el los SP.

ORACLE

Sintaxis:

```
CURSOR nom_cursor(parámetros) IS
SELECT cpo1, cpo2
FROM tab1, tab2,….
WHERE condiciones y joins
```

```
FOR instancia IN nom_cursor(parámetros) LOOP
    BegIN

        Sentencias pl/sql.

END LOOP;
```

DB2

SQLSERVER

Sintaxis:

```
DECLARE nom_cursor CURSOR FOR
SELECT cpo1, cpo2
FROM tab1, tab2,….
WHERE condiciones y joins
```

Ejemplos:

```
CREATE PROCEDURE get_items ()
        DECLARE @p_stock_no integer
        DEFINE @p_manu_code char (3)
        DEFINE @p_quantity integer

        OPEN nom_cursor
                …….
        END
CLOSE nom_cursor
END PROCEDURE;
```

Otras Sentencias.

INFORMIX

Detectando NOT FOUND condición en el aplicativo

NOTA: *Colocando la cláusula "RETURN" sola, la misma fuerza en la aplicación un NOT FOUND.*

Ejemplo:

```
        FOREACH SELECT order_num
                        into p_order_num from orders
                RETURN p_order_num ;
        d;
        RETURN;   -- Retorna un NOT FOUND en el caso que salga del

                                foreach.
```

En la Aplicación se puede controlar el status. (IF STATUS = NOTFOUND).

Procedimientos Recursivos:

Es un procedimiento que se llama asimismo.

Un ejemplo típico es el de Cálculo del Factorial.

Ejemplo:

```
CREATE PROCEDURE factorial (n int default 1)
        RETURNING int;

        IF n <2 then
                RETURN 1
        END IF;

        RETURN n*factorial(n -1);     -- Se vuelve a invocar el
                                         procedimiento.

END PROCEDURE;
```

No hay límite de numero de call procedures anidados
No hay límite de cursores abiertos
Un nuevo cursor puede ser declarado para cada invocación de cada evento

Debug de Stored Procedures

```
...
...
        LET A=10;
        SET DEBUG FILE TO "/tmp/trace.log";
        TRACE ON;
        ...
        ...
        IF x > 0 THEN
                TRACE "entro en el IF"
        END IF

        TRACE OFF;
```

Por default sale por pantalla, en el caso del ejemplo genera un archivo trace.log en tmp.

Obtención del valor asignado a un campo Serial

```
CREATE PROCEDURE serial_insert()
        DEFINE ser int;

        INSERT INTO orders (order_num,order_date,customer_num)
        VALUES (0,"04/01/93",102);

        LET ser = DBINFO("sqlca.sqlerrd1");

        INSERT INTO items
        (item_num,order_num,stock_num,manu_code,quantity)
        VALUES (1,ser,5,"NRG",20);

END PROCEDURE;
```

Obtención del número de la cantidad de finas procesadas.

El siguiente procedimiento devuelve el número de filas procesadas por una sentencia SELECT,DELETE o UPDATE.

```
CREATE PROCEDURE num_rows()
        RETURNING int;

        DEFINE num_rows int;


        DELETE FROM orders WHERE customer_num = 104;

        LET num_rows = DBINFO("sqlca.sqlerrd2");

        RETURN  num_rows;

END PROCEDURE;
```

ORACLE

DB2

SQLSERVER

Procedimientos Recursivos:

Es un procedimiento que se llama asimismo.
Un ejemplo típico es el de Cálculo del Factorial.

Ejemplo:

```
CREATE PROCEDURE [factorial]
@n integer
AS
DEFINE @v_retorno integer

        IF n <2 then
                SET @v_retorno = 1
        ELSE
                SET @v_retorno = n * execute factorial n-1
        END IF;

GO
```

No hay límite de numero de call procedures anidados
No hay límite de cursores abiertos
Un nuevo cursor puede ser declarado para cada invocación de cada evento

Obtención del valor asignado a un campo Serial

```
CREATE PROCEDURE [serial_insert]
```

```
DEFINE ser int;

INSERT INTO orders (order_num,order_date,customer_num)
VALUES (0,"04/01/93",102)

SET @Id = @@IDENTITY

GO
```

Manejo de Transacciones

En esta sección se describe como es el manejo de transacciones en los lenguajes de programación de Strored Procedure de distintos motores de BD.

INFORMIX

Manejo de excepciones para los Stored Procedures

En la ejecución de un SP es factible que ocurran errores tanto de SPL como de SQL.
Las cláusulas par a manejo de excepciones permite controlar en la aplicación ambos tipos de errores, y en base a lo ocurrido tomar determinadas acciones.

INFORMIX

La sentencia **ON EXCEPTION** puede trapear todos los errores encontrados en el procedimiento. Cuando se produce un error la sentencia ON EXCEPTION es ejecutada y el SP ejecuta las sentencias que estén contenidas en su bloque (ON EXCEPTION / END EXCEPTION.

Se debe colocar después de todas las sentencias DEFINE.

Sintaxis:

```
……
ON EXCEPTION SET var_sql_err,var_ isam_err, va_error_info

        --Aquí va una serie de instrucciones a ejecutarse ante un error.

END EXCEPTION [WITH RESUME];
…..
```

Con la cláusula SET se inicializan los errores de acuerdo a:

- En el primer lugar devuelve el error de SQL. Datatype Int.
- En el segundo lugar trae el error ISAM . Datatype int.
- En el tercer lugar usualmente trae string de la tabla, constraint, column, etc.

La cláusula WITH RESUME permite luego de atrapar el error continuar con la siguiente línea a la que ocurrió el error.
Si la cláusula WITH RESUME no estuviera el procedure se comportará:

- *Si se encuentra en el Loop, While, Foreach. Salta al fin del ciclo y continúa con la próxima interacción.*
- *Si Se encuentra en un bloque (BEGIN, END) la ejecución continúa con la primer sentencia después del END.*
- *Si no está dentro de un bloque el procedure finaliza.*

Manejo de Errores específicos

Es posible atrapar errores específicos, o sea realizar un ON EXCEPTION para ciertos tipos de errores.

Sintaxis:

```
……
ON EXCEPTION IN (lista de nros de error)

        --Aquí va una serie de instrucciones a ejecutarse ante un error ocurrido que esté
          en la lista.
```

```
END EXCEPTION    [WITH RESUME];
…..
```

Devolución de un error atrapado o de un error ficticio.

La cláusula RAISE EXCEPTION permite informar a la aplicación el error ocurrido o generar un código de error artificial retornándolo a la aplicación. Hay que asegurarse de que ese error NO exista como error standard de informix.

```
ON EXCETION SET var_sql_err,var_ isam_err, var_error_info
                ……
                RAISE EXCEPTION var_sql_err,var_isam_err,var_error_info
                ……...
END EXCEPTION;
……..
```

Para manejar un error artificial se puede utilizar el nro de error -746 como error de sql y se devuelve 0 com error de ISAM y para el texto info se puede devolver un texto de hasta 72 caracteres.

Ejemplos:

```
DEFINE sql_err int;
DEFINE isam_err int;
DEFINE error_info char(70);
...
...
ON EXCEPTION SET sql_err, isam_err, error_info
        CALL rutina_error(sql_err, isam_err, error_info)
END EXCEPTION;

ON EXCEPTION IN (-206)
        CALL error_create_table()
END EXCEPTION WITH RESUME

…….
```

Si ocurre un error diferente al -206 “Tabla inexistente”, el procedure ejecutará la rutina de manejo de errores y finalizará.

Si ocurre el error -206, el procedure ejecutará el procedure para creación de la tabla faltante y continuará con la próxima instrucción siguiente a la línea en la que se causó el error.

Ejemplo 1:

```
CREATE PROCEDURE    ej_raise1()
        DEFINE sql_err int;
        DEFINE isam_err int;
        DEFINE error_info char(70);

        ON EXCEPTION SET sql_err, isam_err, error_info
                CALL error_routine(sql_err, isam_err, error_info);
                RAISE EXCEPTION sql_err, isam_err, error_info;
```

```
        END EXCEPTION;
...
END PROCEDURE
```

Al ocurrir un error se ejecutará la rutina de manejo de errores error_routine y luego se informará a la aplicación el error de sql, de ISAM y el texto del error ocurrido.

Ejemplo 2:

```
CREATE PROCEDURE    Alta_orden( p_customer_num INT)
……….

        IF p_error >= 100000 THEN

                CALL error_importe ()

        END IF;

……….
END PROCEDURE

CREATE PROCEDURE    error_importe()

        RAISE EXCEPTION -746, 0, “El importe debe ser menor a 100000”;
...
END PROCEDURE
```

En el ejemplo observamos que el RAISE EXCEPTION es utilizado para crear un error ficticio a partir de una determinada regla de negocio. Si el importe de una orden es mayor o igual a 100000, se invoca a un procedure que devuelve a la aplicación un código de error -746 y el mensaje correspondiente.

ORACLE

Excepciones: las mismas indican las acciones a seguir dentro de un bloque de código si ocurre el evento de la excepción, las mas comunes son por no encontrar filas en un select, por encontrar mas de una; y excepción general.

Sintaxis:

```
BEGIN

……
EXCEPTION
WHEN NO_DATA_FOUND THEN
        -- Acciones a tomar
WHEN TOO_MANY_ROWS THEN
        -- Acciones a tomar
EXCEPTION WHEN OTHERS THEN
        -- Acciones a tomar
END;
```

Nota: Si ingresa por una excepción no sigue evaluando las otras, esto implica que la excepción others debe estar última de haber mas de 1.

Ejemplo:

```
PROCEDURE proc_example
      (pi_parametro1 IN NUMERIC,
       po_parametro2 OUT CHAR) IS

v_existe INT;

BEGIN

v_existe := 0;
-- CUERPO DEL PROCEDURE
BEGIN

SELECT campo1
INTO v_existe
FROM tabla
WHERE tablafk = pi_parametro1

EXCEPTION WHEN NO_DATA_FOUND THEN
         po_parametro2:= 'No hay datos';    -- el select precedente no encontró ninguna fila
WHEN TOO_MANY_ROWS THEN
         po_parametro2:= 'Hay varios datos'; -- el select encontró mas de una fila
EXCEPTION WHEN OTHERS THEN
         po_parametro2:= 'Error'; -- cualquier otro error en el bloque definido
END;

END proc_example;
```

DB2

SQLSERVER

Ejemplos de uso de stored procedures con transacciones

```
PROCEDURE proc_example
      (pi_localidad IN VARCHAR2,
       pi_idantiguo OUT INT,
       po_idlocalidad OUT INT,
       po_error OUT CHAR) IS

BEGIN

po_error := 'N';

SELECT sq_localidad.NextVal
INTO po_idlocalidad
FROM DUAL

INSERT INTO tb_localidades
(idlocalidad, nombre, fechacreacion, flagactivo)
VALUES
(po_idlocalidad, pi_localidad, SYSDATE, 'Y')

UPDATE tb_clientes SET idlocalidad = po_idlocalidad
WHERE idlocalidad = pi_idantiguo;

UPDATE tb_localidades SET flagactivo = 'N', fechabaja = SYSDATE
WHERE idlocalidad = pi_idantiguo;
```

```
COMMIT;

EXCEPTION WHEN OTHERS THEN
        ROLLBACK;
        po_error := 'Y';

END proc_example;
```

Ejemplos de uso de stored procedures con cursores

```
CREATE OR REPLACE PACKAGE pack_example
IS

TYPE returnCursor IS REF CURSOR;

TYPE v_valueId_table_type is table of VARCHAR2(20)
                                index by binary_integer;

CURSOR miCursor IS
      SELECT campo FROM tl_table;

// Retorna un cursor
PROCEDURE proc_example
     (parinid            IN      NUMERIC,
      paroutcursor   OUT   pack_example.returncursor) IS
BEGIN

OPEN miCursor FOR
     SELECT *
     FROM tl_table c
     WHERE parinid = nidtable;

END proc_example;

// Retorna un array usando un cursor

PROCEDURE proc_example2
     (parinid            IN      NUMERIC,
      paroutarray     out      v_valueId_table_type) IS

v_posicion NUMBER(3,0):=0;

BEGIN

FOR instancia IN miCursor () LOOP
          paroutarray(v_posicion):= instancia.campo;
          v_posicion:= v_posicion + 1;

END LOOP;

END proc_example2;

END pack_example;
```