Norme interne de modélisation et développement « Base de Données »

par Frédéric Brouard, alias SQLpro
MVP SQL Server
Expert langage SQL, SGBDR, modélisation de données
Auteur de :

- SQLpro http://sqlpro.developpez.com/
- "SQL", coll. Synthex, avec C. Soutou, Pearson Education 2005
- "SQL" coll. Développement, Campus Press 2001

Enseignant aux Arts & Métiers et à l'ISEN Toulon

Une norme de développement est un outil indispensable pour l'efficacité des équipes de développement. Elle à pour but d'uniformiser la communication entre les différents intervenants susceptibles de travailler à un projet informatique utilisant une base de données relationelle.
Cet article décrit une norme interne d'entreprise dont l'étendue va du modèle conceptuel à l'écriture du code "base de données" en passant par la sructuration des tables et des vues.

Préambule

La présente norme interne s'applique pour la modélisation des données, comme pour le développement des objets de la base de données, ainsi que l'écriture des requêtes et appel de procédures stockées depuis les applications clientes.

Le but de cette standardisation est :

- d'une part, de fournir un cadre commun afin que tous les acteurs de la modélisation et du développement parlent le même « langage »,
- mais aussi, par la simple conception des noms d'objets, introduire des méthodes de gestion et de manipulation globales et génériques afin de faciliter l'administration comme le développement et donc économiser à terme du temps et de l'argent.

Plus que les programmes, les références d'accès aux données (nom des objets en particulier) se doivent d'être fortement normalisées :

- la portabilité d'un SGBDR à l'autre en est facilité ;
- l'accès aux données peut se faire par différentes technologies (client lourd, client léger, serveur d'applications…) en différents langages (VB, C++, Delphi, C#, ASP, PHP…), depuis différentes plateformes (Windows, Linux…) et par différents outils tiers, dont certains n'acceptent pas l'à peu près du respect des standards.

En bref, la norme ne doit pas être vécue comme une contrainte, mais comme une aide.
De fait, pour être appliquée, elle se doit d'être comprise !

0 – Principes généraux

0.1 - Trigramme

Un trigramme est un code de trois lettres unique au sein de son univers. Par exemple un trigramme d'identification d'une table doit être unique au sein de l'ensemble des tables et vues constituant la base de données.
En principe le trigramme sera formé par la première lettre suivi des consonnes les plus sonnantes de l'expression nominale, ou des initiales des mots la composant si l'expression en comporte plusieurs, ou enfin d'un trigramme conventionnel lorsque ce dernier est sans ambiguïté.

Exemples :

```
FCT  factures
LCD  lignes de commande
URL  « uniform resource locator »
CLI  clients
REF  références
```

intérêt le trigramme sert à fournir un tag permettant l'identification rapide de l'objet.

0.2 - Règle générale de formation des noms dans les modèles

Les noms des objets des modèles (entités, associations, tables, vues, colonnes, procédures stockées…) devront répondre des caractéristiques suivantes :

- commencer par une lettre non diacritique (pas d'accent, cédille, etc…) ;
- être composé exclusivement de lettres non diacritique (A à Z) et de chiffres (0 à 9) ainsi que du blanc souligné (underscore), soit 37 symboles ;
- être écrit en majuscule;
- être limité à 128 caractères ;
- être parfaitement compréhensible, notamment en n'utilisant des abréviations que lorsqu'elles sont compréhensibles même par un non initié.

intérêt respect des normes internationales, portabilité. Cette règle émane de la norme ISO/CEI 9075 (toutes révisions depuis 1986) pour la formation des noms dans le langage SQL
La casse en majuscule des requêtes SQL et des procédures permet de distinguer dans le listing des programmes, le code exécuté sur le serveur (en MAJUSCULE) et celui exécuté sur le client (en minuscule) au premier coup d'œil.

1 – Modélisation des données

Il sera fait appel à un outil de modélisation de données dont les caractéristiques sont les suivantes :

- Modélisation conceptuelle des données par schéma entité relation : conception du modèle conceptuel de données (MCD) ;
- Capacité à utiliser le concept de DOMAIN SQL ;
- Capacité à réaliser des sous modèles de façon à découper fonctionnellement l'application (schémas SQL) ;
- Génération d'un modèle physique de données d'après le MCD et ce pour différents SGBDR (Oracle, MS SQL Server, IBM DB2, PostGreSQL, MySQL...), avec gestion des schémas ;
- Rétro ingénierie (de la base au MPD, du MPD au MCD) ;
- Notation des modèles suivants méthodes MERISE, IDEF1X (UML), E/R ;
- Modélisation des héritages, y compris en exclusion mutuelle, avec génération des triggers de validation d'intégrité référentielle ascendante et descendante ;
- Génération des scripts SQL de la base de données ;
- Génération des scripts SQL d'évolution de la base de données (delta) ;
- Édition de la documentation technique.

L'outil préconisé est Power Designer (ex AMC*Designor) de PowerSoft (Sybase).

Toute modélisation de données devra être réalisée exclusivement à l'aide de l'outil.

intérêt raccourcir drastiquement le cycle de modélisation et améliorer très sensiblement la qualité. Permettre la mise à niveau des différentes évolutions de la base de données de manière automatique et transparente.

1.1 – Architecte de données

Une seule personne dans l'entreprise aura la responsabilité de la conception du modèle de données pour une base spécifique. Ses compétences ne pourront être transférées qu'en cas d'indisponibilité. Le but étant de garder la cohérence du modèle.

1.2 – Fichiers des modèles

Tout fichier généré par l'applicatif sera archivé et stocké en lieu sur de même que l'historique des modifications du modèle (modèle archivé) et les scripts différentiels.

Le nom d'un fichier peut indifféremment être écrit en majuscule ou en minuscule, voire une combinaison des deux.

1.2.1 – Nom des fichiers du MCD

Le nom d'un fichier de MCD sera constitué :

- D'un trigramme indiquant le nom de la base (trigramme unique dans le SI de l'entreprise).
- D'une extension spécifique à l'éditeur (en général .mcd)

Exemple :

```
EPS.mcd
```

1.2.2 – Nom des fichiers du MPD

Le nom d'un fichier de MPD sera constitué :

- D'un trigramme indiquant le nom de la base (trigramme unique dans le SI de l'entreprise).
- Un caractère blanc souligné (underscore)
- D'un trigramme indiquant le nom du SGBDR
- D'un code numérique sur 4 positions indiquant la version du SGBDR
- D'une extension spécifique à l'éditeur (en général .mpd)

Les trigrammes d'identification de SGBDR seront choisit parmi :

ORA	Oracle corp., Oracle
SQS	Microsoft SQL Server
MYS	MySQL AB, MySQL
DB2	IBM DB2
PGS	PostGreSQL
ASA	Sybase, Adaptive Server Anywhere (Ex Watcom SQL)
ASE	Sybase, Adaptive Server Entreprise (Ex Sybase SQL Server)

Exemples :

```
EPS_sqs0007.mpd
EPS_SQS2000.mpd
```

Pour le modèle physique de la base de trigramme EPS respectivement pour Microsoft SQL Server 7 et 2000.

1.2.3 – Nom des archives de MPD

Le nom d'un modèle physique archivé sera constitué :

- D'un trigramme indiquant le nom de la base (trigramme unique dans le SI de l'entreprise).
- Un caractère blanc souligné (underscore)
- D'un trigramme indiquant le nom du SGBDR
- D'un code numérique sur 4 positions indiquant la version du SGBDR
- Un caractère blanc souligné (underscore)

- La date de création de l'archive au format ISO court
- D'une extension spécifique à l'éditeur (par exemple .mpa)

Exemples :

```
EPS_SQS2000_20051121.mpa
```

Pour le modèle physique archivé de la base de trigramme EPS, pour Microsoft SQL Server 2000 pour l'archive du 21 novembre 2005.

1.2.4 – Nom des scripts de création de la base de données

Le nom d'un script SQL de création de la base de données sera constitué :

- D'un trigramme indiquant le nom de la base (trigramme unique dans le SI de l'entreprise).
- Un caractère blanc souligné (underscore)
- D'un trigramme indiquant le nom du SGBDR
- D'un code numérique sur 4 positions indiquant la version du SGBDR
- De l'extension .SQL.

Exemples :

```
eps_sqs2000.SQL
```

Pour le script SQL de création de la base de données de trigramme EPS, pour Microsoft SQL Server 2000.

1.2.5 – Nom des scripts différentiels

Le nom des scripts de modification d'une base de données générés par calcul de différence entre le dernier modèle physique archivé et le modèle physique actuel, sera constitué par :

- Un trigramme indiquant le nom de la base (trigramme unique dans le SI de l'entreprise).
- Un caractère blanc souligné (underscore)
- D'un trigramme indiquant le nom du SGBDR
- D'un code numérique sur 4 positions indiquant la version du SGBDR
- Un caractère blanc souligné (underscore)
- La date de création du script au format ISO court
- De l'extension .SQL.

Exemples :

```
EPS_SQS2000_20051121.sql
```

Pour le script de mise à niveau de la base de données calculé d'après le différentiel entre le modèle physique archivé du 21 novembre 2005 et le modèle physique actuel pour la base de trigramme EPS, pour Microsoft SQL Server 2000.

1.3 – Modèle conceptuel

Dans la mesure du possible il sera fait appel à la notion de sous modèles afin de ne pas encombrer un seul modèle avec une quantité d'objet illisible. La notion de sous modèle devra correspondre à un découpage fonctionnel dont l'élément périmétrique est le « schéma » SQL.

Rappelons qu'un schéma SQL est un sous ensemble d'un CATALOG SQL (un CATALOG SQL = une base de données) et permet de gérer la sécurité d'accès aux données de manière globale et générique.

Les objets du modèle (entité, attributs, relation) seront tous annotés (descriptions fonctionnelles).

1.3.1 – Auto documentation de la base

Dans chaque modèle il sera créé différentes entités de nature à documenter la base de données.

L'entité de nom T_S_DATABASE_INFO_DBI contiendra les attributs suivants :

Colonne	Type	Longueur	Oblig.	Validité
DBI_LIBELLE	alphanumérique	128	oui	Clef
DBI_TYPE	alphanumérique	12	oui	'alpha', 'entier', 'réel', 'booléen', 'date', 'heure', 'dateheure'
DBI_VALEUR	alphanumérique	256	oui	

(le nom de cette entité est préfixé T_S_ pour « table système », voir § 1.3.4 pour plus de détail)

Cette entité sera transformée en une table lors de la génération du modèle physique et devra contenir au minimum les informations suivantes :

DBI_LIBELLE	DBI_TYPE	DBI_VALEUR
Nom	alpha	mabase
Date création	date	20051125
SGBDR	alpha	MS SQL Server 2000
MCD	alpha	EPS.mcd
MPD	alpha	EPS_SQS2000.mpd
Script SQL	alpha	EPS_SQS2000.sql
Dernier script SQL diff.	alpha	EPS_SQS2000_20051121.sql
Version base	alpha	1.7
Langue	alpha	Français

Les dates étant saisies au format ISO court (AAAAMMJJ) et les chiffres sans espaces avec le point comme séparateur décimal.

L'information « Version » étant reprise du n° de version du modèle de la base.

A chaque application d'une modification de la base, cette table sera mise à jour, notamment pour ce qui concerne le script différentiel comme pour l'information de version.

intérêt Le but étant d'auto documenter les versions des différentes bases de données en exploitation chez les clients.

L'entité de nom T_S_DATABASE_OBJECT_DBO contiendra les attributs suivants :

Colonne	Type	Longueur	Oblig.	Validité
DBO_NOM	alphanumérique	128	oui	Clef
DBO_TYPE	alphanumérique	16	oui	'table', 'vue', 'colonne', 'procédure', 'fonction'
DBO_LONGUEUR	entier		non	
DBO_LIB_COURT	alphanumérique	32	non	
DBO_LIB_LONG	alphanumérique	256	non	
DBO_DESCRIPTION	alphanumérique	2048	non	

L'entité de nom T_S_DATABASE_LANGUAGE_DBL contiendra les attributs suivants :

Colonne	Type	Longueur	Oblig.	Validité
DBL_LANGUE	alphanumérique	32	oui	Clef

Une association de nom T_J_DATABASE_TRANSLATION_DBT contiendra les attributs suivants :

Colonne	Type	Longueur	Oblig.	Validité
DBT_LIB_COURT	alphanumérique	32	oui	
DBT_LIB_LONG	alphanumérique	256	oui	
DBT_DESCRIPTION	alphanumérique	2048		

et sera liée aux entités T_S_DATABASE_OBJECT_DBO et T_S_DATABASE_LANGUAGE_DBL afin d'assurer la traduction de tous les libellés d'objets de la base dans les langues de fonctionnement de l'application.

intérêt Le but étant de permettre une auto traduction dans les différentes langues dans lesquelles l'application devra être utilisée.

L'entité de nom T_S_DATABASE_ADMIN_DBA contiendra les attributs suivants :

Colonne	Type	Longueur	Oblig.	Validité
DBA_ID	autoincrément			
DBA_DATEHEURE	horodatage			par défaut dateheure courante
DBA_NOM	alphanumérique	32		nom, prénom utilisateur

DBA_NATURE	alphanumérique	256		
DBA_COMMANDE	alphanumérique	7750		

Le but étant de tracer toute action de maintenance entreprise sur le serveur.

1.3.2 – Domaines

Il sera fait appel systématiquement à la notion de « domaine » SQL. Rappelons qu'un DOMAIN est un type SQL pourvu d'une éventuelle valeur par défaut, d'éventuelles règles de validation et d'une éventuelle collation si le type est alphanumérique.

intérêt uniformiser et standardiser les structures de données en vu de globaliser leurs traitements. Par exemple tout attribut basé sur un nom de personne (nom de client, nom d'utilisateur…) sera formé à partir du domaine D_NOM qui pourra être un alphanumérique de longueur 32. En cas de modification des spécifications, la nouvelle définition du domaine D_NOM bénéficiera à l'ensemble des attributs qui repose dessus.

Le nom d'un domaine sera composé en majuscule comme suit :

- la lettre D
- un blanc souligné (underscore)
- une lettre parmi : N pour numérique (entier, réel, décimal…), A pour alphanumérique, T pour temporel (date, heure, horodatage), B pour binaire.
- un blanc souligné (underscore)
- une expression significative

Exemples :

```
D_N_POURCENT      pourcentage numérique avec contrôle de validité borné
                  entre 0.0 et 100.0
D_A_ADRESSE       ligne d'adresse de 38 caractères (norme La Poste)
D_A_ADRIP         adresse IP de 15 caractères constitué par masque
                  nnn.nnn.nnn.nnn
```

Les domaines suivants sont créés de manière systématique :

D_N_ID	entier auto incrémenté (32 ou 64 bits suivant OS)
D_B_GUID	GUID (identifiant générique universel)
D_A_REF_CODE	code (unique) dans une entité référence (alpha fixe, taille 8)
D_A_REF_LIB	libelle dans une entité de référence (alpha variable, taille 64)
D_N_RANG	entier (unique) d'ordonnancement dans une entité
D_B_REF_BASE	booléen dans une entité référence indiquant si la ligne est modifiable par défaut non/0
D_N_MOIS	entier court de 1 à 12 (énumération des mois)
D_N_JR_MOIS	entier court de 1 à 32 (énumération des jours du mois)
D_N_JR_SMN	entier court de 1 à 7 (énumération des jours de semaine)
D_N_JR_AN	entier court de 1 à 365 (énumération des jours de l'année)
D_N_SMN	entier court de 1 à 532 (énumération des semaines ISO)
D_N_TRM	entier court de 1 à 4 (énumération des trimestres)
D_N_SMR	entier court de 1 à 2 (énumération des semestres)
D_T_DATE	date
D_T_HEURE	heure

D_T_HORODATA	horodatage
D_N_N	entier 32 bits positif (> 0) (ensemble N des entiers naturels)
D_N_R_POS	réel positif (>= 0)
D_N_D_POS	décimal positif (>= 0)
D_N_PRIX	décimal 16 chiffres, 2 décimales positif (>= 0)
D_N_S_POS	entier 16 bits positif (>= 0)
D_N_T_POS	entier 8 bits positif (>= 0)
D_N_POURCENT	réel positif [0.0 ... 100.0]
D_N_Z	entier 32 bits positif (>= 0) (ensemble Z des entiers zéro inclus)

intérêt l'utilisation de domaines, et en particulier de ceux dotés de contraintes de validité, minimise l'espace mémoire du fait que le cache de procédure, et donc le code des contraintes, est partagé

1.3.2 - Types d'entités

Les différents types d'entité sont les suivantes :

- entité générale (entité).
  `Exemple : les clients`
- entité de référence.
  `Exemple : sexe.`
- entité de référence externe.
  `Exemple : code postal`
- entité héritée.
  `Exemple : voiture héritée de véhicule`
- entité système.
  `Exemple : utilisateurs de l'application`
- entité générique.
  `Exemple : nombres`

intérêt Sérialiser la gestion des entités et donc des tables qui seront générées d'après ces entités, de manière à permettre des traitements d'ensemble.

1.3.4 – Noms des entités

Un nom d'entité est composé comme suit :

- la lettre T ;
- un blanc souligné (underscore) ;
- une lettre parmi :
  - **E** pour entité,
  - **R** pour référence,
  - **S** pour système,
  - **X** pour référence externe,
  - **H** pour héritée,
  - **G** pour générique,
  - **A** pour administrative,
  - **H** pour historique ;

- un blanc souligné (underscore) ;
- un nom explicite jamais pluralisé, comprenant éventuellement des blancs soulignés ;
- un blanc souligné (underscore) ;
- le trigramme de l'entité.

Exemples :

```
T_E_CLIENT_CLI              entité clients
T_R_SEXE_SEX                entité de référence sexe
T_X_CODE_POSTAL_CPL         entité de référence externe code postal
T_H_VOITURE_VTR             entité héritée voiture
T_S_UTILISATEUR_USR         entité système utilisateurs
```

1.3.5 – Noms des associations

Un nom d'association est composé comme suit :

- la lettre t
- un blanc souligné (underscore)
- la lettre j (pour jointure)
- un blanc souligné (underscore)
- un verbe explicite jamais pluralisé, comprenant éventuellement des blancs soulignés
- un blanc souligné (underscore)
- le trigramme de l'association

nota l'ensemble formé par les trigrammes d'entité et d'association ne doit pas introduire de doublons (unicité). Autrement dit, un même trigramme ne peut servir à une entité et une association.

Exemples :

```
T_J_CONDUIT_CDT         association conduit (par exemple entre client et
                        voiture)
T_J_HABITE_HBT          association habite (par exemple entre utilisateur
                        et code postal)
```

1.3.6 – Noms des attributs

Le nom d'un attribut d'une entité ou d'une association doit être composé en majuscule comme suit :

- le trigramme de l'entité ou de l'association ;
- un blanc souligné ;
- un nom ou une expression explicite, si besoin complété par une indication du type ou de la nature des données.

nota les identifiants de type « compteurs » seront notés ID. Les identifiants de type « identifiant unique » (GUID) seront notés GUID.

Exemples :

CLI_ID	attribut identifiant auto incrémenté de l'entité client
CLI_NOM	attribut nom de l'entité client
CMD_CREATION_DATE	attribut date de création de l'entité commande
LCD_REMISE_POURCENT	attribut pourcentage de remise de l'entité ligne de commande

1.3.7 – Forme des entités de référence

Les entités de référence seront toujours modélisées de la même manière :

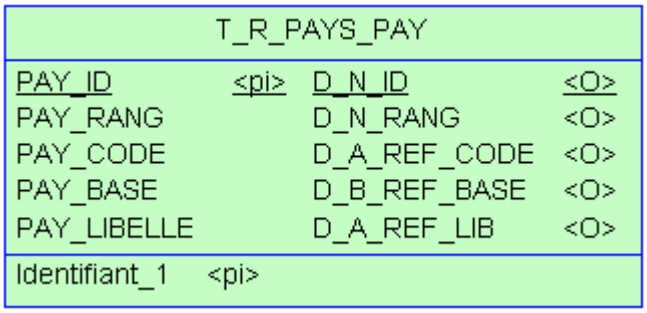
rang	nom	domaine	type	
1	xxx_ID	D_N_ID	entier	auto incrémenté, clef
2	xxx_RANG	D_N_RANG	entier	> = 0
3	xxx_CODE	D_A_REF_CODE	alpha fixe 8	
4	xxx_BASE	D_B_REF_BASE	booléen	par défaut non/0
5	xxx_LIBELLE	D_A_REF_LIB	alpha variable 64	

ou xxx est en fait le trigramme de l'entité.

Des attributs supplémentaires peuvent apparaître après l'attribut situé en 5e position.

intérêt le but de cette uniformisation est de pouvoir opérer globalement sur l'ensemble des références constitué par l'union des entités auxquelles on ajoute le trigramme afin de les distinguer. Par exemple on implémentera côté client un tableau de l'ensemble des valeurs de référence afin d'éviter des allers-retours inutiles entre serveurs et clients.

Exemple :

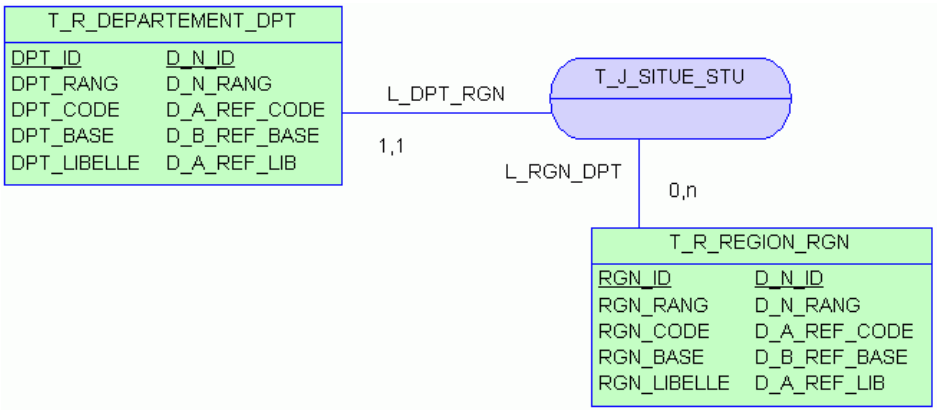


1.3.8 – Nom des liens

Les liens sont les éléments de jonction entre entité et association. Ils se représentent généralement sous forme de traits dans les différentes notations du MCD.

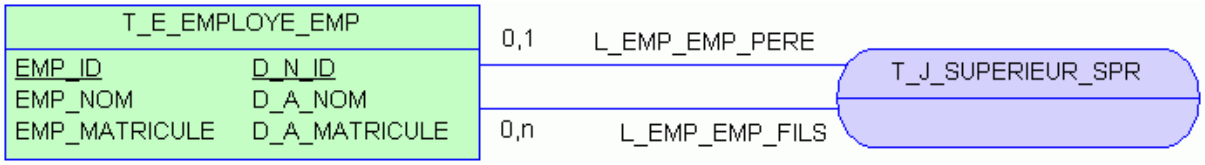
Dans la mesure du possible, le nom d'un lien sera préfixé par la lettre L et reprendra les trigrammes des entités auquel il est lié.

Exemple :



Dans le cas d'une auto relation les liens seront suffixés « FILS » et « PERE ».

Exemple :



intérêt les relations seront traduites par l'outil de modélisation parfois en contraintes et parfois en index. Il est plus facile de savoir quoi faire avec un message d'erreur explicite comme « violation de la contrainte L_DPT_RGN » que « violation de la contrainte C945876468 ».

1.3.9 – Choix des types

L'un des gages de réussite des bases de données relationnelles performantes repose sur le choix judicieux des types de données. Par conséquent on doit apporter le maximum de soin à étudier chaque attribut pour en définir le type en adéquation avec sa longueur.

Voici quelques règles à respecter :

- étudier les normes et les standards internationaux et respecter les formats à chaque fois que cela est possible ;
- utiliser des formats de taille fixe pour les informations de petite tailles, souvent sollicitées en recherche ou fréquemment mises à jour ;
- utiliser des formats de taille variable pour des informations de grande tailles, peu sollicitées en recherche ou rarement mises à jour ;
- utiliser le type le plus précis possible et non un type « fourre-tout »
- utiliser un type numérique exact pour les données comptable (afin d'éviter les erreurs dû aux écarts des arrondis)
- utiliser un type numérique approché pour les données physiques
- éviter d'utiliser des BLOB lorsqu'il est possible de les externaliser, sauf si ces BLOB doivent réellement être exploitées en manipulation de données

Exemple :

ADR_LIGNE	alpha fixe 38 (standard « La Poste » pour les lignes d'adresse)
PRD_REF	alpha fixe 8 (une référence de produit est fréquemment recherchée)
EMP_COMMENTAIRES	alpha variable 4096
PRD_PRIX	numérique longueur 16, 2 décimale (un prix est un élément comptable)
PRD_QUANTITE	réel (une quantité comme du carburant à la pompe, doit pouvoir être exprimée par fraction de litres).

Attention : pour certaines données numérique une précision de mesure doit être apportée. Soit la mesure est immuable et doit apparaître dans le nom de l'attribut, soit la mesure est variable et on doit ajouter un attribut qui complète la mesure.

Exemples :

TCH_DUREE_H_DECIMALE	durée d'une tâche en heures décimale
PRD_QTE	quantité
PRD_QTE_UNITE	unité relative à l'attribut quantité

1.3.10 - Contraintes

Ne devront figurer dans les contraintes d'attributs du modèle que celles :

- obligeant une spécification de valeur ;
- obligeant l'unicité ;
- vérifiant une limite de valeurs (max, min, max + min).

Toute autre contrainte de validité telle que celles qui sont calculée ou encore nécessitent l'usage d'un trigger seront décrite en dehors du modèle.

1.3.11 – Identifiant

Par définition, l'identifiant d'une entité est l'élément primordial de la qualité des relations entre les différentes entités. Pour cela, il se doit d'être mûrement réfléchi et notamment obéir dans la mesure du possible aux caractéristiques suivantes :

- **concision** : l'identifiant doit être le plus court possible (plus court est-il, plus rapide seront les traitements)
- **stabilité** : la valeur de l'identifiant ne doit jamais changer au cours de la vie de la base de données (des changements de valeurs d'identifiant sont de nature à se propager dans le système d'information et peuvent conduire à des paradoxes temporels).
- **dispersion** : la plage de valeur de l'identifiant doit être la plus large possible (l'identifiant doit permettre la manipulation d'entité dont la cardinalité peut être très importante : plusieurs millions d'occurrences).
- **indépendance** : la valeur de l'identifiant ne doit avoir aucun sens sémantique dans le système d'information (un identifiant informatif suppose que l'on connaisse d'avance l'information ce qui n'est pas toujours le cas. De plus l'information doit pouvoir être modifiable ce qui n'est pas bon pour un identifiant).

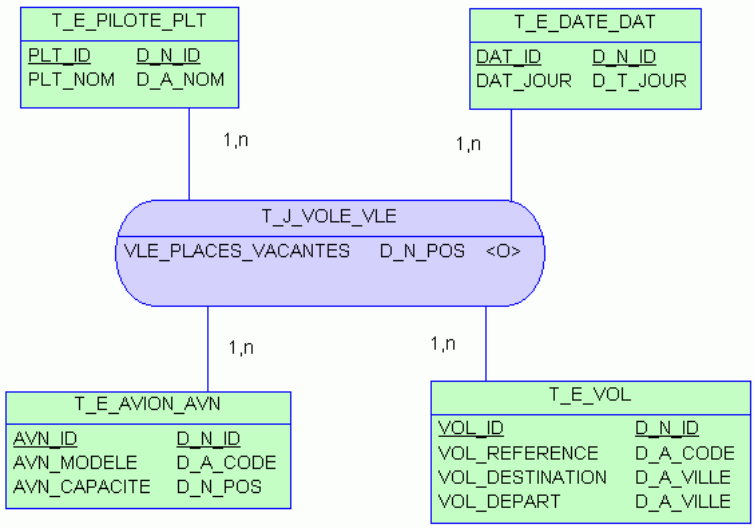
L'identifiant étant un mécanisme nécessaire à l'élaboration des relations entre les informations de la base, il convient qu'il ne soit pas dépendant des informations à modéliser.

Chaque fois que cela est possible on optera pour un entier auto incrémenté dont la longueur sera calquée sur la longueur du mot du processeur dans l'environnement de l'OS. Le type de données d'un tel identifiant sera spécifié par un domaine de nom D_N_ID (voir 1.3.2).

1.3.12 – Associations multijointurées

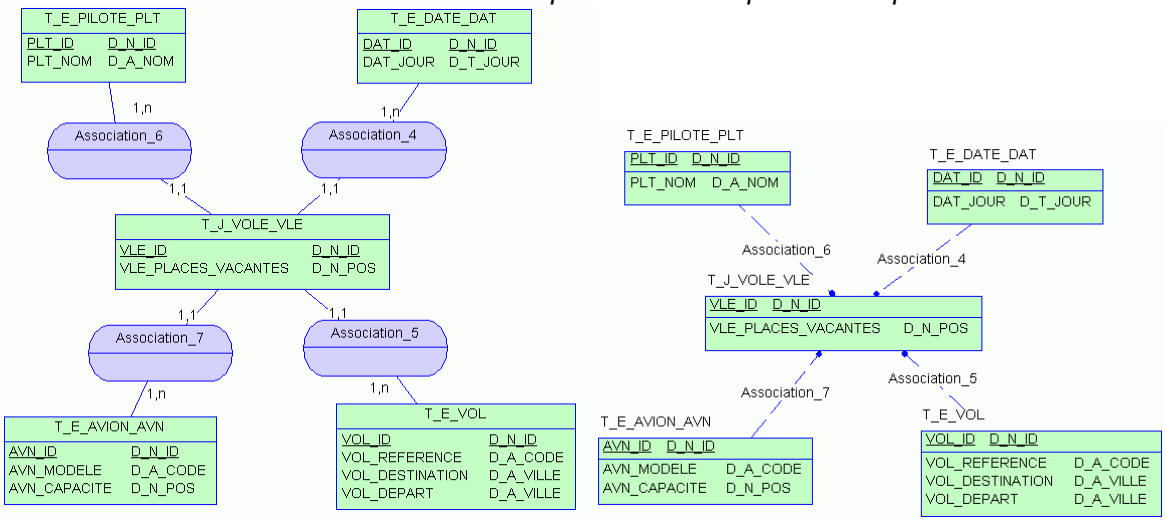
Lorsque l'association comportera ne nombreux liens avec différentes entités, il sera parfois judicieux de transformer cette association en nouvelle entité et remplacer les liens en association de cardinalité 1.1 du côté de l'entité ainsi nouvellement crée. On évitera ainsi la transformation de l'association en table pourvue d'une clef composite peu efficace.

Exemple (schéma entité relation, notation Merise) **:**



Ce modèle conceptuel va générer une clef composée de 4 colonnes lors de la transformation de l'association T_J_VOLE_VLE en table.

Mieux vaut lui substituer le modèle conceptuel sémantiquement équivalent :



Notation Merise et IDEF1X (UML)

intérêt Une clef composite nécessite des efforts importants à l'insertion et une structure de l'index sous jacent particulier. Les opérations d'insertion et de maintenance d'index sont plus faciles lorsque la clef est mono colonne, quitte à ce que plusieurs index complémentaires soient créés.

1.3.13 – Entité génériques

Il sera créé des entités génériques destinées à être utilisées par de multiples tables. Ces entités pourront éventuellement être liées ou non pour leur utilisation.

L'entité nombre, de nom **T_G_NUM** (numérotation) : comportera les colonnes NUM_NUM et NUM_ALEA. La première colonne comportera les nombre de 0 à n (n étant une limite que l'on se fixera). La seconde colonne étant un nombre aléatoire unique pris dans l'intervalle 0, n.

L'entité planning, de nom **T_G_PLN** : comportera les colonnes :

Attribut	**Domaine**
PLN_DATE	D_T_DATE
PLN_JOUR_MOIS	D_N_JR_MOIS
PLN_MOIS	D_N_MOIS
PLN_AN	D_N_I_POS
PLN_JOUR_SEMAINE	D_N_JR_SMN
PLN_JOUR_ANNEE	D_N_JR_AN
PLN_SEMAINE_ISO	D_N_SMN
PLN_TRIMESTRE	D_N_TRM
PLN_SEMESTRE	D_N_SMR
PLN_NUM	D_N_Z
PLN_ALEA	D_N_R_POS

(Voir § 1.3.2 pour la définition des domaines SQL sus mentionnés)

Une fois transformées en tables ces entités seront alimentées en données afin d'être directement exploitables.

nota PLN_NUM contient une séquence de nombre de 0 à n
PLN_ALEA est un nombre réel compris entre]0 ... 1[calculé de manière aléatoire.
Ces nombres permettent des requêtes élaborées notamment pour le comptage ou le classement aléatoire

intérêt Cette table permet l'écriture élégante et simplifiée de requêtes apparemment difficiles.

1.3.13 – Modèle conceptuel de référence

A partir du premier modèle ayant été validé au niveau opérationnel, il sera créé un nouveau modèle conceptuel dit « de référence », contenant notamment, les domaines, les tables génériques et les tables d'auto documentation, afin de permettre la création de nouveaux modèles conceptuels pour de nouvelles applications, à partir de l'existant constitué par ce modèle de référence.

intérêt Accélérer le temps de modélisation d'une nouvelle base en partant d'éléments communs pré établis un peu à la manière des « modèles de document » de Word..

1.4 – Modèle physique

Il sera automatiquement produit par l'outil de modélisation choisit.

1.4.1 – Correction du modèle

Les noms des objets (tables et colonnes) seront directement dérivés du modèle conceptuel.

Aucune retouche ne sera apporté au modèle physique, à l'exception de :
- la modification des noms d'objet en cas d'erreur détecté après génération du MPD (doublons) ou de la vérification préalable à la génération du script ;
- la modification du positionnement des objets dans l'espace graphique afin de le rendre plus lisible.

Si l'outil de modélisation n'en a pas la capacité, le redécoupage en schéma sera fait au niveau physique, à moins que l'outil de conception permette de paramétrer les noms des tables avec un préfixe pointé. Dans ce cas, il faut penser, lors de la conception, à nommer les objets entité et association avec le préfixe de schéma.

1.4.2 – Évolution du modèle

Tant que la base de données ne sera pas en production (interne ou externe), ses évolutions se feront directement, sans conservation des modèles périmés.
A partir du moment où la base sera en production, chaque évolution du modèle fera l'objet d'une archive et il sera généré un script différentiel de modification de la base, comme un script de création.

A chaque script, il sera ajouté une commande SQL permettant de modifier dans la table T_S_DATABASE_INFO_DBI les données suivantes :
- nom du dernier script SQL ;
- version de la base.

intérêt les scripts différentiels SQL permettent de faire évoluer la base de données alors qu'elle est déjà en production. Différents clients peuvent être sur différentes versions et faire évoluer leur application a leur rythme. Le script complet permet à tout moment d'installer un nouveau client avec la dernière version, sans devoir passer toutes les mises à jour consécutives.

1.4.3– Préfixe des objets de la base

Des règles établies au niveau du modèle conceptuel découlent les préfixes T_ pour table et D_ pour domaine.

Touts les noms des objets de la base seront préfixés par une lettre suivie d'un blanc souligné. Les lettres de préfixes des objets de la base seront les suivants :

Lettre	Objet
S	**S**chéma
T	**T**able
V	**V**ue
P	**P**rocédure
F	**F**onction
D	**D**omaine & type utilisateur
C	**C**ontrainte
U	**U**tilisateur
R	**R**ôle
E	trigger, déclencheur (**É**vénement)
X	inde**X**
K	espace de stoc**K**age (file group, table Space)
G	ré**G**le

intérêt les préfixes permettent un repère immédiat de l'objet. La plupart des objets d'une base étant décrit dans des tables système « fourre-tout » il est plus aisé de comprendre la nature de l'objet par la structure de son nom que de s'obliger à des requêtes complexes pour la retrouver.

1.4.4 - Interfaçage des domaines aux SGBDR choisit

Dans le cas ou le SGBDR choisit ne comporte pas certains des domaines spécifiquement décrit, ces domaines seront complétés pas des contraintes élaborées par tout moyen propre au SGBDR.

Exemple : Le SGBDR MS SQL Server 2005 ne comporte pas les types SQL DATE et TIME (heure). Pour le domaine D_T_DATE, on partira du type DATETIME (date + heure) et on introduira une contrainte de mise à zéro de la partie horaire. Cela peut être fait comme suit :

```
-- ajout du type brut
sp_addtype 'D_T_DATE', DATETIME

-- ajout d'une règle de validation
CREATE RULE G_T_DATE
AS
FLOOR(CAST(@VALUE AS FLOAT)) = CAST(@VALUE AS FLOAT)

-- liaison de la règle au type
sp_bindrule 'G_T_DATE', 'D_T_DATE'
```

De même, le SGBDR SQL Server 2005, ne comportant pas le type SQL TIME (heure). On pourra réaliser ainsi le domaine D_T_HEURE, en manipulant des heures décimales et deux fonctions de conversion :

```
sp_addtype 'D_T_HEURE', FLOAT

CREATE RULE G_T_HEURE
AS
@VALUE BETWEEN 0.0 AND 23.999999444444444444444444444444

sp_bindrule 'G_T_HEURE', 'D_T_HEURE'

-- fonction de conversion heure décimale en string HH:MM:SS.nnn :

CREATE FUNCTION F_CONVERT_HD_HMS (@HD FLOAT)
RETURNS VARCHAR(12)
AS

BEGIN

DECLARE @H INTEGER
DECLARE @M INTEGER
DECLARE @S INTEGER
DECLARE @RETVAL VARCHAR(20)
DECLARE @MS CHAR(3)

-- cas triviaux
IF @HD IS NULL RETURN NULL
IF @HD = 0     RETURN '00:00:00 000'

-- récupération des heures, minutes, secondes, millième de seconde
SET @H  = FLOOR(@HD)
SET @HD = @HD - @H
SET @HD = @HD * 60
SET @M  = FLOOR(@HD)
SET @HD = @HD - @M
SET @HD = @HD * 60
SET @S  = FLOOR(@HD)
SET @HD = @HD - @S
SET @HD = @HD * 1000
IF ROUND(@HD, 0) = 1000
BEGIN
   SET @S = @S + 1
   SET @MS = '000'
END
ELSE SET @MS = CAST(ROUND(@HD, 0) AS CHAR(3))

-- conversion

IF @H < 10 SET @RETVAL = '0' + CAST(@H AS CHAR(1))+ ':'
   ELSE SET @RETVAL = CAST(@H AS VARCHAR(2))+':'
IF @M < 10 SET @RETVAL = @RETVAL + '0' + CAST(@M AS CHAR(1))+':'
   ELSE    SET @RETVAL = @RETVAL + CAST(@M AS CHAR(2))+':'
IF @S < 10 SET @RETVAL = @RETVAL + '0' + CAST(@S AS CHAR(1))
   ELSE    SET @RETVAL = @RETVAL + CAST(@S AS CHAR(2))
WHILE LEN(@MS) < 3
   SET @MS = '0' + @MS
SET @RETVAL = @RETVAL + '.' + @MS

RETURN @RETVAL

END
GO

-- fonction de conversion d'heure string HH:MM:SS.nnn en heure décimale:

CREATE FUNCTION F_CONVERT_HMS_HD (@HMS VARCHAR(12))
RETURNS FLOAT
AS

BEGIN
```

```
DECLARE @H FLOAT
DECLARE @M FLOAT
DECLARE @S FLOAT
DECLARE @MS FLOAT

-- cas trivial
IF @HMS IS NULL
   RETURN NULL

-- test du format
IF NOT @HMS LIKE '[0-9][0-9]:[0-9][0-9]:[0-9][0-9].[0-9][0-9][0-9]'
   RETURN NULL

-- récupération des données :
SET @H = CAST(SUBSTRING(@HMS, 1, 2) AS FLOAT)
SET @M = CAST(SUBSTRING(@HMS, 4, 2) AS FLOAT)
SET @S = CAST(SUBSTRING(@HMS, 7, 2) AS FLOAT)
SET @MS = CAST(SUBSTRING(@HMS, 10, 3) AS FLOAT)

RETURN @H + (@M / 60.0) + (@S / 3600.0) + (@MS / 3600000.0)

END
GO
```

Il faudra penser à réaliser ces mêmes fonctions et ces mêmes contraintes du côté du client avec en sus les masques de saisie correspondant.

1.4.4 - Cartographie des objets sur les fichiers de la base

Les objets de la base seront créés sur différents espace de stockage (File Group pour MS SQL Server, Table Space pour Oracle).
Pour cela sera établie une règle de base qui est la suivante :

- les données des tables seront créées dans l'espace de stockage par défaut (PRIMARY pour MS SQL Server)
- les index non cluster seront créés sur un espace de stockage de nom K_INDEXES

Pour les très grandes installations, il pourra être fait appel à un découpage plus détaillé.

intérêt La spécification d'emplacement des différents objets composant la base permet de répartir le volume des données (et donc la charge) sur différent disques physique afin d'améliorer sensiblement les temps de réponse sur les bases de données de grande taille.

1.4.5 – Fichiers SQL de création de la base

Les fichiers de création de la base de données, doivent être produits dans cet ordre :

0	utilisateurs, schémas, connexions	
1	objets du modèle	
2	contraintes additionnelles et triggers	
3	fonctions	
4	procédures	
5	primo insertion	données des références internes et externes
6	Indexation	

7	primo lancement	procédures à lancer pour initialiser la base
8	privilèges	

Cet ordre est l'ordre logique d'exécution pour l'enchaînement des scripts lors de la création de la base de données.

Toute base de données « cliente » devra être exclusivement créée par le biais de l'exécution de ces scripts, auquel il convient de faire précéder la création de la base elle-même (CREATE DATABASE…) et ses espaces de stockage.

nota L'implantation d'une base « cliente » par copier-coller (sauvegarde/restauration par exemple) reproduit la structure des données à l'identique, y compris sa fragmentation, ses données obsolètes et l'architecture des fichiers de la machine d'origine, grevant ainsi dès le départ les performances de la base.

Valeurs dans le référentiel :

(voir paragraphe 1.3.7)

Pour toutes les entrées de données concernant les tables du référentiel, la forme sera la suivante :

- la clef est forcée (valeur fournie)
- le code est en majuscule sans caractères diacritiques
- le libellé commence par une majuscule, tout le reste en minuscule
- la colonne xxx_BASE vaudra 1 pour toute donnée immuable

On entend par données « immuable » des données du référentiel explicitement utilisées dans les programmes et dont la valeur ne doit en aucun cas changer, sinon cela compromettrait le comportement du logiciel. En quelques sortes, ces entrées sont des constantes, écrites par facilité dans des tables.

Exemple :

```
SET IDENTITY_INSERT T_R_TITRE_TTR ON
INSERT INTO T_R_TITRE_TTR (TTR_ID, TTR_RANG, TTR_CODE, TTR_BASE, TTR_LIBELLE)
       VALUES             (1,      1,        'M.',     1,        'Monsieur')
INSERT INTO T_R_TITRE_TTR (TTR_ID, TTR_RANG, TTR_CODE, TTR_BASE, TTR_LIBELLE)
       VALUES             (2,      2,        'Mme.',   1,        'Madame')
INSERT INTO T_R_TITRE_TTR (TTR_ID, TTR_RANG, TTR_CODE, TTR_BASE, TTR_LIBELLE)
       VALUES             (3,      3,        'Mlle.',  1,        'Mademoiselle')
INSERT INTO T_R_TITRE_TTR (TTR_ID, TTR_RANG, TTR_CODE, TTR_BASE, TTR_LIBELLE)
       VALUES             (4,      6,        'Me.',    0,        'Maître')
INSERT INTO T_R_TITRE_TTR (TTR_ID, TTR_RANG, TTR_CODE, TTR_BASE, TTR_LIBELLE)
       VALUES             (5,      5,        'Mgr',    0,        'Monseigneur')
INSERT INTO T_R_TITRE_TTR (TTR_ID, TTR_RANG, TTR_CODE, TTR_BASE, TTR_LIBELLE)
       VALUES             (6,      4,        'MM.',    0,        'Messieurs')
SET IDENTITY_INSERT T_R_TITRE_TTR OFF
```

intérêt L'intérêt de mettre en majuscule la première lettre de l'expression du libellé est que cette fonction est difficile à réaliser par du code SQL, alors que le tout majuscules comme le tout minuscules s'opère par les fonctions basiques UPPER() et LOWER().

2 – Création de la base

C'est en amont que se jouent les problèmes les plus délicats de performances. Ce qui est fait au moment de la création de la base de données ne peut être modifié à chaud et s'il faut y remédier, cela nécessite une migration complète dans le cadre d'une réinstallation du serveur.
C'est pourquoi la création d'une base de données, comme l'installation du serveur doit être parfaitement planifiée dans son paramétrage.

2.1 – Installation du serveur (MS SQL Serveur)

2.1.1 – Disques et répartition des données

Le SGBDR sera installé dans la mesure du possible d'une des façons suivantes :

Nb disques	Grappe	Contenu
5	RAID 1	OS fichier d'échange mémoire (pagefile.sys), exécutable SQL Server, base tempDB, fichier journaux des bases applicatives
	RAID 5	fichier de données des bases applicatives fichier des index non cluster des bases applicatives
7	RAID 0	base tempDB fichier des index non cluster des bases applicatives fichier d'échange mémoire (pagefile.sys)
	RAID 1	OS exécutable SQL Server, fichier journaux des bases applicatives
	RAID 5	fichier de données des bases applicatives
8	RAID 1	OS fichier d'échange mémoire (pagefile.sys), exécutable SQL Server, base tempDB, fichier journaux des bases applicatives
	RAID 5	fichier de données des bases applicatives
	RAID 5	fichier des index non cluster des bases applicatives
10	RAID 0	base tempDB fichier d'échange mémoire (pagefile.sys)
	RAID 1	OS exécutable SQL Server, fichier journaux des bases applicatives
	RAID 5	fichier de données des bases applicatives
	RAID 5	fichier des index non cluster des bases applicatives

D'autres configurations plus spécifiques pourront être décidées en fonction des spécificités du client.

intérêt l'organisation en différentes grappes RAID à pour but d'assurer la redondance de l'écriture des informations dans le cadre de la garantie de bon fonctionnement sur service SQL Server (rappelons que Microsoft préconise les données en RAID 5 et les journaux en RAID 1), ainsi que la parrallélisation possible des ressources du disque afin d'augmenter les performances.

2.1.2 – Collation

Dans la mesure du possible, on choisira une collation binaire dans la langue local en s'assurant que c'est aussi la langue du serveur (une désynchronisation entre le jeu de caractères Windows et une collation localisée est contre performante).

En particulier, dans le cas d'une installation cliente en français, on prendra une version française de l'OS Server et la collation French_BIN (SQL Server 2000) ou French_BIN2 (SQL Server 2005) comme collation d'installation pour le serveur SQL.

intérêt le choix de la collation initiale de l'installation serveur participe de beaucoup aux performances en matière de recherches et mise en relation de données littérales.
D'autre part une collation forte oblige l'écriture des requêtes SQL telle que les noms ont été formés ce qui évite ainsi des calculs supplémentaires et minimise l'utilisation du cache de procédure.

2.2 – Emplacement de stockage des bases

Il sera créé au moins deux groupes de fichiers distincts pour le stockage des objets de la base.
Le premier (celui par défaut – PRIMARY pour MS SQL Server) sera la destination des objets logiques que sont : les tables, les vues, les types, etc… et tous les objets procéduraux (procédures stockées, fonctions, triggers…).
Le second, nommé expressément K_INDEXES contiendra tous les index non cluster.

intérêt la séparation des tables et des index en deux groupes de fichiers distincts permet de placer éventuellement ces fichiers sur différentes grappes RAID permettant ainsi de paralléliser lectures et écritures afin d'augmenter les performances.

2.3 – Création des espaces de stockage des données

Les différents espaces de stockage des données de différentes bases seront créés sur des disques neufs ou formatés immédiatement avant, en taille fixe avec une éventuelle possibilité de croissance automatique dont le pas d'incrément sera un multiple de 16 Mo (par exemple 16 Mo pour les petites bases, 64 Mo pour les moyennes bases et 256 Mo pour les grandes bases).

La taille de la base devant être calculée à l'aide de l'outil de modélisation, par l'estimation prévisionnelle du nombre de lignes de chacune des tables dans le cadre d'une exploitation de 36 mois. Selon la capacité des grappes RAID du serveur, et compte tenu d'un coefficient de sécurité qui doit se situer entre 1,3 et 2, il y aura peut-être intérêt à prendre 60% de la capacité du disque virtuel devant être affecté aux données.

La taille de l'espace de stockage des index, peut être estimée à 50% de la taille estimée de la base.

intérêt la création des fichiers de taille fixe pour la base de données garantie qu'il n'y aura pas de fragmentation physique (au niveau OS) du fichier, car les SGBDR comme SQL Server, Oracle ou IBM DB2, sont pourvu d'un algorithme qui recherche sur le disque le meilleur emplacement de granules contiguës pour créer le fichier de manière à minimiser le déplacement des têtes lors des lectures et écritures. L'avantage de prévoir une taille maximale est de réserver physiquement la place des données applicatives et d'empêcher ainsi une pollution du disque (occupation des espaces de stockage par des fichiers incongrus) qui risque de nuire aux performances, voire d'empêcher l'expansion de la base de données (disque plein).

Les fichiers de journalisation seront constitués en fichiers à taille variable, à croissance automatique (même pas d'incrément que ceux constituant la base) et fixé au départ à 25% de la taille estimée de la base.

intérêt un fichier de journalisation est en principe écrit par ajout et très rarement parcouru en arrière (en général uniquement dans le cas de ROLLBACK ou dans le cas de reprise après panne). Dans ce cas la meilleure stratégie de croissance consiste à favoriser le « WRITE AHEAD » par une taille de fichier variable.

2.4 – Gestion de l'intégrité référentielle

Le mode CASCADE est à proscrire (ON DELETE CASCADE, ON UPDATE CASCADE...). Préférez dans l'ordre, les modes : NO ACTION / RESTRICT, SET NULL, SET DEFAULT.

3 – Écriture du code des bases de données

L'ensemble du code SQL (requêtes, procédures, fonctions, trigger…) écrit à l'aide du langage Transact SQL devra être spécifié en majuscule pour tout ce qui concerne les noms des objets de la base (tables, vues, procédure, fonctions, variables…) et dans la casse de création pour les appels aux objets du serveur (procédures systèmes, fonctions systèmes, tables et vues systèmes).
Toutes les indentations devront se faire exclusivement à l'aide du caractère espace par bloc de trois espaces. L'utilisation du caractère de tabulation est à proscrire.
Dans les outils graphique d'écriture de requêtes (Analyseur de requêtes, éditeur Visual Studio, fenêtre de codage Entreprise Manager, Word…) on prendra soin d'utiliser toujours une police à espacement fixe. La police préconisée est Lucida Console.

intérêt Le caractère tabulation génère des différences importantes de présentation suivant l'environnement dans lequel il est vu (Analyseur de requêtes, éditeur Visual Studio, fenêtre de codage Entreprise Manager, bloc note, Word…). De même pour l'utilisation d'une police à espacement proportionnelle.
Or une présentation homogène rend la lecture et la comparaison plus aisée

Dans la mesure du possible le style d'écriture des programmes devra être calqué sur le modèle relationnel objet, permettant la manipulation d'objet relationnel comme un « tout ». Par exemple, l'insertion ou la modification des informations relatives à une personne et ses différents mails, adresse, téléphones, etc… devra être réalisé à partir d'une seule procédure pouvant éventuellement en appeler d'autres.

intérêt Minimisation des aller et retour réseau et de la durée des transactions, dans le but de maximiser les performances.

3.1 – Écriture des requêtes SQL

3.1.1 – Tag

Qu'il s'agisse d'une requête complète, paramétrée ou dynamique, toute requête encapsulée dans du code régulièrement lancé devra être identifié à l'aide d'un tag composé d'un GUID, comme suit :

```
/* BF6ED289-0E07-4BA7-8B6C-B0E3C1537930 */
```

placé immédiatement derrière le premier mot clef de la requête (CREATE, DROP, ALTER, SELECT, INSERT, UPDATE, DELETE…)

Exemple

```
UPDATE /* BF6ED289-0E07-4BA7-8B6C-B0E3C1537930 */ MA_TABLE
```

```
SET
   MA_COLONNE1 = 'xyz '
WHERE
   MA_COLONNE2 = 456
```

Ce GUID sera obtenu par un appel à un générateur de GUID (par exemple SELECT NEWID() dans MS SQL Server)

intérêt Le tag est le seul moyen de tracer des requêtes dont la composition change (paramètre, composition dynamique du code SQL), afin d'auditer les performances du SGBDR et d'en extraire les éléments à optimiser en priorité.

3.1.2 – Présentation et règles syntaxiques

Que la requête soit externe ou interne (imbrication dans le cadre de sous requêtes par exemple), les différentes clauses devront être indentées comme suit :

- le mot clef de la clause sur une ligne ;
- le détail de la clause sur n lignes ne dépassant pas en principe 80 caractères et indenté de 3 espaces (pas de tabulation).

Exemple

```
SELECT /* BF6ED289-0E07-4BA7-8B6C-B0E3C1537930 */
   MA_COLONNE1, MON_AUTRE_COLONNE2, UNE_COLONNE3
FROM
   MA_TABLE
ORDER BY
   MA_COLONNE1
```

SELECT :

- toutes les expressions de la clause SELECT seront nommées.
- la clause ORDER BY doit faire référence aux noms exprimés dans la clause SELECT et non à la position ordinale de l'expression.

Exemples :

A proscrire :	`SELECT /* BF6ED289-0E07-4BA7-8B6C-B0E3C1537930 */`
	`   355 / 113, COLONNE_A`
	`FROM`
	`   T_XYZ`
	`ORDER BY`
	`   1`

A utiliser	`SELECT /* BF6ED289-0E07-4BA7-8B6C-B0E3C1537930 */`
	`   355 / 113 AS COL_CALC, COLONNE_A`
	`FROM`
	`   T_XYZ`
	`ORDER BY`
	`   COL_CALC`

L'utilisation d'alias (AS) non conforme aux spécifications des noms SQL est proscrit.
L'utilisation des syntaxes SQL non-conformes aux spécifications de la norme SQL2 de 1992 est proscrit (jointures normalisées en particulier). Chaque fois qu'une syntaxe normalisée est disponible il y a lieu de l'utiliser.

A proscrire :	A utiliser
MA_COLONNE = 123	123 AS MA_COLONNE
MA_COLONNE = GETDATE()	MA_COLONNE = CURRENT_TIMESTAMP
CONVERT(CHAR(20), MA_COLONNE)	CAST(MA_COLONNE, CHAR(20))
ISNULL(MA_COLONNE, 0)	COALESCE(MA_COLONNE, 0)

intérêt Portabilité d'un SGBDR à l'autre.

Cas des jointures :

- les jointures devront toujours s'effectuer à l'aide de clause JOIN (norme SQL2, 1992) et non dans la clause WHERE.
- des alias composé à l'aide du trigramme seront systématiquement utilisés pour les spécifications de colonnes en relations
- en cas de multijointure sur la même table l'alias (trigramme) sera suffixé par un chiffre incrémenté à partir de 1 et introduit par un blanc souligné
- la table racine sera écrite sur une ligne (indenté de 3 espaces)
- chaque nouvelle table en dehors de la table racine sera introduite sur une nouvelle ligne avec son type de jointure (...JOIN...) avec une indentation de 6 espaces
- chaque clause de jointure (ON) sera introduite immédiatement, si possible à côté de la table quelle lie, sinon en dessous avec une indentation de 9 espaces
- si la clause de jointure ne tient pas sur une seule ligne, elle sera écrite sur plusieurs lignes indentées de 12 espaces.

Exemples :

A proscrire :

```
FROM
   T_ABC, T_DEF, T_GHI
WHERE
   T_ABC.COL1 = T_DEF.COL2
   AND T_ABC.COL4 *= T_GHI.COL7
```

A utiliser

```
FROM
   T_ABC ABC
      INNER JOIN T_DEF DEF       ON ABC.COL1 = DEF.COL2
      LEFT OUTER JOIN T_GHI GHI  ON ABC.COL4 = GHI.COL7
```

intérêt La syntaxe de jointure dans la clause WHERE ne sera plus supportée dans les versions futures des SGBDR.
Les clauses JOIN permettent un calcul du plan de requête plus rapide.
La syntaxe normative permet plus facilement d'isoler tout ou partie des éléments d'une jointure lorsque l'on met au point la requête et d'éviter les produits cartésiens intempestifs lorsque l'on modifie la clause WHERE lors de la mise au point de la requête.

Lorsque la clause SELECT de l'ordre SELECT, ou les filtres WHERE et HAVING font apparaître de multiples colonnes venant de différentes tables (jointures) il y aura lieu de regrouper les expressions par table, dans la mesure du possible et de les faire précéder d'une ligne de commentaire faisant référence à la table.

Exemples :

A proscrire :

```
SELECT UDC.epsDateCloture, EDP.epsUdP,
EDP.epsContreRemboursement, ADE.epsLibelle AS
NomExpediteur, ADE.epsCodePostal AS CodePostalExpediteur,
ADE.epsVille AS VilleExpediteur, cast(CLS.epsPoids as
decimal(20,2)) as epsPoids, CLS.epsNoColis,
CLS.epsRangColis, SPE_DGDIFF_UDP, DDF.CoursierMPL

FROM   ...

WHERE  CLS.epsId = 789   {idColis} AND  UDC.epsDateCloture
BETWEEN DATEADD(month, 1, CURRENT_TIMESTAMP ) AND
CURRENT_TIMESTAMP AND  ADE.epsVille IN ('Paris', 'Berlin',
'Toronto')
```

A utiliser

```
SELECT -- EXP_UDC
       UDC.epsDateCloture,

       -- EPS_UDP
       EDP.epsUdP,
       EDP.epsContreRemboursement,

       -- TRP_ADRESSESEXPEDITION
       ADE.epsLibelle AS NomExpediteur,
       ADE.epsCodePostal AS CodePostalExpediteur,
       ADE.epsVille AS VilleExpediteur,

       -- EXP_UDC_COLIS
       cast(CLS.epsPoids as decimal(20,2)) as epsPoids,
       CLS.epsNoColis,
       CLS.epsRangColis,

       -- SPE_DGDIFF_UDP
       DDF.CoursierMPL

FROM   ...


WHERE  -- EXP_UDC_COLIS
       CLS.epsId = 789   {idColis}

       -- EXP_UDC
  AND  UDC.epsDateCloture
          BETWEEN DATEDD(month, 1, CURRENT_TIMESTAMP )
              AND CURRENT_TIMESTAMP

       -- TRP_ADRESSESEXPEDITION
  AND  ADE.epsVille IN ('Paris', 'Berlin', 'Toronto')
```

INSERT :

La clause de spécification des colonnes cible devra être explicite. Seules les colonnes réellement renseignées ou spécifiée à NULL en cas de présence d'une valeur défaut devront y figurer. L'ordre des colonnes doit être celui de la position ordinale des colonnes dans la table (ordre de création des colonnes).

Autant que faire se peut, les références des colonnes à insérer seront mises en correspondance avec leurs valeurs l'une au dessus de l'autre.

Exemple, soit la table définie comme suit :

```
CREATE TABLE T_CLIENT_CLI
(CLI_ID     INT          NOT NULL PRIMARY KEY,
```

```
 CLI_NOM    CHAR(32)    NOT NULL,
 CLI_TITRE  VARCHAR(12)          DEFAULT 'Monsieur',
 CLI_PRENOM VARCHAR(25),
 CLI_SEXE   CHAR(1)              DEFAULT 'M'
                        CHECK (CLI_SEXE IN ('M', 'F'))
)
```

A proscrire :	INSERT INTO T_CLIENT_CLI CLI_ID, CLI_NOM, CLI_PRENOM, CLI_TITRE, CLI_SEXE) VALUES (1, 'DUPONT', NULL, 'Monsieur', DEFAULT)
A utiliser	INSERT INTO T_CLIENT_CLI (CLI_ID, CLI_NOM) VALUES (1, 'DUPONT')

DELETE :

Le mot FROM est requis dans la clause DELETE.

Exemples :

A proscrire :	DELETE T_CLIENT_CLI WHERE CLI_NOM = 'Dupont'
A utiliser	DELETE FROM T_CLIENT_CLI WHERE CLI_NOM = 'Dupont'

intérêt Portabilité des requêtes d'un SGBDR à l'autre et en cas de modification de schéma.

3.2 – Écriture du code Transact SQL

En tête de chaque procédure, fonction, trigger ou vue, devra figurer un cartouche, après le mot clef AS dont le modèle est le suivant :

```
/**************************************************************************
Base         : MaBase
Schema       : dbo
Objet        : fonction
Auteur       : Frédéric Brouard
Copyright    : MonEntreprise
Créée le     : 2006-01-29
Release      : C3
********************************* REVISIONS ******************************
Date         : 2005-01-17
Auteur       : Frédéric Brouard
Tag          : #FBD050117#
Nature       : ajout de la fonction COALESCE pour la colonne EpsId dans la
               requête SELECT
******************************** UTILISATION *****************************
Description : Permute les deux parties d'une chaîne de caractères autour
              du caractères d'indice "pivot"
Paramètres  : @data IN (alphanum) la chaîne à traiter
              @pivot IN (int) la position du caractère
              OUT (alphanum) le résultat du pivotement
Utilise     : <objets de la base utilisé>
**************************************************************************/
```

Chaque révision du code devra faire l'objet d'un tag identifiant la partie de code révisée. L'ancien code sera conservé en commentaire.

Exemple :

```
/* #FBD050117# début de modif, ancien code
SELECT Eps_ID
nouveau code */
SELECT COALESCE(Eps_ID,0)
-- fin de modif #FB050117#
```

Le tag de révision est constitué d'un trigramme permettant d'identifier l'auteur et de la date au format AAMMJJ. Il devra indiquer le début et la fin de la modification.

Les boucles et test seront écrits sur une seule ligne lorsqu'ils ne concernent qu'une instruction.

3.2.1 – Écriture des fonctions SQL (Transact SQL)

- Le nom de toute fonction SQL sera préfixée F_.
- Lorsque la fonction est une fonction table, elle sera dotée d'un préfixe complémentaire T_.
- Lorsque la fonction est un calcul d'agrégat (SQL Server 2005) elle sera dotée d'un préfixe complémentaire A_.

Par défaut, les fonctions sont présumées scalaires.

Exemples :

```
F_MAX_DATES (D1 DATETIME, D2 DATETIME)          fonction scalaire
F_T_SEMAINE (D DATETIME)                        fonction table
F_A_CONCAT_STRING (S NVARCHAR(max))             fonction agrégat
```

Par principe une fonction ne doit jamais conduire à une erreur d'exécution. En particulier en présence de paramètres non renseignés (absence de valeur caractérisée par le marqueur NULL) on renverra immédiatement le marqueur NULL avant tout déroulement du code.
De même, si pour une valeur ou plage de valeurs particulière, une fonction est susceptible de générer une erreur, alors on testera préalablement ces conditions et en cas de succès on retournera immédiatement une valeur de principe ou à défaut le marqueur NULL.
Par exemple, si une fonction contient une division, on testera le diviseur pour savoir si sa valeur est égale à zéro, et dans ce cas, on renverra le marqueur NULL en sortie.

Les variables utilisées dans le corps d'une fonction doivent être déclarées au plus près de leur utilisation et non en tête de code.

Exemple : soit la fonction F_FLIP qui intervertit les différentes parties d'une expression littérale que nous appellerons MOT à partir de la lettre d'indice PIVOT…

F_FLIP (MOT, PIVOT)	**résultat**
F_FLIP('flap', 2)	aplf
F_FLIP('flep', 1)	lepf
F_FLIP('flip', 4)	pfli

F_FLIP('flop', 0)	flop
F_FLIP('flup', -3)	flup
F_FLIP('flyp', 12)	flyp
F_FLIP('', 5)	
F_FLIP(NULL, 2)	<NULL>

La fonction devra être codée comme suit :

```
CREATE FUNCTION F_FLIP (@MOT VARCHAR(8000), @PIVOT INTEGER)
   RETURNS VARCHAR(8000)
AS

/*
================================================================================
Fichier :     MaBase-PRIMARY
Copyright :   MonEntreprise
Auteur :      Frédéric Brouard
Créée le :    2006-01-29
Description : Permute les deux parties d'une chaîne de caractères autour
              du caractères d'indice "pivot"
================================================================================
*/

BEGIN

-- condition générales SQL, paramètre NULL
IF @MOT IS NULL OR @PIVOT IS NULL
   RETURN NULL

-- effet de bord : mot vide
IF @MOT = ''
   RETURN ''

-- pivote en dehors des limites de calculs
IF @PIVOT NOT BETWEEN 1 AND LEN(@MOT) + 1
   RETURN @MOT

-- cas général, les paramètres sont opérables
DECLARE @RETVAL VARCHAR(8000)
DECLARE @LONG   INTEGER

SET @LONG = LEN(@MOT)

-- pivot est le premier caractère
IF @PIVOT = 1
BEGIN
   IF @LONG = 1
      RETURN @MOT
   RETURN SUBSTRING(@MOT, 2, @LONG - 1) + SUBSTRING(@MOT, 1, 1)
END

-- pivot est le dernier caractère
IF @PIVOT = @LONG
   RETURN SUBSTRING(@MOT, @LONG, 1) + SUBSTRING(@MOT, 1, @LONG - 1)

-- pivot est au milieu
RETURN SUBSTRING(@MOT, @PIVOT + 1, @LONG - @PIVOT) +
       SUBSTRING(@MOT, @PIVOT, 1) +
       SUBSTRING(@MOT, 1, @PIVOT - 1)

END

GO
```

intérêt Une fonction SQL peut être appelée des millions de fois en fonction des lignes qu'elle est susceptible de traiter dans le cadre d'un SELECT. Le traitement à priori des valeurs triviales des paramètres (NULL, vide, zéro…) permet d'éviter de dérouler un code inutile répétés des milliers de fois…

3.2.2 – Écriture des procédures stockées (Transact SQL)

Nom d'une procédure :

Le nom de toute procédure SQL sera préfixée P_ suivit d'un préfixe complémentaire choisit parmi les lettres A, B, S, I, U, M ou D :

A Admin : procédure à vocation d'administration de la base de données (maintenance, sauvegarde…)
B Batch : procédure d'import, export de données
S Select : la procédure renvoie essentiellement un jeu de résultat (elle contient au moins un ordre SELECT).
I Insert : la procédure insère des données
U Update : la procédure modifie les données
D Delete : la procédure supprime des données
M Merge : la procédure insère ou modifie des données (combinaison d'INSERT ou UPDATE en fonction des conditions d'existence des tuples en entrée).

Un suffixe de deux chiffres indiquera la version majeure de la procédure.

Exemples :

P_D_MEUBLE_00	procédure de suppression d'un meuble
P_M_ATELIER_01	procédure d'insertion ou mise à jour d'un atelier, release 1
P_B_ARCHIVE_00	procédure d'archivage des données
P_A_MAINT_INDEX_00	procédure d'administration : maintenance des index

nota Seules les modifications majeures des procédures doivent être spécifiées. Par modification majeure on entend toute modification qui entraîne un appel de la procédure ou renvoi un jeu de résultat dont la forme diffère de l'origine.
Par exemple un paramètre en plus ou en moins, une colonne en plus ou en moins dans le jeu de résultat.
Par opposition une modification mineure n'entache pas l'enveloppe externe d'exécution de la procédure (correction de bug, optimisation… par exemple)

Paramètres :

Le nom des paramètres devra :

- être celui de la colonne si le paramètre représente une colonne de table (cas des INSERT et UPDATE notamment) ;
- être significatif
- être préfixé par son type en cas d'ambiguïté

Le type des paramètres devra :

- être celui de la colonne (domaine SQL) si le paramètre représente une colonne de table (cas des INSERT et UPDATE notamment) ;
- être un type simple SQL, le plus proche de l'expression à traiter.

- être choisit le plus générique possible (par exemple NVARCHAR pour les alpha.) dans le cas de procédures traitant des données « génériques »
- ne pas utiliser le type SQL_VARIANT dans la mesure du possible.

L'ordre des paramètres devra :
- commencer par les paramètres impératifs ;
- suivi par les paramètres output s'il y en a
- finir par les paramètres optionnels, dans l'ordre du moins au plus optionnel

tous les paramètres optionnels devront être pourvus d'une valeur par défaut

Transaction :

A moins qu'elles ne contiennent qu'une seule requête, les procédures de type I, U, D ou M devront faire l'objet d'une transaction dont le niveau d'isolation doit être soigneusement choisit.
En particulier on laissera le niveau d'isolation par défaut (READ COMMITTED) dans la majorité des cas. Si le besoin de relecture stable est nécessaire, alors il y aura lieu de passer au niveau d'isolation REPEATABLE READ. En cas de génération ou de modification de valeurs de clefs avec répercussion dans d'autres tables, le niveau d'isolation SERIALIZABLE sera préféré.

Toute modification du niveau d'isolation (SET TRANSACTION ISOLATION LEVEL ...) devra être remise à sa valeur par défaut en sortie de procédure.

Enfin, les transactions seront anonymes (aucun nom de transaction ne doit être explicitement fourni pour les commandes BEGIN TRANSACTION et COMMIT / ROLLBACK)

Compte tenu de la problématique de récupération sur erreur dans les transactions et de la problématique de transactions imbriquées, il sera mis en place le code suivant dans toutes les procédures devant être transactionnées :

```
...

DECLARE @RETVAL INT

... code ...

COMMIT TRANSACTION

SET @RETVAL =0

GOTO LBL_RESUME

LBL_ERROR:

WHILE @@TRANCOUNT > 1
   COMMIT TRANSACTION

IF @@ROWCOUNT = 1
   ROLLBACK TRANSACTION

SET @RETVAL =-1

LBL_RESUME:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

```
RETURN @@RETVAL
GO
```

Codage :

Le corps d'une procédure sera contenu dans un bloc BEGIN / END.

Lorsqu'il y a lieu :

- les paramètres passés à une procédure pourront être en sortie (OUTPUT).
- une valeur par défaut peut être assignée aux paramètres.

Les variables utilisées dans le corps d'une procédure doivent être déclarées au plus près de leur utilisation et non en tête de code.

Le flag NOCOUNT sera mis à ON en tête de toute procédure contenant une ou plusieurs requêtes.

Gestion des erreurs :

L'instruction GOTO sera exclusivement réservée à la gestion des erreurs (SQL Server 2000). Avec la version 2005, le GOTO est proscrit.

La gestion des erreurs devra faire apparaître :

- Un bloc de branchement en cas de succès (label LBL_OK:) suivi d'un dépilage de COMMIT.
- Un bloc de branchement en cas d'échec (label LBL_ERROR:) suivi d'un dépilage de ROLLBACK.
- Un bloc de finalisation s'il y a lieu (label LBL_RESUME:), prévoyant notamment le rétablissement du niveau d'isolation par défaut (READ COMMITED)

Le mot clef RETURN ne devra jamais être détourné de sa fonction de renvoi du contrôle d'exécution de la procédure. En d'autre terme il ne doit pas être utilisé, sauf à transmettre une erreur de procédure appelée en procédure appelante.

Exemple :
Soit les tables suivantes décrivant une personne physique :

```
CREATE TABLE T_TITRE_TTR
(TTR_ID          INT           NOT NULL IDENTITY PRIMARY KEY,
 TTR_CODE        CHAR(8)       NOT NULL          UNIQUE,
 TTR_LIBELLE     VARCHAR(32)
)

CREATE TABLE T_PERSONNE_PRS
(PRS_ID          INT           NOT NULL IDENTITY PRIMARY KEY,
 TTR_ID          INT                            FOREIGN KEY
                               REFERENCES T_TITRE_TTR (TTR_ID),
 PRS_NOM         CHAR(32)      NOT NULL,
```

```
 PRS_PRENOM     VARCHAR(25)
)

CREATE TABLE T_EMAIL_EML
(EML_ID         INT          NOT NULL IDENTITY PRIMARY KEY,
 PRS_ID         INT          NOT NULL          FOREIGN KEY
                             REFERENCES T_PERSONNE_PRS (PRS_ID),
 EML_MAIL       VARCHAR(256) NOT NULL,
 EML_ORDRE      INT
)

SET IDENTITY_INSERT T_TITRE_TTR ON
INSERT INTO T_TITRE_TTR(TTR_ID, TTR_CODE, TTR_LIBELLE) VALUES (1, 'M.', 'Monsieur')
INSERT INTO T_TITRE_TTR(TTR_ID, TTR_CODE, TTR_LIBELLE) VALUES (2, 'Mme.', 'Madame')
SET IDENTITY_INSERT T_TITRE_TTR OFF
```

Procédure d'insertion ou d'ajout d'un mail pour une personne physique :

```
CREATE PROCEDURE P_M_PERSONNE_MAIL_00 @PRS_ID    INT,
                                      @EML_QUERY CHAR(1),
                                      @EML_MAIL  VARCHAR(256),
                                      @EML_ORDRE INT
AS

/*
================================================================================
Fichier :     MaBase-PRIMARY
Copyright :   MonEntreprise
Auteur :      Frédéric Brouard
Créée le :    2006-01-29
Description : ajoute ou modifie un email rattaché à une personne
================================================================================
*/

BEGIN

SET NOCOUNT ON

-- ne rien faire si références ou données NULL
IF @EML_ORDRE IS NULL OR @PRS_ID IS NULL OR @EML_MAIL IS NULL RETURN

IF @EML_QUERY = 'U'
-- c'est probablement une mise à jour (U pour UPDATE)
BEGIN
   IF EXISTS(SELECT *
             FROM   T_EMAIL_EML
             WHERE  PRS_ID = @PRS_ID
                    AND  EML_ORDRE = @EML_ORDRE)
   BEGIN
-- si mail vide => suppression
      IF @EML_MAIL = ''
         DELETE FROM T_EMAIL_EML
         WHERE  PRS_ID = @PRS_ID
           AND  EML_ORDRE = @EML_ORDRE
      ELSE
-- mail non vide => mise à jour
         UPDATE T_EMAIL_EML
         SET    EML_MAIL = @EML_MAIL
         WHERE  PRS_ID = @PRS_ID
           AND  EML_ORDRE = @EML_ORDRE
         RETURN
   END
   ELSE
-- cette référence de mail n'existe pas, donc c'est une insertion
      SET @EML_QUERY = 'I'
END

IF @EML_QUERY = 'I' AND @EML_MAIL <> ''
-- c'est une insertion
   INSERT INTO T_EMAIL_EML (PRS_ID,  EML_MAIL,  EML_ORDRE)
          VALUES           (@PRS_ID, @EML_MAIL, @EML_ORDRE)

END
```

Procédure d'insertion ou d'ajout d'une personne physique :

```
CREATE PROCEDURE P_M_PERSONNE_00 @PRS_ID  INT,
                                 @PRS_NOM CHAR(32),
                                 @PRS_PRENOM VARCHAR(25),
                                 @TTR_ID  INT,
                                 @EML_MAIL1 VARCHAR(256),
                                 @EML_MAIL2 VARCHAR(256),
                                 @EML_MAIL3 VARCHAR(256)
AS

/*
================================================================================
Fichier :     MaBase-PRIMARY
Copyright :   MonEntreprise
Auteur :      Frédéric Brouard
Créée le :    2006-01-29
Description : ajoute ou modifie les données relative à une personne
================================================================================
*/

BEGIN

SET NOCOUNT ON

DECLARE @RETVAL INT

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION

IF @PRS_ID IS NULL
-- c'est une insertion
BEGIN
   -- insertion dans la table personne
   INSERT INTO T_PERSONNE_PRS (TTR_ID,  PRS_NOM,  PRS_PRENOM)
          VALUES              (@TTR_ID, @PRS_NOM, @PRS_PRENOM)
   IF @@ERROR <> 0 GOTO LBL_ERROR

   -- récupération de l'identifiant de personne
   SET @PRS_ID = @@IDENTITY

   -- insertion MAIL 1 :
   EXEC P_M_PERSONNE_MAIL_00 @PRS_ID, 'I', @EML_MAIL1, 1
   IF @@ERROR <> 0 GOTO LBL_ERROR

   -- insertion MAIL 2 :
   EXEC P_M_PERSONNE_MAIL_00 @PRS_ID, 'I', @EML_MAIL2, 2
   IF @@ERROR <> 0 GOTO LBL_ERROR

   -- insertion MAIL 3 :
   EXEC P_M_PERSONNE_MAIL_00 @PRS_ID, 'I', @EML_MAIL3, 3
   IF @@ERROR <> 0 GOTO LBL_ERROR
END
ELSE
-- c'est une modif
BEGIN
   -- mise à jour personne
   UPDATE T_PERSONNE_PRS
   SET    TTR_ID = @TTR_ID,
          PRS_NOM = @PRS_NOM,
          PRS_PRENOM = @PRS_PRENOM
   WHERE  PRS_ID = @PRS_ID
   IF @@ERROR <> 0 GOTO LBL_ERROR

   -- mise à jour MAIL 1 :
   EXEC P_M_PERSONNE_MAIL_00 @PRS_ID, 'I', @EML_MAIL1, 1
   IF @@ERROR <> 0 GOTO LBL_ERROR

   -- mise à jour MAIL 2 :
   EXEC P_M_PERSONNE_MAIL_00 @PRS_ID, 'I', @EML_MAIL2, 2
   IF @@ERROR <> 0 GOTO LBL_ERROR

   -- mise à jour MAIL 3 :
   EXEC P_M_PERSONNE_MAIL_00 @PRS_ID, 'I', @EML_MAIL3, 3
   IF @@ERROR <> 0 GOTO LBL_ERROR
END
```

```
-- code terminé

-- succès
LBL_OK:

COMMIT TRANSACTION

SET @RETVAL = 0

GOTO LBL_RESUME

-- erreur
LBL_ERROR:

WHILE @@TRANCOUNT > 1
   COMMIT TRANSACTION

IF @@TRANCOUNT = 1
   ROLLBACK TRANSACTION

SET @RETVAL = -1

-- finalisation
LBL_RESUME:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

SET NOCOUNT OFF

RETURN @@RETVAL

END
```

intérêt Les procédures stockées sont le meilleur moyen de gérer les transactions de la manière la plus rapide possible afin de minimiser la contention.
Le fait de calquer les écritures dans la base sur un modèle proche de l'objet diminue les risques d'incohérence et accélère les traitements.

3.2.3 – Écriture des triggers (Transact SQL)

En principe le codage d'un déclencheur ne doit être entrepris que dans les cas de l'extension du modèle relationnel : héritage avec exclusion multiple, non relation, contrainte de validité complexe, contrainte d'unicité partielle…
Dans tous les autres cas, il faut systématiquement envisager la réalisation d'une procédure stockée.

Le code contenu dans un trigger doit toujours être écrit de manière ensembliste (requête SQL uniquement). Il ne doit jamais y avoir de déclaration de variable ni utilisation de curseur.
Lorsqu'un tel cas doit être envisagé, alors il faut transformer la logique en différentes procédures stockées.

Lorsque la logique d'un trigger entraîne un ROLLBACK de la transaction appelante, ce ROLLBACK doit être immédiatement suivi de la génération d'une exception.

intérêt Le trigger est l'objet le plus coûteux dans un SGBDR. Son code doit donc être le plus performant possible, ce qui n'est réalisable que s'il

est écrit uniquement à base de requêtes SQL qui, par nature, sont optimisées.

Les pseudos tables de scrutation du curseur seront écrites et aliassées comme suit :

pseudo table	alias
inserted	i
deleted	d

Exemple, réalisation d'une contrainte d'unicité avec de multiples entrées vides, (marqueur NULL) - soit la table :

```
CREATE TABLE T_EMPLOYE_EMP
(EMP_ID           INT      NOT NULL PRIMARY KEY IDENTITY,
 EMP_NOM          CHAR(32) NOT NULL,
 EMP_MATRICULE    CHAR(8))
```

Le trigger ci dessus réalise une contrainte d'unicité sur la colonne EMP_MATRICULE telle que l'entend la norme SQL (SQL Server n'étant pas conforme à la norme sur ce point) :

```
CREATE TRIGGER E_EMP_MATRICULE_UNIQUENULL
   ON T_EMPLOYE_EMP
FOR INSERT, UPDATE
AS
BEGIN

IF EXISTS(SELECT COUNT(EMP.EMP_MATRICULE)
          FROM   T_EMPLOYE_EMP EMP
                 INNER JOIN inserted i
                       ON EMP.EMP_MATRICULE = i.EMP_MATRICULE
          GROUP  BY EMP.EMP_MATRICULE
          HAVING COUNT(EMP.EMP_MATRICULE) > 1)
   BEGIN
     ROLLBACK
     RAISERROR(' TRIGGER E_EMP_MATRICULE_UNIQUENULL : Violation de la'
               + ' contrainte d''unicité sur les valeurs'
               + ' exprimées de la colonne EMP_MATRICULE, '
               + ' table T_EMPLOYE_EMP', 16, 1)
   END

END
```

3.2.4 – Écriture des scripts (Batch Transact SQL)

Dans la mesure du possible les scripts ne doivent pas être transactionnés.

Lorsque les scripts concernent de grands volumes de données (import et particulier), il y a lieu de recherche un découpage du script par paquet de données Dans le même esprit, le débranchement des index et des triggers préalable à une insertion massive est souhaitable.

intérêt Le découpage en lots plus petits accélère les traitements. Se souvenir que l'unité de stockage dans les tables MS SQL Server est l' « extent », soit 64 Ko. Un découpage en sous lots multiples de cet unité minimise le volume des transactions.
La désactivation des index et triggers préalable à une insertion massive, puis leur réactivation une fois le travail terminé, diminue drastiquement les temps de réponse.

3.3 – Écriture du code CLR (SQL Server 2005)

SQL CLR permet d'écrire :

- des types de données utilisateurs (UDT)
- des procédures stockées
- des fonctions SQL
- des fonctions d'agrégation SQL
- des triggers

dans une des langages concourant à la plateforme .net de Microsoft (C#, VB .net, Delphi .net, etc…)
Cependant, les tests effectués, confirmés par les recommandations de Microsoft, montrent que SQL CLR est moins performant dans tous les cas d'accès aux données que le Transact SQL. De plus comme tout langage de type L4G, il est victime du défaut d'impédance très difficile à contourner.

intérêt

On réservera SQL CLR à l'écriture de :

- fonction mathématiques ou de traitement de chaînes de caractères
- procédures faisant peu appel aux données et beaucoup aux calculs
- réalisation de nouveaux types de données complexes
- fonctions de calculs d'agrégat

On s'interdira d'utiliser SQL CLR pour l'écriture des triggers

ANNEXE 1 – VOCABULAIRE

Association	spécification des liens et relations entre deux ou plusieurs entités.
Attribut	Information de nature atomique propre à décrire une des caractéristiques d'une entité. Exemple : attribut « titre » d'une entité « livre »
Catalogue	Un CATALOG SQL constitue en fait une base de données qui elle-même peu être découpée en schémas.
Collation	Ensemble des règles qui dictent le comportement d'un jeu de caractères dans un environnement base de données (sensibilité ou non à la casse, aux caractères diacritiques, à la largeur d'encodage…)
Défaut d'impédance	(en informatique) différence de comportement de deux système de gestion de l'information en présence d'un même traitement, dû à la façon dont sont gérées de manière interne les données. Par exemple entre SQL et les L4G (Java, C++, C#...) la représentation de l'absence de valeur n'est pas traitée de manière identique (NULL), comme les collations ou la gestion ensembliste des données (UNION).
Diacritique	Du grec diakrinein, "distinguer". Se dit d'un signe qui accompagne un caractère pour permettre de le distinguer phonétiquement : accents, cédille, tilde, par extension ligature (œ=oe, æ=ae, &=et, @=ad...)
DOMAIN (SQL)	Voir domaine
Domaine	Ensemble des valeurs qu'un attribut peut prendre. Exemple, un attribut caractérisant un pourcentage peut prendre toute la gamme des valeurs numérique de 0 à 100. Dans ce cas, la contrainte de domaine sera la plage des valeurs de 0 à 100, traduite en SQL par la contrainte de validité CHECK VALUE BETWEEN 0.0 AND 100.0
E/R	(Entité / Relation) notation de modèle relationnel proche du concept élaboré par Peter Chen : The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems (TODS) Volume 1 Number 1: pp 9-36 (1976)
Entité	ensemble d'information cohérente permettant de décrire une même famille d'objet. Représentation d'une famille d'éléments matériels ou immatériels que l'on désire modéliser dans le système d'information.
Exclusion mutuelle	Lors de la modélisation de données en héritage, les éléments fils peuvent s'exclure ou s'associer suivant la nature du modèle. Exemple : père VEHICULE, fils MOTO, VOITURE, CAMION => exclusion mutuelle. Père VEHICULE, fils AVION, BATEAU, VOITURE => pas d'exclusion mutuelle sinon que faire de l'hydravion ou de la voiture amphibie ?
GUID	Globally Unique IDentifier : Identificateur global unique codé sur 128-bits (16 octets). L'algorithme utilisé pour le générer garantit statistiquement que le code résultant est unique dans l'espace et le temps. Cet algorithme est tiré d'un standard de l'Open Software Foundation (OSF) Distributed Computing Environment (DCE). Les critères retenus pour l'implémentation de l'algorithme spécifié par OSF DCE sont : • Date et heure système (à 1/10 micro-secondes près). • Incrémentation forcée de l'identifiant pour les créations de GUID à fréquence élevée. • Adresse MAC de la carte réseau, réellement unique car déterminé par IEEE (si la machine ne possède pas de carte réseau, un code aléatoire est généré à partir de l'état de la machine – RAM, disque, etc…). Le GUID est également appelé « Universally Unique Identifier » (UUID) selon la dénomination initiale par OSF DCE
IDEF1X	Format graphique de notation d'un modèle de données utilisée notamment par UML. IDEF1x signifie : Icam DEFinition language 1.
Lien	Les liens sont les éléments de jonction entre entité et association. Ils se représentent généralement sous forme de traits dans les différentes modèles.
MCD	Le MCD ou Modèle Conceptuel de Données, est une représentation sous forme de schéma visuel de la mise en relation des données à un niveau abstrait (pas de lien avec l'implémentation qui en sera faite)

MERISE	Méthode d'élaboration de système informatique française introduite dans les années 70 par Hubert Tardieu. La partie modélisation des données repose sur le modèle Entité Relation de Peter Chen.
MLD	Le MLD ou Modèle Logique de Données, est représentation graphique de l'architecture de la base de données sous forme de tables et de liens d'intégrité conçu de manière générique en respectant une norme ou un standard (généralement SQL).
Modèle Conceptuel de Données	Voir MCD
Modèle Logique de Données	Voir MLD
Modèle Physique de Données	Voir MPD
MPD	Le MPD ou Modèle Physique de Données est une représentation graphique de l'architecture de la base de données sous forme de tables et de liens d'intégrité propre à une implémentation sur un SGBDR spécifié.
Paradoxe temporel	action dans un temps passé remis en cause par une action postérieure. Exemple : l'affectation d'une valeur de clef à une ligne de table qui vient à être archivée, ne doit pas être réutilisée pour une autre ligne, sinon la restauration de l'archive écraserait les données afférente à cette ligne.
Schéma	Un schéma SQL est un sous ensemble d'un CATALOG SQL (une base de données) et permet de gérer la sécurité d'accès aux données de manière globale et générique. Le nom qualifié d'un objet SQL commence toujours par son préfixe de schéma.
Trigramme	code composé de trois lettres permettant une identification rapide d'un objet.