

Introduction aux Procédures Stockées d'Interbase

Ecrit par Etienne Bar, développeur et formateur indépendant et membre de l'équipe de www.developpez.com.

Reproduction, copie, publication sur un autre site Web interdite sans autorisation de l'auteur

Avant Propos.....	2
Introduction : Pourquoi les procédures stockées ?	2
La première procédure stockée.....	3
1) Création de la procédure stockée.....	3
2) Suppression d'une procédure stockée.....	4
3) Utilisation d'une procédure stockée.....	4
Règles de base pour écrire une Procédure Stockée	6
1) Partie Déclarative.....	7
Paramètre d'entrée	7
Paramètre de sortie	7
2) Partie procédurale.....	9
L'affectation.....	9
Les opérations arithmétiques	10
Concaténation de chaînes de caractères.....	11
Utilisation de fonctions.....	11
Transtypage	12
1) Utilisation des variables et des paramètres	12
Instructions disponibles	14
1) L'itération et la condition.	15
Itération avec WHILE...DO.....	15
Condition avec IF ... THEN ...ELSE	16
Renvoi d'informations avec Suspend	17
2) Utilisation de commandes SQL	19
Select.....	19
Update, Insert, Delete.....	20
Mise à jour de données renvoyées par un select	23
Exécution de procédures stockées	24
3) Terminer une procédure stockée avant la dernière instruction.....	24
Fin d'exécution avec EXIT.....	24
Gestion d'erreurs avec la commande Exception.....	24
Documentation officielle sur les procédures stockées	26
En conclusion	26
Annexe 1 : Schéma de la base employee	27

Avant Propos

Le cours qui suit s'applique à Interbase V 6.0.1. Le fonctionnement n'est pas garanti avec Interbase 7.0 et Firebird. Mais connaissant la bonne compatibilité ascendante de ces deux logiciels, je pense que ce qui suit devrait parfaitement fonctionner. Vos remarques à ce sujet sont les bienvenues.

Les exemples de ce cours sont effectués sur la base employee, fournie normalement avec Interbase. Le schéma de cette base est fourni en annexe 1 (Sous Windows avec une installation classique, elle se trouve dans le répertoire C:\Program Files\Borland\InterBase\Examples\Database)

La compréhension de ce tutorial implique de connaître les bases de la programmation (notion de variable, de paramètres, d'itération, de condition...) et celles du sql (allez donc faire un tour sur <http://sqlpro.developpez.com/>)

Je remercie particulièrement Frédéric Huet (Barbibulle) pour toutes les fois où il m'a dépanné et pour sa relecture attentive de ce tutoriel. Sans lui, les procédures stockées seraient restées fort mystérieuses pour moi.

Merci également à Jean-Marc Rabilloud pour ses remarques constructives.

N'hésitez pas à me faire part de vos remarques et commentaires pour améliorer ce cours.

Introduction : Pourquoi les procédures stockées ?

Une procédure stockée est, comme son nom l'indique, une procédure manipulant des données¹ écrite dans un langage procédural souvent spécifique au SGBD qui est stockée dans la base de données.

- Une procédure stockée est plus rapide à utiliser : Elle utilise la puissance du serveur, généralement mieux pourvu que le poste client.
- Elle soulage le trafic réseau puisque tout le traitement se fait sur le serveur. Seule la requête navigue du client vers le serveur et le résultat, si besoin, du serveur vers le client
- C'est un mode de partage des tâches d'un développement. On peut ainsi écrire les code de traitement des données dans la base puis (ou parallèlement) l'IHM dans son langage préféré. L'inconvénient d'écrire du code métier au cœur de la base de données est évidemment que l'application n'est plus indépendante du SGBD mais présente des avantages de performance certains (essayez, vous n'en reviendrez pas).
- Un autre avantage est que la mise à jour d'une procédure stockée permet à tous les clients² de la base de données, quels qu'ils soient (Client-Serveur, Web, etc...), de bénéficier de cette mise à jour. Exemple, j'écris une procédure stockée pour connaître le prix d'un article dans une gestion commerciale. Si les règles changent, je modifie cette procédure stockée et tous les clients de l'application en bénéficient. Si les paramètres d'appel et de sortie de la procédure stockée ne changent pas, mes clients n'ont même pas besoin d'être modifiés.

¹ Il est parfaitement concevable d'imaginer une procédure stockée qui ne manipule pas de données de la base. Mais j'avoue avoir du mal sur les applications pratiques d'une telle procédure stockée.

² Comprendre 'client' au sens d'objet logiciel qui utilise ma base et non comme quelqu'un qui m'achète quelque chose

La première procédure stockée

1) Création de la procédure stockée

Nous désirons connaître le nombre d'employés et le nombre de ceux pour lesquels la colonne phone-ext est renseignée.

Le SQL qui va nous permettre d'écrire ceci est :

```
SELECT COUNT(PHONE_EXT), COUNT(EMP_NO) FROM EMPLOYEE ;
```

Et Interbase, dans sa grande bonté et avec sa célérité habituelle, nous répond :

COUNT	COUNT1
39	42

Si nous voulons stocker ce traitement sous la forme d'une procédure stockée, nous écrivons donc dans ISQL :

```
Set term ^ ;

CREATE PROCEDURE MA_PREMIERE_PS
RETURNS (
    NBRETEL INTEGER,
    NBREEMP INTEGER)
AS
BEGIN

    SELECT COUNT(PHONE_EXT), COUNT(EMP_NO) FROM EMPLOYEE
    INTO
        :NBRETEL,
        :NBREEMP;

    SUSPEND;

END ^

Set term ;^
```

Ce qui précède mérite quelques explications :

```
Set term ^ ;
```

Afin que les points-virgules qui parsèment notre procédure ne soient pas interprétés comme un séparateur de deux instructions SQL , il faut indiquer à Interbase que le séparateur ne sera plus le point virgule mais le ^.

Dans le cas contraire, nous obtiendrons le message :

```
Dynamic SQL Error
SQL error code = -104
Unexpected end of command
```

Indiquant qu'InterBase a quelques difficultés à s'y retrouver.

Nous écrivons ensuite notre procédure stockée. L'écriture doit être vue comme une seule et même instruction SQL, d'où l'intérêt de l'instruction qui précède. La fin de l'instruction se fait par le nouveau séparateur ^.

Enfin, l'instruction

```
Set term ;^
```

Remet le point virgule comme séparateur.

Intéressons nous maintenant à la définition de la PROCÉDURE STOCKÉE proprement dite :

L'instruction

```
CREATE PROCEDURE MA_PREMIERE_PS  
...
```

Entraîne la création d'un nouvel objet dans la base.

A noter que l'instruction la plus courte pour créer une procédure stockée serait :

```
Set term ^ ;  
CREATE PROCEDURE FAIT_RIEN  
AS  
BEGIN  
    EXIT ;  
END ^  
Set term ;^
```

qui crée une procédure stockée qui ne fait... absolument rien.

2) **Suppression d'une procédure stockée**

La suppression d'une procédure stockée se fera par l'instruction :

```
DROP PROCEDURE FAIT_RIEN
```

Instruction qui ne réussira que si la procédure stockée Fait_rien n'est pas utilisée ailleurs dans la base –dans une autre procédure stockée par exemple-, car InterBase conserve dans ces tables système la référence de toutes les procédures stockées (et de tous les autres objets) utilisés

3) **Utilisation d'une procédure stockée**

L'utilisation d'une procédure stockée qui renvoie des valeurs se fait par l'instruction :

```
SELECT Champ1, Champ2, etc... FROM NomdeLaProcédureStockée
```

Ou champ1 et champ2 correspondent à des paramètres de retour de la procédure stockée.

Il est à noter que l'instruction

```
select * from fait_rien
```

```
SQL error code = -84  
procedure FAIT_RIEN does not return any values
```

ce qui s'explique puisque la procédure stockée ne renvoie aucune valeur.

A l'inverse, l'instruction

```
select nbretel, nbreemp from ma_premiere_ps
```

renvoie bien un résultat correct.

Si cette procédure utilise des paramètres, on les spécifiera entre parenthèses, séparés par des virgules

```
select nbretel, nbreemp from ma_premiere_ps(1,2)
```

Pour utiliser une procédure stockée qui fait des traitements mais ne renvoie rien, on pourra utiliser l'instruction

```
Execute procedure NomdelaProcédureStockée
```

Qui exécutera les instructions de la procédure stockée sans rien renvoyer³

Si cette procédure utilise des paramètres, on les spécifiera SANS parenthèses après le nom de la procédure, séparés par des virgules

```
Execute procedure NomdelaProcédureStockée 1,2
```

³ A moins d'utiliser Returning Values

Règles de base pour écrire une Procédure Stockée

- On peut utiliser toutes les instructions de manipulation de données (Select, Insert, Update, Delete...) mais l'utilisation de langage de définition de données (Create, Alter) pour modifier la structure de la base n'est pas possible dans une procédure stockée
- On ne peut pas utiliser de domaines dans les définitions de paramètres et de variables
- La commande SET Generator n'est pas utilisable dans une procédure stockée.
- Les variables doivent toujours être déclarées (soient comme paramètres d'entrée, de sortie ou de variables)
- Les instructions sont séparées par des points-virgules (BEGIN et END ne sont pas des instructions)
- Le code d'une procédure stockée commence par une partie déclarative et est suivi par une partie procédurale située entre un Begin et un End.
- Les commentaires doivent être placés entre /* */. Un commentaire ouvert doit être fermé
- Le retour à la ligne n'est pas une fin d'instruction
- On ne peut pas faire de SQL Dynamique (ce qui, à ce qu'on m'a dit, serait possible dans Firebird 1.5)

Cette partie utilise comme exemple la procédure stockée DEPT_BUDGET (qui se trouve dans la base Employee) et qui est la suivante :

```
CREATE PROCEDURE DEPT_BUDGET (
    DNO CHAR (3))
RETURNS (
    TOT NUMERIC (18, 2))
AS
    DECLARE VARIABLE sumb DECIMAL(12, 2);
    DECLARE VARIABLE rdno CHAR(3);
    DECLARE VARIABLE cnt INTEGER;
BEGIN
    tot = 0;

    SELECT budget FROM department WHERE dept_no = :dno INTO :tot;

    SELECT count(budget) FROM department WHERE head_dept = :dno INTO :cnt;

    IF (cnt = 0) THEN
        SUSPEND;

    FOR SELECT dept_no
        FROM department
        WHERE head_dept = :dno
        INTO :rdno
    DO
        BEGIN
            EXECUTE PROCEDURE dept_budget :rdno RETURNING_VALUES :sumb;
            tot = tot + sumb;
        END

    SUSPEND;
END
```

1) **Partie Déclarative**

Cette partie est appelée « header » dans la documentation officielle d'Interbase

Examinons l'entête de la

```
CREATE PROCEDURE DEPT_BUDGET (  
    DNO CHAR (3))  
RETURNS (  
    TOT NUMERIC (18, 2))  
AS  
    DECLARE VARIABLE sumb DECIMAL(12, 2);  
    DECLARE VARIABLE rdno CHAR(3);  
    DECLARE VARIABLE cnt INTEGER;
```

Cette procédure stockée a un paramètre d'entrée (DNO) défini en CHAR(3) et renvoie la valeur TOT défini en NUMERIC (18,2)

Pour la définition des types de données, je vous renvoie au chapitre «Specifying Datatypes » du « data définition guide » de la documentation d'InterBase.

On a ensuite trois variables qui vont être utilisées dans la partie procédurale

Paramètre d'entrée

Un paramètre d'entrée est une information nécessaire au traitement de la procédure stockée. Par exemple, la procédure stockée cité plus haut prend un numéro de département en paramètre, numéro qu'elle va utiliser pour sélectionner les données nécessaires au calcul.

Paramètre de sortie

Les paramètres de sortie sont les informations que renvoie la procédure stockée à l'appelant après traitement.

A l'inverse d'une fonction ou d'une procédure écrite dans un langage de programmation traditionnel (Pascal, C, Visual Basic...), les paramètres de sortie peuvent être renvoyés plusieurs fois avec des valeurs différentes.

Je vous invite à consulter le paragraphe « Renvoi d'informations avec Suspend » page 17 pour mieux comprendre à quoi correspond et comment les paramètres de sortie dans une procédure stockée.

Il est tout à fait possible d'écrire une procédure stockée qui ne prend pas de paramètres et ne renvoie rien non plus.

```
CREATE PROCEDURE PROC_SANS_PARAMETRES
AS
  DECLARE VARIABLE sumb DECIMAL(12, 2);
  DECLARE VARIABLE rdno CHAR(3);
  DECLARE VARIABLE cnt INTEGER;
```

Si on utilise plusieurs paramètres, ceux ci sont séparés par des virgules.

```
CREATE PROCEDURE BLIN_INTERROGELIVRAISONS (
  TYPO INTEGER,
  SELECTIONARTICLES INTEGER,
  FINANALYSE DATE,
  SITEDEPART VARCHAR (12))
RETURNS (
  RES_RF_COM INTEGER,
  RES_RF_PRT VARCHAR (12),
  RES_RF_SIT VARCHAR (12),
  RES_DATEHEURE TIMESTAMP,
  RES_QTE DECIMAL (9, 3),
  RES_TYPEMVT VARCHAR (5))
AS
```

Toutes les variables sont locales. La notion de variable globale n'existe pas dans InterBase. Le seul moyen de passer des données d'une procédure à une autre est d'utiliser des paramètres ou de stocker ces données dans une table.

2) **Partie procédurale**

Cette partie est appelée « body » dans la documentation officielle d'Interbase

Le corps de la procédure commence toujours par BEGIN et se termine par END.

Entre le BEGIN et le END, les instructions sont séparées par des points-virgules.

On ne mettra jamais de point-virgule derrière un END ou un BEGIN. Les fans de pascal/delphi doivent se méfier.

Dans la même procédure, nous trouvons :

```
BEGIN
  tot = 0;

  SELECT budget FROM department WHERE dept_no = :dno INTO :tot;

  SELECT count(budget) FROM department WHERE head_dept = :dno INTO :cnt;

  IF (cnt = 0) THEN
    SUSPEND;

  FOR SELECT dept_no
    FROM department
    WHERE head_dept = :dno
    INTO :rdno
  DO
    BEGIN
      EXECUTE PROCEDURE dept_budget :rdno RETURNING_VALUES :sumb;
      tot = tot + sumb;
    END

  SUSPEND;
END
```

Pour un premier essai, nous attaquons fort car nous utilisons une procédure récursive.

Nous allons passer en revue quelques instructions élémentaires

L'affectation

L'affectation se fait à l'aide du symbole égal (=)

<Variable ou paramètre> = <Valeur ou Opération>

Les opérations arithmétiques

Pour les 4 opérations, on utilisera les 4 symboles traditionnels +,-,*,/

```
CREATE PROCEDURE TESTDECALCUL
RETURNS (
    TEXTE CHAR (50),
    NOMBRE FLOAT)
AS
BEGIN

    TEXTE = 'Une addition 10 + 10 =' ;
    NOMBRE = 10 + 10 ;
    suspend;

    TEXTE = 'Une soustraction 100 - 10 =' ;
    NOMBRE = 100 - 10 ;
    suspend;

    TEXTE = 'Une multiplication 100 * 10 =' ;
    NOMBRE = 100 * 10 ;
    suspend;

    TEXTE = 'Une autre multiplication 100 - 5 * 10 =' ;
    NOMBRE = 100 - 5 * 10 ;
    suspend;

    TEXTE = 'Une division 100 / 10 =' ;
    NOMBRE = 100 / 10 ;
    suspend;

    TEXTE = 'Une autre division 100 - 5 / 10' ;
    NOMBRE = 100 - 5 / 10 ;
    suspend;

END
```

qui renvoie les valeurs suivantes

TEXTE	NOMBRE
Une addition 10 + 10 =	20.000
Une soustraction 100 - 10 =	90.000
Une multiplication 100 * 10 =	1 000.000
Une autre multiplication 100 - 5 * 10 =	50.000
Une division 100 / 10 =	10.000
Une autre division 100 - 5 / 10	99.500

et qu'en déduisons nous ?

qu'InterBase respecte l'ordre naturel de priorité des opérations (multiplication et division avec addition et soustraction).

Bien sur, l'utilisation des parenthèses est dans ce cas recommandée pour la lecture du code.

L'instruction SUSPEND, pour ceux qui ne l'ont pas encore deviné, permet de renvoyer des données à ce qui (une autre procédure stockée, un programme client, une instruction iSQL) appelle la procédure stockée. Chaque suspend renvoie la valeur des paramètres de sortie au moment de l'appel de l'instruction.

Concaténation de chaînes de caractères

L'opérateur de concaténation dans les SQL d'InterBase est le double | (altgr+6 sur un clavier azerty standard)

```
Chaine1 = 'Developpez'  
Chaine2 = '.com'  
Resultat = Chaine1 || Chaine2
```

Résultat contiendra, vous l'avez deviné, la chaîne 'Developpez.com'

Utilisation de fonctions

En standard, InterBase est extrêmement pauvre en fonctions.

A ma connaissance, les seules fonctions utilisables sont

- CAST pour transtyper une variable dans un autre type est dont la syntaxe est

```
CAST(Variable AS TypeInterBase)
```

Où TypeInterBase désigne un type de données connu d'InterBase (pas un domaine)

Ainsi, Cast(VarEnt as DoublePrecision) transformera la variable VarEnt en réel double précision

- GEN_ID pour utiliser un générateur
- UPPER pour mettre en majuscules une chaîne de caractères.

L'utilisation d'UDF (fonctions externes) est donc fortement recommandée, sinon obligatoire.

Transtypage

InterBase est relativement gentille et les acrobaties (d'aucuns diront les horreurs) du genre qui suit ne l'arrêtent pas

```
CREATE PROCEDURE TRANSTYPAGE
RETURNS (
    TEXTE CHAR (50),
    REEL FLOAT,
    ENTIER INTEGER)
AS

BEGIN
    TEXTE = '10' ;
    REEL = 20;
    ENTIER = TEXTE + REEL ;
    SUSPEND;
END
```

A l'exécution, la procédure stockée renverra :

TEXTE	REEL	ENTIER
10	20.000	30

Par contre, il y a quand même une justice car

```
TEXTE = 'Un texte' ;
REEL = 20;
ENTIER = TEXTE + REEL ;
suspend;
```

renvoie le message d'erreur

```
Conversion error from string "Un texte"
```

Que nous n'avons pas volé.

1) Utilisation des variables et des paramètres

Soit la procédure stockée suivante :

```
CREATE PROCEDURE VARIABLES
RETURNS (
    NBRE INTEGER)
AS

DECLARE VARIABLE DEPT_NO CHAR(3);

BEGIN
    DEPT_NO = '600' ;

    SELECT COUNT(*)
    FROM EMPLOYEE
    WHERE DEPT_NO = DEPT_NO
    INTO
        :NBRE ;

    SUSPEND ;
END
```

Exécutons là, elle nous renvoie la valeur de 42, autrement dit la même chose que

```
Select count(*) from EMPLOYEE
```

Alors que

```
Select count(*) from EMPLOYEE where dept_no = '600'
```

Renvoie la valeur 2

Notez que si nous écrivons

```
Select count(*) from EMPLOYEE where dept_no = dept_no
```

Nous obtenons également 42 : nous cherchons en effet tous les employés qui ont un département égal à leur département, ce qui revient à chercher tous les employés ou encore à faire une recherche qui ne sert strictement à rien !

Alors que si nous écrivons et exécutons :

```
CREATE PROCEDURE VARIABLES
RETURNS (
    NBRE INTEGER)
AS
DECLARE VARIABLE DEPT_NO CHAR(3);

BEGIN
    DEPT_NO = '600';

    SELECT COUNT(*)
    FROM EMPLOYEE
    WHERE DEPT_NO = :DEPT_NO
    INTO
        :NBRE ;

    SUSPEND ;
END
```

Nous obtiendrons bien la valeur 2.

La seule chose qui a changé entre les 2 procédures stockées et le : qui précède le second dept_no dans la condition

```
WHERE DEPT_NO = :DEPT_NO
```

Qui indique que le second dept_no est à considérer comme une variable et non comme la désignation de la colonne dept_no de la table employee.

Il est bien sur évident que cet exemple (pervers) n'est là que pour vous éviter des erreurs cruelles à l'avenir. Ne nommez pas vos variables comme des colonnes de la base et vous vous éviterez bien des ennuis ou alors faites le en toute connaissance de cause.

Il faut noter qu'il est parfaitement possible de modifier dans une procédure stockée des paramètres d'entrée.

```
CREATE PROCEDURE VARIABLES_2 (
    PARAM1 INTEGER,
    PARAM2 INTEGER)
RETURNS (
    NBRE INTEGER)
AS
BEGIN
    PARAM1 = PARAM1 + 1;
    PARAM2 = PARAM2 + 1;

    NBRE = PARAM1 + PARAM2;

    SUSPEND ;
END
```

Instructions disponibles

Le langage est décrit dans le chapitre « Working with stored procedures » du manuel « Data definition Guide ».

A coté des langages évolués que nous avons l'habitude de manipuler, ce langage est pauvre. Ne nous en attristons pas, il n'en sera que plus vite appris.

Nous disposons de :

- Une instruction pour itérer : WHILE ... DO
- Une instruction pour renvoyer des données à l'appelant de la procédure stockée : Suspend
- Une instruction pour quitter la procédure (saute directement au END final) : EXIT
- Une instruction de condition : IF ... THEN ... ELSE
- Une instruction de lecture de données FOR <Instruction Select> DO
- Une instruction de gestion d'erreur WHEN ... DO
- Une instruction pour lever une exception : EXCEPTION <Nom de l'exception>

Et bien sur, des instructions SQL qui suivent :

- Select
- Update
- Delete
- Insert

Nous avons vu plus haut la pauvreté des fonctions d'Interbase.

Les commentaires sont placés entre les symboles /* et */. Il va de soi que ne pas commenter ses procédures ne procure qu'un gain de temps minimal que vous paierez au centuples dans quelque temps (vous ne pourrez pas dire qu'on ne vous a pas prévenu)

```
/* Cette ligne est un commentaire */
```

```
/* Ces deux là  
également */
```

1) *L'itération et la condition.*

Itération avec WHILE...DO

Attention, les itérations qui permettent de traiter successivement les lignes renvoyées par une instruction Select sont traitées avec une instruction FOR (voir un peu plus loin dans la partie 'utilisation de SQL')

La syntaxe de cette instruction est la suivante :

```
WHILE <Condition> DO
  BEGIN
    <Instruction 1> ;
    <Instruction 2> ;
    ...
    <Instruction n> ;
  END

CREATE PROCEDURE ITERATION (
  NOMBRE INTEGER)
RETURNS (
  VALEUR INTEGER)
AS
BEGIN
  VALEUR = 0 ;

  WHILE (VALEUR < NOMBRE) DO
    BEGIN
      VALEUR = VALEUR + 3 ;
    END
  SUSPEND ;
END
```

La procédure qui précède permet par exemple de trouver le premier multiple de 3 supérieur ou égal au nombre passé en paramètre.

Notez que cette procédure stockée pourrait s'écrire ainsi :

```
CREATE PROCEDURE ITERATION (
  NOMBRE INTEGER)
RETURNS (
  VALEUR INTEGER)
AS
BEGIN
  VALEUR = 0 ;

  WHILE (VALEUR < NOMBRE) DO
    VALEUR = VALEUR + 3 ;

  SUSPEND ;
END
```

Car il n'y a qu'une seule instruction dans l'itération. On ne doit utiliser BEGIN et END que lorsqu'on a plusieurs instructions. Les mettre systématiquement est néanmoins une bonne idée car cela permet de mieux comprendre la procédure.

Ainsi, les lignes qui suivent :

```
WHILE (VALEUR < NOMBRE) DO
  BEGIN
    VALEUR = VALEUR + 3;
    VALEUR = VALEUR + 1;
  END
```

Ajoutent 4 à la variable valeur à chaque itération

Alors que celles-ci :

```
WHILE (VALEUR < NOMBRE) DO
  VALEUR = VALEUR + 3;
  VALEUR = VALEUR + 1;
```

Ajoutent 3 à chaque itération puis 1, une seule fois, quand elle est terminée, car seule l'instruction VALEUR = VALEUR + 3 est comprise dans l'itération

Condition avec IF ... THEN ...ELSE

D'usage beaucoup plus fréquent, elle obéit aux mêmes principes d'utilisation de BEGIN et END que la boucle WHILE.

```
/* Calcul du plus grand de 2 nombres */
IF (Param1 > Param2) THEN
  Resultat = Param1 ;
ELSE
  Resultat = Param2 ;
```

Ce qui précède pourra aussi s'écrire ainsi :

```
/* Calcul du plus grand de 2 nombres */
IF (Param1 > Param2) THEN
  BEGIN
    Resultat = Param1 ;
  END
ELSE
  BEGIN
    Resultat = Param2 ;
  END
```

et devra s'écrire obligatoirement ainsi s'il y a plus de 2 instructions entre le if et le else ou après le else

```
/* Calcul du plus grand de 2 nombres */
IF (Param1 > Param2) THEN
  BEGIN
    Resultat = Param1 ;
    Lequel = 'Premier' ;
  END
ELSE
  BEGIN
    Resultat = Param2 ;
    Lequel = 'Second' ;
  END
```

Il est bien sur tout à fait possible d'imbriquer plusieurs IF.

```
IF (:PALIER1 IS NOT NULL) THEN BEGIN /* IL EXISTE UN PRIX
SPÉCIFIQUE */
  IF ((:PALIER3 IS NOT NULL) AND (QTETOTALE >= :PALIER3)) THEN
  BEGIN
    CLN_PRIX = PRIX3;
  END ELSE BEGIN
    IF ((:PALIER2 IS NOT NULL) AND (QTETOTALE >= :PALIER2)) THEN
  BEGIN
    CLN_PRIX = PRIX2;
  END ELSE BEGIN
    CLN_PRIX = PRIX1;
  END
  END
  PRIXTROUVE = -1;
  TYPETARIF = 1 ;
END
```

Renvoi d'informations avec Suspend

L'instruction Suspend renvoie la valeur des variables contenues dans la clause Returns à l'appelant de la procédure stockée.

Ainsi, la procédure stockée qui suit

```
CREATE PROCEDURE AUCARRE
RETURNS (
  CARRE SMALLINT)
AS
  DECLARE VARIABLE NBRE SMALLINT;
/* Cette procédure stockée renvoie le carré des 10 premiers nombres */
BEGIN

  NBRE = 0 ;

  WHILE (NBRE < 10) DO
  BEGIN
    NBRE = NBRE + 1 ;
    CARRE = NBRE * NBRE ;
    SUSPEND ;
  END

END
```

Renvoie

CARRE
1
4
9
16
25
36
49
64
81
100

alors que la procédure qui suit :

```

CREATE PROCEDURE AUCARRE_2
RETURNS (
    NBRE SMALLINT,
    CARRE SMALLINT)
AS

/* Cette procédure stockée renvoie le carré des 10 premiers nombres */
BEGIN

    NBRE = 0 ;

    WHILE (NBRE < 10) DO
        BEGIN
            NBRE = NBRE + 1 ;
            CARRE = NBRE * NBRE ;
            SUSPEND ;
        END
    END
END

```

Renverra ce qui suit :

NBRE	CARRE
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

C'est donc ce qui est présent dans l'entête après l'instruction RETURNS qui conditionne ce qui est renvoyé à l'appelant.

2) Utilisation de commandes SQL

Toutes les commandes SQL d'Interbase (Select, Update, Insert, Delete) sont utilisables dans une procédure stockée.

Les commandes du langage de définition de données (Create, Alter) ne sont pas utilisables. Il n'est donc pas possible d'agir sur la structure de la base par l'intermédiaire d'une procédure stockée.

Select

On différenciera les commandes qui ne renvoient qu'une ligne de données (Singleton Select) par opposition à celle qui en renvoient plusieurs.

La ou les lignes renvoyées par une commande SQL Select doivent être stockées dans des variables préalablement déclarées

Ainsi, la commande

```
SELECT COUNT(*) FROM EMPLOYEE
```

Sera utilisée comme suit pour récupérer la valeur dans la variable NBRE

```
SELECT COUNT(*) FROM EMPLOYEE INTO :NBRE
```

Si le Select renvoie plusieurs lignes, on écrira donc

```
FOR
  SELECT
    first_name,
    last_name
  FROM
    EMPLOYEE
  INTO
    :Prenom,
    :Nom
DO
  BEGIN
    /* Traitement d'une ligne */
  END ;
```

Ainsi, la procédure stockée qui suit renvoie tous les projets affectés à un employé dont le code est passé en paramètre.

```
CREATE PROCEDURE GET_EMP_PROJ (
  EMP_NO SMALLINT)
RETURNS (
  PROJ_ID CHAR (5))
AS
BEGIN
  FOR SELECT proj_id
  FROM employee_project
  WHERE emp_no = :emp_no
  INTO :proj_id
DO
  SUSPEND;
END
```

Procédure que pour ma part je préfère écrire comme suit afin d'éviter une mauvaise surprise ...

```
CREATE PROCEDURE GET_EMP_PROJ (  
    EMP_NO SMALLINT)  
RETURNS (  
    PROJ_ID CHAR (5))  
AS  
BEGIN  
    FOR SELECT proj_id  
        FROM employee_project  
        WHERE emp_no = :emp_no  
        INTO :proj_id  
    DO  
        BEGIN  
            SUSPEND;  
        END  
    END  
END
```

Notez que dans la ligne « ... WHERE emp_no = :emp_no », le second :emp_no fait référence, ainsi que l'atteste les deux points au début de son nom, au paramètre d'entrée emp_no.

Update, Insert, Delete

L'utilisation des commandes Update, Insert, Delete ne présente pas de différence par rapport aux commandes SQL classiques, si ce n'est la possibilité d'utiliser des variables.

Nous allons écrire une procédure stockée qui va mettre à jour ou créer un client dans la table Customer selon que ce dernier existe ou non.

Si le pays du client n'existe pas, on le crée avec 'Euro' comme devise

La recherche de l'existence du client se fait sur le libellé du client (colonne Customer). Par simplification, on considère que deux clients ne peuvent pas avoir le même libellé.

La procédure doit renvoyer le code du client créé ou mis à jour et une rubrique indiquant si le pays a été créé. Les booléens n'étant pas gérés par interbase 6.0, on gère dans un entier avec vrai = -1 et faux = 0.

On est ici en présence d'une utilisation judicieuse d'une procédure stockée car un seul appel permet de créer ou de mettre à jour des données dans deux tables distinctes, ce qui est bien sur impossible à faire avec un seul ordre SQL.

```

CREATE PROCEDURE NOUVEAU_CLIENT (
    NOMCLIENT VARCHAR (25),
    PRENOMNOMCONTACT VARCHAR (15),
    NOMCONTACT VARCHAR (20),
    PAYS VARCHAR (10))
RETURNS (
    CODECLIENT INTEGER,
    PAYSCHREE SMALLINT)
AS
/*
Rédacteur : Etienne Bar
Date de rédaction : 18/03/2004

Description :
    Création d'un client ou mise à jour du nom du contact si
    le client existe.
    Création de la fiche pays si elle n'existe pas avec l'Euro
    comme monnaie par défaut
*/

DECLARE VARIABLE LIBELLE_EXACT_PAYS VARCHAR(10) ;

BEGIN
    /* Corps de la procédure */
    /* Le pays existe t'il ?
    /* Pour la recherche, on met tout en majuscules car InterBase est
sensible
    à la casse */

    PAYS = UPPER(PAYS);

    SELECT COUNTRY
    FROM COUNTRY
    WHERE UPPER(COUNTRY) = UPPER(:PAYS)
    INTO LIBELLE_EXACT_PAYS ;

    /* On récupère le libellé du pays avec la casse correcte
    pour le stocker dans la table client de manière correcte */

    IF (LIBELLE_EXACT_PAYS IS NULL) THEN BEGIN
        /* Création du pays */
        INSERT INTO COUNTRY(COUNTRY, CURRENCY)
        VALUES (:PAYS, 'EURO');
        PAYSCHREE = -1 ;
        LIBELLE_EXACT_PAYS = PAYS ;
    END ELSE BEGIN
        PAYSCHREE = 0;
    END

    /* Recherche du client : on considère qu'on ne peut pas avoir 2 clients
    avec le même libellé. Dans le cas contraire, le code qui suit ne
    fonctionnerait pas */

    SELECT CUST_NO
    FROM CUSTOMER
    WHERE UPPER(CUSTOMER) = UPPER(:NOMCLIENT)
    INTO :CODECLIENT ;

    /* Le client existe, on le met à jour */
    IF (CODECLIENT IS NOT NULL) THEN BEGIN

        UPDATE CUSTOMER
            SET CONTACT_FIRST = :PRENOMNOMCONTACT,

```

```

        CONTACT_LAST = :NOMCONTACT,
        COUNTRY =:LIBELLE_EXACT_PAYS
WHERE (CUST_NO = :CODECLIENT) ;

END ELSE BEGIN /* Le client est à créer */

    /* Ititialisation du code grace à un générateur */
CODECLIENT = GEN_ID(CUST_NO_GEN,1) ;
INSERT INTO CUSTOMER(
        CUST_NO,
        CUSTOMER,
        CONTACT_FIRST,
        CONTACT_LAST,
        COUNTRY)
VALUES(
        :CODECLIENT,
        :NOMCLIENT,
        :PRENOMNOMCONTACT,
        :NOMCONTACT,
        :LIBELLE_EXACT_PAYS
        ) ;

END

/* Il ne faut surtout pas oublier le suspend final
pour renvoyer le résultat de la procédure */
SUSPEND ;

END

```

Mise à jour de données renvoyées par un select

Pour mettre à jour des données, on utilisera bien sur une commande update, comme dans la procédure stockée qui suit et qui permet de remplacer les null par des espaces dans certains champs de la table customer.

```
CREATE PROCEDURE REMPLACE_NULL_PAR_ESPACES
AS

DECLARE VARIABLE CODECLIENT INTEGER;
DECLARE VARIABLE PRENOMCONTACT VARCHAR (15);
DECLARE VARIABLE NOMCONTACT VARCHAR (20);
DECLARE VARIABLE MAJ SMALLINT;

BEGIN

FOR /* Pour chaque ligne de la table customer */
SELECT
    CUST_NO,
    CONTACT_FIRST,
    CONTACT_LAST
FROM
    CUSTOMER
INTO
    :CODECLIENT,
    :PRENOMCONTACT,
    :NOMCONTACT
DO
    BEGIN

        MAJ = 0; /* On ne fera la mise à jour que si nécessaire */
        IF (PRENOMCONTACT IS NULL) THEN BEGIN
            PRENOMCONTACT = ' ';
            MAJ = -1;
        END

        IF (NOMCONTACT IS NULL) THEN BEGIN
            NOMCONTACT = ' ';
            MAJ = -1;
        END

        IF (MAJ = -1) THEN BEGIN
            UPDATE CUSTOMER
                SET CONTACT_FIRST = :PRENOMCONTACT,
                    CONTACT_LAST = :NOMCONTACT
            WHERE (CUST_NO = :CODECLIENT) ;
        END /* (MAJ = -1) */

    END /* Pour chaque ligne de la table customer */

END
```

Exécution de procédures stockées

3) *Terminer une procédure stockée avant la dernière instruction*

Fin d'exécution avec EXIT

L'instruction exit interrompt la procédure en cours.

Gestion d'erreurs avec la commande Exception

Il est possible de lever une exception dans une procédure stockée.

Cette exception doit avoir été préalablement créée à l'aide de l'instruction CREATE EXCEPTION. Comme il s'agit d'une instruction qui a un impact sur la structure de la base, une exception ne peut pas être créée dans une procédure stockée. On la créera donc au préalable.

```
CREATE EXCEPTION REAFFECTE_VENTES 'Cet employé a des ventes enregistrées  
dans la base de données'
```

Et on l'utilisera comme suit dans une procédure stockée (rendons à César ce qui est à César : L'exemple qui suit n'est que la traduction –au niveau des commentaires- de la procédure DELETE_EMPLOYEE placée dans les exemples de la base)

```

CREATE PROCEDURE EFFACE_EMPLOYEE (
    EMP_NUM INTEGER)
AS
    DECLARE VARIABLE any_sales INTEGER;
BEGIN
    any_sales = 0;

    /*
        Si des ventes sont affectées à cet employé,
        on ne peut l'effacer avant d'avoir affecté les ventes à
        un autre employé
    */
    SELECT COUNT(PO_NUMBER)
    FROM sales
    WHERE sales_rep = :emp_num
    INTO :any_sales;

    IF (any_sales > 0) THEN
        BEGIN
            EXCEPTION REAFFECTE_VENTES;
        END

    /*
        Si l'employé est un manager, modifier le département
    */
    UPDATE department
    SET mngr_no = NULL
    WHERE mngr_no = :emp_num;

    /*
        Si l'employé est le chef d'un projet, mettre à jour le projet
    */
    UPDATE PROJECT
    SET team_leader = NULL
    WHERE team_leader = :emp_num;

    /*
        Supprimer l'employé des projets auxquels il participe
    */
    DELETE FROM employee_project
    WHERE emp_no = :emp_num;

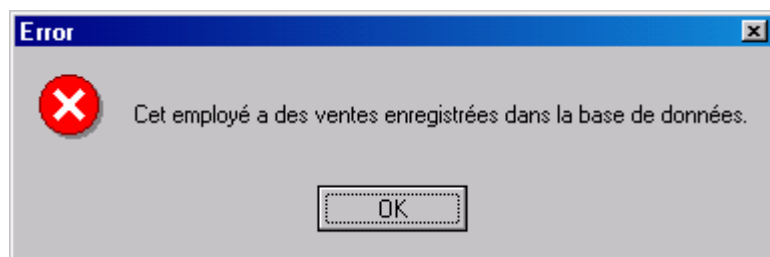
    /*
        Supprimer l'historique de l'employé
    */
    DELETE FROM salary_history
    WHERE emp_no = :emp_num;

    /*
        Supprimer l'employé de la table Employee
    */
    DELETE FROM employee
    WHERE emp_no = :emp_num;

END

```

Et si on exécute cette procédure avec un employé ayant des ventes, on reçoit en réponse :



ce qui est bien l'effet escompté.

Documentation officielle sur les procédures stockées

Dans la documentation officielle Interbase, vous trouverez d'autres informations (en anglais) sur les procédures stockées dans les ouvrages suivants :

Nom de l'ouvrage	Nom et emplacement du fichier correspondant au livre	Chapitre	Pages	Contenu
Data Definition Guide	...\InterBase\Doc\DataDef\DataDef.pdf	Working with stored procedures	135-174	Déclaration et écritures des procédures stockées
Developer'Guide	...\InterBase\Doc\DevGuide\DevGuide.pdf	Working with stored procedures	211-218	Utilisation et gestion des procédures stockées avec les produits Borland (Delphi, C++ Builder, JBuilder)
Embedded SQL Guide	...\InterBase\Doc\EmbedSQL\EmbedSQL.pdf	Working with stored procedures	225-232	Utilisation des procédures stockées en C/C++
Language reference	...\InterBase\Doc\LangRef\LangRef.pdf	Procédures and Triggers	161-182	Manuel de référence

En conclusion

Ce document n'est qu'une introduction aux procédures stockées, il reste un certain nombre de choses à voir.

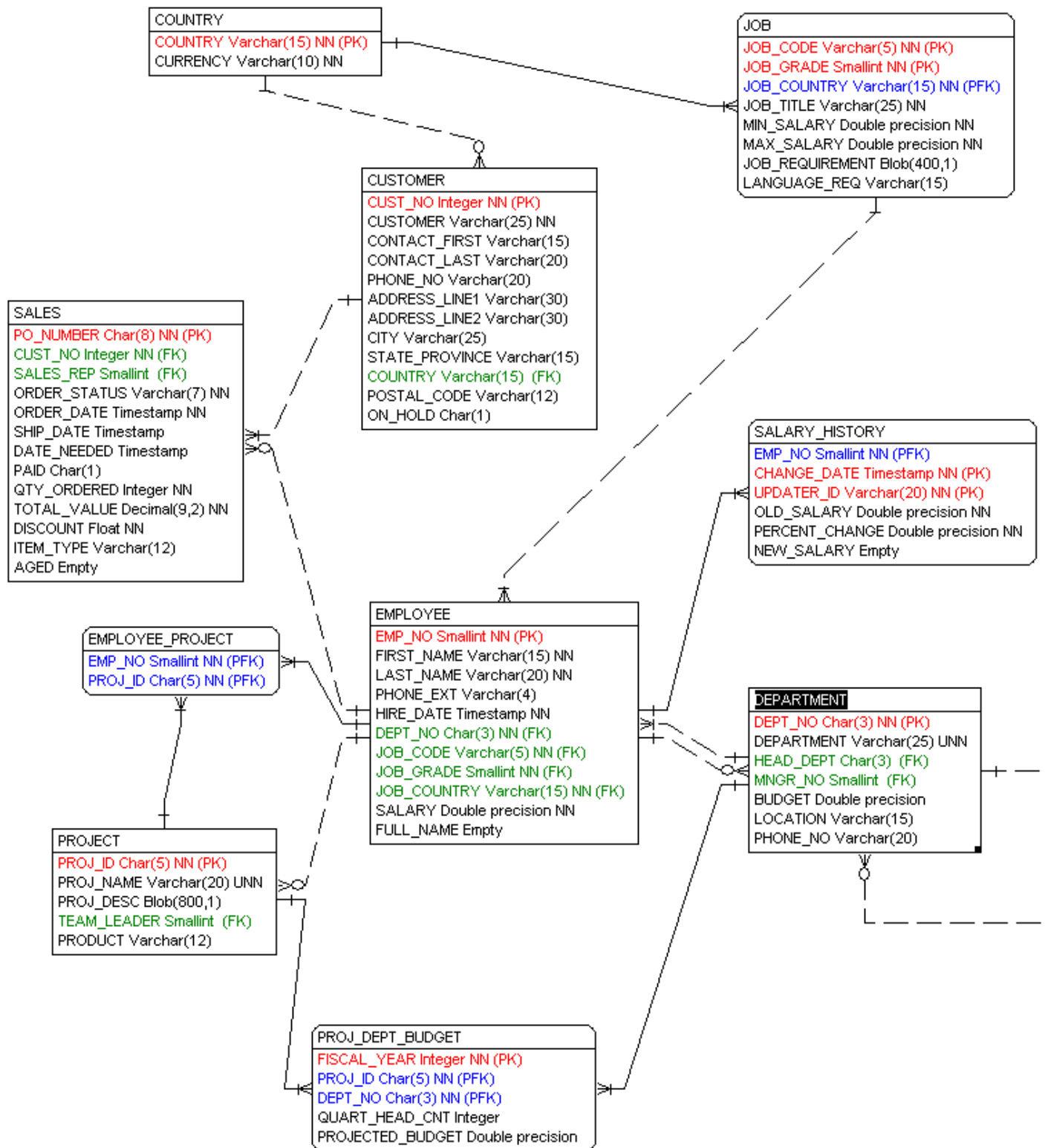
Je n'ai pas par exemple évoqué la gestion des erreurs.

Il va de soi que l'utilisation d'une procédure stockée implique d'avoir les privilèges adéquats pour l'utilisateur qui la lance sur les tables qu'elle manipule.

Enfin, il est à mon avis illusoire de vouloir écrire des procédures stockées conséquentes sans un environnement de développement digne de ce nom. J'utilise pour ma part EMS IB Manager et j'en suis assez satisfait. Il en existe d'autres, je vous laisse faire votre choix.

Pour améliorer votre connaissance des procédures stockées, je vous conseille de regarder celles existant dans la base employée et de bien les comprendre.

Annexe 1 : Schéma de la base employee



(Rétro conception de la base de données effectuée avec Case Studio www.CaseStudio.com)

- NN => Non Null
- PK => Clé primaire
- FK => Clé étrangère