Calendrier, timing et horaires en SQL...

Date de publication : 18/04/2004

Par SQLPro

niveau : intermédiaire

La gestion du temps et la manipulation des données temporelles sont les éléments les plus ardus des développements.
Pour mettre toutes les chances de votre côté et passer d'un problème complexe à une solution presque enfantine, je vous propose d'étudier cette méthode basée sur la modélisation d'un planning de dates...

Préambule

La difficulté des calculs portant sur des dates ou des horaires (et parfois les deux) est lié à la codification même de la mesure du temps ainsi qu'au fait que le temps est une entropie (disons le, même une "isentroprie"), c'est à dire qu'il s'écoule de manière uniforme (iso) et dans un seul sens irréversible (vers le futur).

La mesure du temps n'obéit pas à des règles conventionnelles, comme le comptage décimal ou l'origine zéro. En effet :

- Les années comptent tantôt 365 tantôt 366 jours en fait 365 jours 5 heures 48 min. 45,97 sec...
- Les mois de 29 à 31 jours
- Il n'y a pas d'année 0, mais il y a des années négatives (avant JC !)
- Les siècles et millénaires commencent par une année unitaire (1, 1901, 2001...)
- Les jours comptent 24 heures et il y a une heure zéro !
- Les heures comptent 60 minutes et les minutes 60 secondes,
- Il n'y a pas recouvrement éxact des semaines par rapport aux mois...
- Les heures changent par rapport aux différents fuseaux horaires de la planète !
- certaines opérations sur les dates sont impossible (par exemples rajouter exactement 3 mois à une date)...
- le 10 octobre 1582 n'a jamais existé !!!

Tous ces éléments font que les calculs, notamment de comptage du temps obéissent à des algorithmes complexes.

De ce fait, la norme SQL 2, propose une solution générale assez intelligente. Mais elle est rarement implantée. Nous étudierons donc la norme, ce que propose les éditeurs et finalement une solution basée sur des relations entre tables avec une table principale stockant toutes les dates.

NOTA : il est dommage que le calendrier mis en place lors de la révolution française n'ai pas subsisté. L'année y étais divisé en 12 mois de 30 jours. Chaque semaine faisait 10 jours. Les 5 ou 6 jours restant étaient placés à la fin de l'année et constituait des vancances... En quelques sortes, les révolutionnaires étaient en avance sur les congés payés de 1936 et sur les 35 heures de la mère Aubry !

ATTENTION : La numérotation des semaines est standardisée depuis 1976 par l'ISO, avec les règles suivantes :

- Le lundi est considéré comme le premier jour de la semaine.
- Les semaines d'une même année sont numérotées de 01 à 52 (parfois 53).
- La semaine qui porte le numéro 01 est celle qui contient le premier jeudi de

janvier.
- Il peut exister une semaine n°53 (années communes finissant un jeudi, bissextiles finissant un jeudi ou un vendredi).

La numérotation des semaines aux USA, comme la numérotation des jours diffère totalement de cette norme !

1. La solution normative

Elle se compose de trois éléments : des types spécifiques, une algèbre et des opérateurs particuliers.

1.1. Les types temporels SQL 2

La norme propose les types :

```
DATE
TIME [WITH TIME ZONE]
TIMESTAMP [WITH TIME ZONE]
INTERVAL
```

La précision de TIME ZONE permet de définir le décalage du fuseau horaire par rapport à l'heure universelle (UTC).

La norme SQL 2 impose en outre la représentation des dates et heures suivant le masque :

HEURE	DATE	DATE et HEURE
hh:mm:ss.nnn	AAAA-MM-JJ	AAAA-MM-JJ hh:mm:ss.nnn

Avec des dates allant du premier janvier de l'an 1 au 31 décembre de l'an 9999.

NOTA : le typage rapide est permis et même conseillé dans les expression temporelles. En effet comment savoir si 21:16 fait référence à 21h16 ou 21 minutes et 16 secondes ? Pour lever le doute, on peut, plutôt que d'utiliser la fonction CAST, préfixer la donnée :

Exemple :

```
TIME '00:21:16', DATE '2002-04-05'
```

Le type INTERVAL permet de stocker des durées. Sa syntaxe est de la forme :

```
nom_colonne INTERVAL mesure_temps1 [TO mesure_temps2]
```

dans laquelle mesure_temp peut être :

```
YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
```

avec les contraintes suivante :

- mesure_temps1 doit englober mesure_temp2
- mesure_temps2 et toutes les mesures intermédiaires entre mesure_temps1 et mesure_temps2 ne peuvent "déborder".

Exemples :

VALIDE	NON VALIDE
`DUREE1 INTERVAL DAY`	`DUREE2 INTERVAL DAY TO DAY`
`DUREE3 INTERVAL YEAR TO DAY`	`DUREE4 INTERVAL MINUTE TO HOUR`
`DUREE5 INTERVAL HOUR TO MINUTE`	`DUREE6 INTERVAL DAY TO MONTH`
`CAST ('300:5:20' AS INTERVAL HOUR TO SECOND)`	`CAST('10:300:20' AS INTERVAL HOUR TO SECOND)`
`CAST ('2002-09-04 21:16' INTERVAL YEAR TO MINUTE)`	`CAST ('4/9/2002 21h16' INTERVAL YEAR TO MINUTE)`

1.2. Mathématique normative des calculs temporels

1.2.1. Fonctions

Les fonctions **CURRENT_DATE, CURRENT_TIME,CURRENT_TIMESTAMP** permettent de récupérer respectivement la date, l'heure et le combiné date/heure courantes depuis le système. Attention, ce sont des fonctions non déterministes, c'est à dire que ré exécutées plusieurs fois de suite, elles peuvent ne pas donner un résultat identique...

La fonction EXTRACT permet d'extraire une partie temporelle sous forme numérique d'une donnée de type temporel.
La syntaxe de le fonction EXTRACT est la suivante :

```
EXTRACT( {YEAR | MONTH | DAY | HOUR | MINUTE | SECOND } FROM donnée)
```

Exemples :

EXTRACT (MONTH FROM '2002-04-13')	4
EXTRACT (MINUTE FROM '2002-04-13 21:16:11.050')	16

1.2.2. Le prédicat OVERLAPS

Un outil puissant nous est fournit pas SQL avec le prédicat OVERLAPS. Il permet de préciser si une période en recouvre (au moins partiellement) une autre. C'est très pratique si vous voulez gérer un diagramme de GANTT par exemple, ou il convient que certaines tâches ne démarrent pas avant la fin d'autres tâches.

La syntaxe du prédicat OVERLAPS est la suivante :

```
(période1) OVERLAPS (période2)
période ::
borne_debut, borne_fin
borne_debut ::
DATE | TIME | TIMESTAMP
borne_fin ::
DATE | TIME | TIMESTAMP | INTERVAL
```

Avec la contrainte suivante :

- borne_début et borne_fin doivent être de même type dans les périodes sauf si borne_fin est de type INTERVAL.

Exemples :

VALIDE	NON VALIDE
(TIME '08:11:25', TIME '09:25:11') OVERLAPS (TIME '09:11:25', TIME '09:11:25')	TIME '08:11:25', DATE '2002-01-01') OVERLAPS (TIME '09:11:25', TIME '09:11:25')
(DATE '2000-01-01', INTERVAL '100' DAY) OVERLAPS (DATE '2000-04-01', INTERVAL '1' DAY)	(DATE '2000-01-01', INTERVAL '100' DAY) OVERLAPS (TIME '20:04:01', INTERVAL '1' DAY)
(TIMESTAMP '2002-01-01 08:11:25', TIMESTAMP '2002-02-01 21:25:11') OVERLAPS (TIMESTAMP '2002-01-10 09:11:25', INTERVAL '1 01:01:30' DAY TO SECOND)	(TIMESTAMP '2002-01-01 08:11:25', TIMESTAMP '2002-02-01 21:25:11') OVERLAPS (INTERVAL '1 01:01:30' DAY TO SECOND, TIMESTAMP '2002-01-10 09:11:25')

Maintenant penchons nous sur le résultat. OVERLAPS étant un prédicat il ne peut fournir que... 3 valeurs : TRUE, FALSE et UNKNOWN (du fait de la présence possible de marqueur NULL dans les données).

Le prédicat OVERLAPS est vrai si :

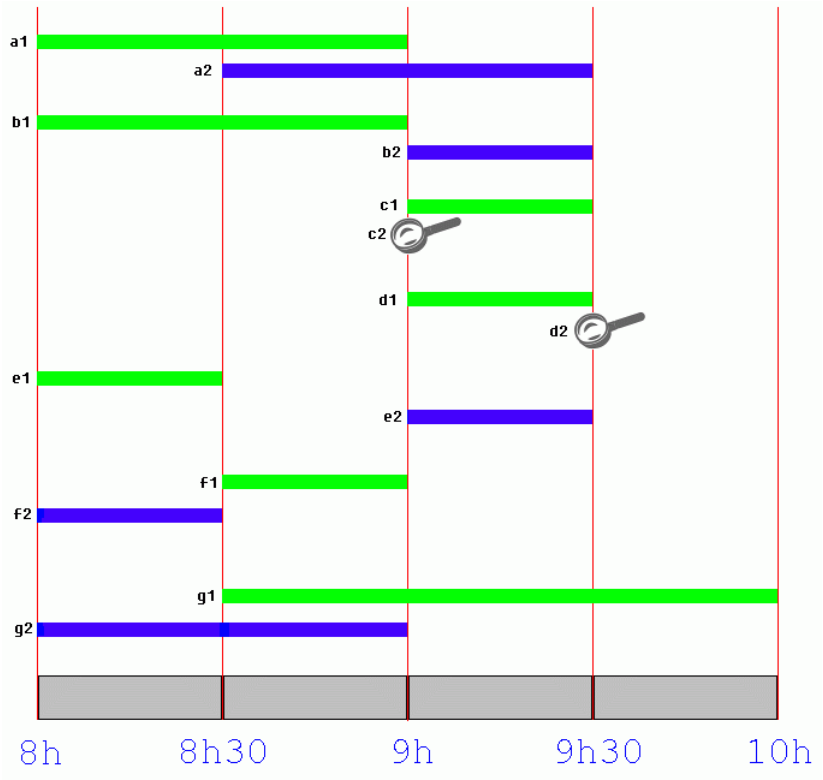
```
((période1.borne_debut > période2.borne_debut
   et (période1.borne_debut < période2.borne_debut ou période1.borne_fin <
période2.borne_fin)) ou
(période2.borne_debut > période1.borne_debut
   et (période2.borne_debut < période1.borne_fin ou période2.borne_fin <
période1.borne_fin)) ou
(période1.borne_debut = période2.borne_debut et (période1.borne_fin NON NUL et
période2.borne_fin NON NUL)
```

Ce qui peut se traduire en bon français par : la période P2 recouvre la période P1, si tout ou partie de la période P2 est inclus dans la période P1.
ATTENTION : un effet de bord due à l'asymétrie du prédicat est à remarquer...

Un bon exemple valant mieux qu'un long discours, voici quelques données permettant de mieux comprendre l'intérêt de ce prédicat. Le jeu d'essais suivant va nous permettre de mieux comprendre...

Les périodes dont définies comme suit :

- **a1** va de 8h à 9h et **a2** de 8h30 à 9h30
- **b1** va de 8h à 9h et **b2** de 9h à 9h30
- **c1** va de 9h à 9h30 et **c2** de 9h à 9h
- **d1** va de 9h à 9h30 et **d2** de 9h30 à 9h30
- **e1** va de 8h à 8h30 et **e2** de 9h à 9h30
- **f1** va de 8h30 à 9h et **f2** de 8h à 8h30
- **g1** va de 8h30 à 10h et **g2** de 8h à 9h



Détails des intervalles de temps c2 et d

On peut les modéliser ainsi :

```
CREATE TABLE PERIODE
(CODE CHAR(1),
 P1_DEBUT TIME,
 P1_FIN   TIME,
 P2_DEBUT TIME,
 P2_FIN   TIME)
```

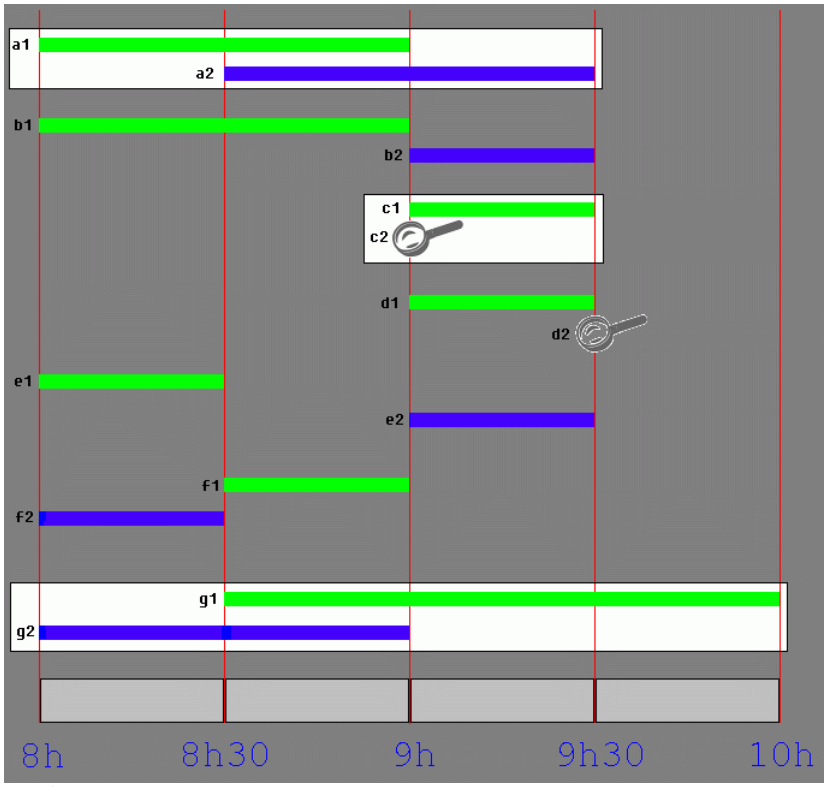
```
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('a', '08:00:00', '09:00:00', '08:30:00', '09:30:00')
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('b', '08:00:00', '09:00:00', '09:00:00', '09:30:00')
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('c', '09:00:00', '09:30:00', '09:00:00', '09:00:00')
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('d', '09:00:00', '09:30:00', '09:30:00', '09:30:00')
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('e', '08:00:00', '08:30:00', '09:00:00', '09:30:00')
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('f', '08:30:00', '09:00:00', '08:00:00', '08:30:00')
INSERT INTO PERIODE (CODE, P1_DEBUT, P1_FIN, P2_DEBUT, P2_FIN)
VALUES ('g', '08:30:00', '10:00:00', '08:00:00', '09:00:00')
```

La requête suivante, donne :

```
SELECT *
FROM   PERIODE
WHERE  P1_DEBUT, P1_FIN OVERLAPS P2_DEBUT, P2_FIN
```

CODE	P1_DEBUT	P1_FIN	P2_DEBUT	P2_FIN
a	08:00:00	09:00:00	08:30:00	09:30:00
c	09:00:00	09:30:00	09:00:00	09:00:00
g	08:30:00	10:00:00	08:00:00	09:00:00

Seules, les périodes des exemples **a**, **c** et **g** répondent à l'opérateur OVERLAPS.



Les périodes recouvrantes sont sur fond blanc, celle non recouvrantes sur fond gris.

L'explication est extraite de mon ouvrage, "SQL", la référence, collection développement, Campus Press Editeur Paris 2001.

"
Par analogie, avec les données que nous venons de voir, il s'agit de considérer des spectateurs qui seraient entrés dans une salle de cinéma pendant la projection d'un film. Le film se serait déroulé en période P1 et chaque spectateur aurait séjourné dans la salle en période P2. Dès lors tout devient clair : le prédicat OVERLAPS permet de savoir qui a vu le film, au moins en partie !
On constate au passage que quelqu'un qui ne reste qu'un temps infiniment court, c'est à dire que la période est caractérisé par le fait que le début est égal à la fin, a vu le film, si cet instant est inclus dans la limite [début, fin[de la période de référence. En effet s'il arrive au début il est considéré comme ayant vu le film tandis que s'il arrive à la fin, il est considéré comme n'ayant pas vu le film. Ceci est logique, mais va à l'encontre de l'intuition... C'est logique parce que le temps s'écoule de manière uniforme dans un seul sens ! On ne peut revenir en arrière et la mesure du temps ne peut qu'augmenter puisqu'allant toujours dans le sens du vieillissement On peut dire que l'écoulement du temps est de nature isentropique, c'est à dire à entropie constante. En fait si on se livre à un calcul mathématique aux limites on peut toujours partir d'un intervalle donné que l'on restreint petit à petit pour arriver à une durée nulle. Si cet instant de durée infime démarre au début d'un autre instant il reste toujours inclus dedans, tandis que s'il démarre à la fin, il ne sera jamais inclus dedans ! Là ou l'affaire se complique, c'est quand une valeur au moins est nulle...
"

Bien entendu en l'absence d'un tel prédicat, vous pouvez le fabriquer de toutes pièces par une construction SQL. En effet reformulé sous SQL, notre prédicat OVERLAPS est :

```
SELECT *
FROM   PERIODE
WHERE  (P1_DEBUT > P2_DEBUT AND (P1_DEBUT < P2_FIN OR P1_FIN < P2_FIN)) OR
       (P2_DEBUT > P1_DEBUT AND (P2_DEBUT < P1_FIN OR P2_FIN < P1_FIN)) OR
       (P1_DEBUT = P2_DEBUT AND (P1_FIN IS NOT NULL AND P2_FIN IS NOT NULL))
```

Et donne, bien évidemment le même résultat que précédemment.

1.2.3. Algèbre temporelle

Il est possible d'utiliser les opérations algébriques + - x et / avec quelques restrictions :

premier opérande	opérateur	second opérande	résultat
TIMESTAMP | DATE | TIME	-	TIMESTAMP | DATE | TIME	INTERVAL
TIMESTAMP | DATE | TIME	+	INTERVAL	TIMESTAMP | DATE | TIME
TIMESTAMP | DATE | TIME	-	INTERVAL	TIMESTAMP | DATE | TIME
INTERVAL	+	TIMESTAMP | DATE | TIME	TIMESTAMP | DATE | TIME
INTERVAL	+	INTERVAL	INTERVAL
INTERVAL	-	INTERVAL	INTERVAL

INTERVAL	*	nombre	INTERVAL
INTERVAL	/	nombre	INTERVAL
nombre	*	INTERVAL	INTERVAL

Exemples :

VALIDE	NON VALIDE
DATE '2002-01-01' - DATE '2001-12-24' :: INTERVAL '8' DAY	DATE '2002-01-01' - TIMESTAMP '2001-12-24 16:12:30' :: ???
TIMESTAMP '2001-12-24 16:12:30' - INTERVAL '8' DAY :: '2001-12-16 16:12:30'	DATE '2001-01-01' - DATE '2002-12-24' :: ???
INTERVAL '7' DAY / 2 :: INTERVAL '3 12:00:00' DAY TO SECOND	DATE '2001-01-01' * INTERVAL '2' DAY

1.2.4. Logique temporelle

Bien entendu la comparaison entre des types temporels est possible, mais dans certaines limites.
Voici les différentes combinaisons possibles :

DATE	[NOT] < <= = >= > <>	DATE
TIME	[NOT] < <= = >= > <>	TIME
TIMESTAMP	[NOT] < <= = >= > <>	TIMESTAMP
INTERVAL YEAR	[NOT] < <= = >= > <>	INTERVAL YEAR
INTERVAL MONTH	[NOT] < <= = >= > <>	INTERVAL MONTH
INTERVAL YEAR TO MONTH	[NOT] < <= = >= > <>	INTERVAL YEAR TO MONTH
INTERVAL DAY	[NOT] < <= = >= > <>	INTERVAL DAY
INTERVAL DAY TO HOUR	[NOT] < <= = >= > <>	INTERVAL DAY TO HOUR
INTERVAL DAY TO MINUTE	[NOT] < <= = >= > <>	INTERVAL DAY TO MINUTE
INTERVAL DAY TO SECOND	[NOT] < <= = >= > <>	INTERVAL DAY TO SECOND

On pourra noter que tout interval comprenant le mois autrement qu'en borne de fin ne peut faire l'objet d'une comparaion. Ceci est du au fait que le nombre de jours d'un mois différe d'un mois à l'autre ce qui rend impossible la comparaison d'intervalle basé sur des durée de mois...

2. Le discours des éditeurs

La complexité de la logique temporelle et du calcul de date ont fait que peu d'éditeurs de SGBDR ont implémenté la norme de manière drastique. Certains même se contentent de ne fournir que le type TIMESTAMP.

Quels sont alors les trucs qu'ils utilisent pour donner satisfaction à leurs clients.

SQL Server de Microsoft, n'inclut que le type DATETIME (équivalent du normatif TIMESTAMP) et propose en sus, les cinq fonctions suivantes : CURRENT_TIMESTAMP, DATEPART, DATEADD, DATEDIFF et DATENAME (nous passerons sous silence les DAY, MONTH, YEAR qui sont virtuellement inclus dans DATEPART).

CURRENT_TIMESTAMP	Date et heure courante
DATEPART	Equivalent du EXTRACT de la norme
DATEADD	Ajout de durée dans date
DATEDIFF	Retrait de durée dans date
DATENAME	Nom d'une partie de date

Intéressons nous aux fonctions DATEADD et DATEDIFF. Leurs syntaxe est :

```
DATEADD | DATEDIFF (partie_de_date, nombre, date)
```

Exemple :

SELECT DATEADD(MONTH, 1, CAST('2002-01-31' AS DATETIME))	2002-02-28 00:00:00.000
SELECT DATEADD(MONTH, -1, DATEADD(MONTH, 1, CAST('2002-01-31' AS DATETIME)))	2002-01-28 00:00:00.000

L'ajout d'un mois au 31 janvier 2002 ne provoque pas un saut à mars, car l'algorithme reprend bien la fin du mois suivant, soit le 28 février. *Bravo SQL Server*.
En revanche la seconde requête est une abération... en effet l'ajout et le retrait imbriqué d'un mois, donne une date décalée de 3 jours. C'est une catastrophe... *Au secours SQL Server* !!!

Néanmoins la solution SQL Server permet des calculs de base sur les données temporelles pour peu que l'on prenne quelques précautions.

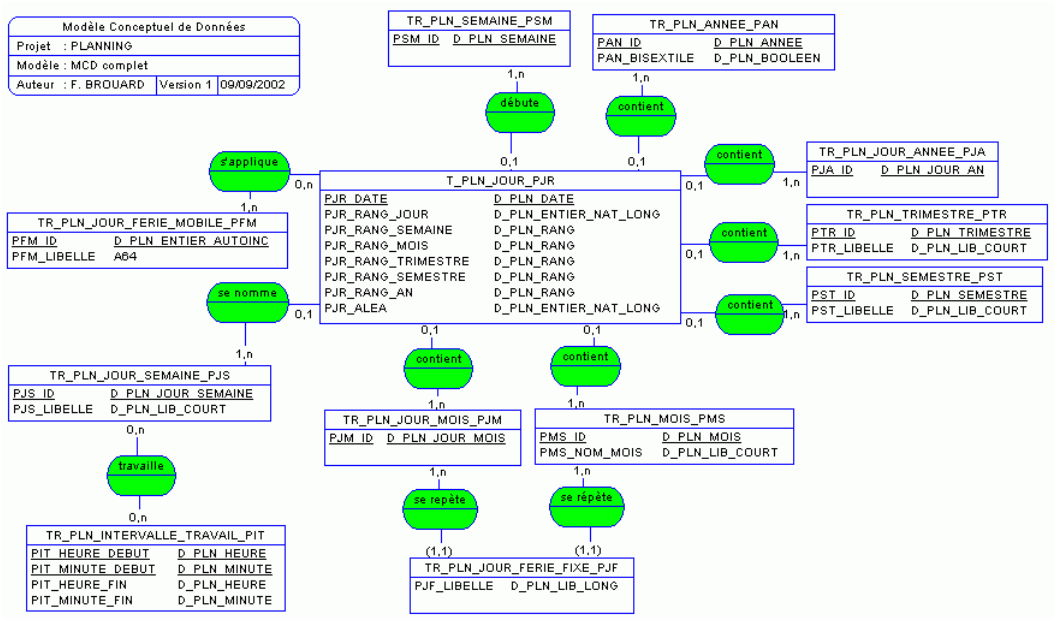
3. La solution intelligente

Elle consiste tout simplement à modéliser un planning avec non seulement une continuité des dates exploitées dans la base mais un ensemble de tables "satellites" ayant chacune un découpage du temps. Autrement dit, autour de la table des dates, une table des mois,

des jours de semaine (de 1 lundi à 7 dimanche), des jours du mois (de 1 à 28, 29, 30 ou 31) une table des années, des semestres, des trimestres, des semaines...

3.1. Le modèle de données

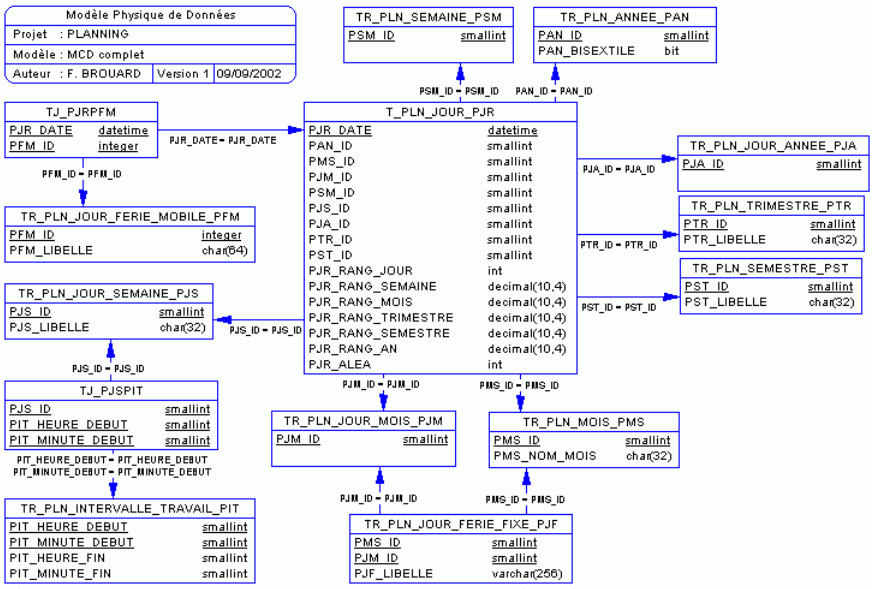
Voici un tel modèle MERISE :



On trouvera dans T_PLN_JOUR_PJR une entité composée d'une colonne clef représentant les dates, une colonne rang numérotant les dates dans leur ordre chronologique à l'aide d'une valeur discrète (entier), mais aussi les semaines, les mois les trimestres les semestres et les années avec des valeurs continues (décimaux) et une colonne alea dont je vous expliquerai un jour l'utilité !

J'y ai rajouté une petite entité "INTERVALLE_TRAVAIL" permettant de définir des plages horaires d'ouverture de l'entreprise. Par exemple de 8h30 à 12h30 et de 14h à 18h du lundi au jeudi et de 9h à 12h et de 14h à 16h le vendredi (cette entité étant liée avec les jours de semaine).

Ce modèle aboutit à la représentation physique suivante :



Où l'on voit que la table T_PLN_JOUR_PJR est garnie de 8 clefs étrangères...

Le secret de ce modèle et de son utilité, ce sont les colonnes PJR_RANG_... de la table des dates. Nous allons voir ce qu'on y met dedans, mais surtout, comment on s'en sert...

3.2. Les données du référentiel

Bien entendu, pour pouvoir fonctionner, un tel modèle doit être garni, c'est à dire que toutes les tables doivent être populées. En annexe vous trouverez les ordres SQL pour créer cette base et la peupler.

Ce script et les deux procédures stockées (les procédures ont été écrites pour SQL Server

mais sont facilement transposable dans le langage procédurale de votre SGBDR ou dans un langage comme DELPHI) permettent de peupler les tables référentielles, TR_* mais aussi une partie de la table principale (T_PLN_JOUR_PJR), notamment les colonnes PJR_DATE (date du jour), PAN_ID (année), PMS_ID (mois), PJM_ID (jour du mois de 1 à 31), PSM_ID (semaine de l'année de 1 à 52 ou 53), PJS_ID (jour de la semaine de 1 à 7), PJA_ID (jour de l'année de 1 à 365 ou 366) PTR_ID (trimestre de l'année) et PST_ID (semestre de l'année).

3.3. Les mesures temporelles

Intéressons nous maintenant aux colonnes PJR_RANG_*.

Notre table est composée de lignes possédant chacune une valeur discrète : celle d'un jour, d'une date dans la continuité du temps.
La colonne PJR_RANG_JOUR est donc numéroté en continue de façon a ce que chaque lendemain soit incrémenté d'une unité. Autrement dit, PJR_RANG_JOUR + 1 équivaut à ajouter un jour... Le type de données sous jacent à PJR_RANG_JOUR est donc un entier. Mais il faut trouver une origine de numérotation. On peut la fixer arbitrairement, par exemple au 14 octobre 1967 (c'est fou ce qu'on trouve à une date aussi arbitraire que celle là lorqu'on cherche sur un moteur du web comme Yahoo [1]) mais l'habitude est de se caler sur le premier janvier 1900 qui constitue le point d'origine de l'axe des dates et par conséquent le jour 0.

Dès lors nous allons avoir la numérotation suivante :

```
DATE           RANG
----------     -----
1900-01-01     1
1900-01-02     2
1900-01-03     3
...
1900-12-31     365
1901-01-01     366
1901-01-02     367
...
1967-10-14     24757
...
1999-12-31     36523          => autrement dit 365 * 100 + 23 soit 23 anées
bisextiles !
2000-01-01     36524
...
2030-12-31     47846
```

Là où notre affaire se complique, c'est pour numéroter chaque jour en fraction de mois, semaine, année...

En fait ce n'est pas très compliqué.
Le premier jour d'une année, par exemple le jour de l'an de 1930 :

- le rang de l'année est **30,0000**
- le rang du mois est 30*12 + 1 = **361,0000**
- le rang de la semaine est **1571, 000**
- le rang du trimestre est 30 * 4 + 1 = **121**
- le rang du semestre est 30 * 2 + 1 = **61**

Pour le rang de la semaine, c'est un plus difficile car il faut compter le nombre de semaines de chaque année, cela pouvant varier entre 52 et 53... Au passage notez les décimales !

Le lendemain de ce jour, au 2 janvier 1930, les rang sont les suivants :

- année : 30 + 1/365 (365, nombre de jours de l'année) = **30,0027**
- mois : 361 + 1/31 (31, nombre de jours du mois) = **361,0323**
- semaine : 1571 + 1/7 (7, nombre de jours de la semaine) = **1571,1429**
- trimestre : 121 + 1/90 (90, nombre de jour du trimestre) = **121,0111**
- semestre : 61 + 1/181 (181, nombre de jours du semestre) = 61,0055

Etc...

Voici une deux des requêtes pour calculer ces nouvelles données. Elle concerne la mise à jour du rang de l'année dans la table T_PLN_JOUR_PJR :

```
-- calcul du rang ANNEE
SELECT PJR_DATE,
       CAST(
       (CAST(PAN_ID AS FLOAT) - 1899.0)
     + (CAST(PJR_RANG_JOUR AS FLOAT)
     - (SELECT CAST(PJR2.PJR_RANG_JOUR AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR2
        WHERE  PJR2.PJM_ID = 1
          AND  PJR2.PMS_ID = 1
          AND  PJR2.PAN_ID = PJR.PAN_ID +1))
     / (SELECT CAST(COUNT(*) AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR3
        WHERE  PJR3.PAN_ID = PJR.PAN_ID)
        AS DECIMAL(10,4)) AS PJR_RANG_AN
FROM T_PLN_JOUR_PJR PJR
```

```
=> le rang de l'année [1900 = 0]
=> plus ( le rang du jour
=>          moins le nombre de jours
            pour aller à l'année
            suivante

=>      ) divisé par le nombre
           de jours écoulés dans l'année
```

```
-- calcul du rang MOIS
SELECT PJR_DATE,
       CAST(
       (CAST(PAN_ID AS FLOAT) - 1900.0) * 12 + PMS_ID + 1
     + (CAST(PJR_RANG_JOUR AS FLOAT)
     - (SELECT CAST(PJR2.PJR_RANG_JOUR AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR2
        WHERE  PJR2.PJM_ID = 1
          AND  PJR2.PMS_ID = (PJR.PMS_ID  % 12) + 1
          AND  PJR2.PAN_ID =
               CASE
               WHEN PJR.PMS_ID + 1 = 13
                    THEN PJR.PAN_ID +1
                    ELSE PJR.PAN_ID
               END))
```

```
=> le rang du mois [12 mois par an]
=> plus ( le rang du jour
=>          moins le nombre de jours
            pour aller au début du
            mois suivant [attention
            au passage de l'année
            suivante]

=>      ) divisé par le nombre
```

```
    / (SELECT CAST(COUNT(*) AS FLOAT)
       FROM   T_PLN_JOUR_PJR PJR3
       WHERE  PJR3.PMS_ID = PJR.PMS_ID
         AND  PJR3.PAN_ID = PJR.PAN_ID)
       AS DECIMAL(10,4)) AS PJR_RANG_MOIS
FROM T_PLN_JOUR_PJR PJR
```

```
        de jours dans le mois

NOTA : j'ai utilisé ici la fonction %
qui fait le modulo.
```

```
-- calcul du rang SEMAINE
SELECT PJR_DATE,
       CAST(
       CAST(PSM_ID AS FLOAT)  + 51
    + (SELECT SUM(MAX_PSM_ID)
       FROM   (SELECT MAX(PSM_ID) AS MAX_PSM_ID
               FROM   T_PLN_JOUR_PJR
               WHERE  PAN_ID < PJR.PAN_ID
               GROUP  BY PAN_ID)
       T )
   + (CAST(PJS_ID AS FLOAT) - 1) / 7
     AS DECIMAL(10,4)) AS PJR_RANG_SEMAINE
FROM T_PLN_JOUR_PJR PJR
```

```
On utilise ici une astuce vu que l'on a toujours
le même nombre de jour dans une semaine.
Il suffit donc de rajouter 1/7 à chaque jour,
les jours étant numéroté de 1 à 7. On a ainsi :
(1 [pour lundi] - 1 ) / 7
(2 [pour mardi] - 1 ) / 7
etc...
Ceci suppose quand même que l'on a bien
numéroté les semaines depuis l'origine
c'est à dire le premier janvier 1900,
sinon il faut y rajouter le nombre de semaines
cumulées que l'on trouve en annexe
```

```
-- calcul du rang TRIMESTRE
SELECT PJR_DATE,
     CAST((CAST(PAN_ID AS FLOAT) - 1900) * 4 + PTR_ID
    + (CAST((SELECT COUNT(*)
             FROM   T_PLN_JOUR_PJR PJR2
             WHERE  PJR2.PTR_ID = PJR.PTR_ID
                AND PJR2.PAN_ID = PJR.PAN_ID
                AND PJR2.PJR_DATE <= PJR.PJR_DATE)
       AS FLOAT) - 1)
    / (CAST((SELECT COUNT(*)
             FROM   T_PLN_JOUR_PJR PJR3
             WHERE  PJR3.PTR_ID = PJR.PTR_ID
                AND PJR3.PAN_ID = PJR.PAN_ID)
      AS FLOAT))
      AS DECIMAL(10,4)) AS RANG_TIMESTRE
FROM T_PLN_JOUR_PJR PJR
```

```
=> nombre de trimestre
=> plus (  nombre de jours écoulé
           depuis le début du trimestre

=>       ) divisé par le nombre de jours
           du trimestre
```

```
-- calcul du rang SEMESTRE
SELECT PJR_DATE,
     CAST((CAST(PAN_ID AS FLOAT) - 1900) * 2 + PST_ID
-- + nombre de jours écoulé
    + (CAST((SELECT COUNT(*)
             FROM   T_PLN_JOUR_PJR PJR2
             WHERE  PJR2.PST_ID = PJR.PST_ID
                AND PJR2.PAN_ID = PJR.PAN_ID
                AND PJR2.PJR_DATE <= PJR.PJR_DATE)
             AS FLOAT) - 1)

-- divisé par le total de jours
    / (CAST((SELECT COUNT(*)
             FROM   T_PLN_JOUR_PJR PJR3
             WHERE  PJR3.PST_ID = PJR.PST_ID
                AND PJR3.PAN_ID = PJR.PAN_ID) AS FLOAT))
      AS DECIMAL(10,4)) AS RANG_SEMESTRE
FROM T_PLN_JOUR_PJR PJR
```

```
Pour les semestre le principe est le même
que pour celui des trimestres.
Le 4 est changé en deux, et la requête
porte sur la colonne PST_ID au lieu de PTR_ID
```

Bien évidemment ces requêtes peuvent être transformées en requêtes de mise à jour dont on trouvera une version en annexe...

Une fois ces données saisie dans notre table, les calculs sur les mesures temporelles deviennent triviaux. Pour nous aider à voir comment cela fonctionne, nous allons ajouter une table de test qui servira pour nos calculs.

Exemple :

```
CREATE TABLE TEST_PLN
(DATE_DEBUT DATE,
 DATE_FIN DATE)
```

```
INSERT INTO TEST_PLN VALUES ('2001-01-15', '2003-05-18')
INSERT INTO TEST_PLN VALUES ('2003-01-15', '2003-01-18')
INSERT INTO TEST_PLN VALUES ('2004-12-24', '2006-12-23')
INSERT INTO TEST_PLN VALUES ('2002-11-11', '2003-12-12')
```

3.3.1. Le nombre de ... jour, mois, anneé, trimestre, semestre

Comment donc obtenir le nombre de... jour, mois, anneé, trimestre, semestre entre deux dates ?

Le nombre de jours s'obtient par sous straction du rang jour. Mais pour faire cette soustraction il faut deux valeurs de rang jour, donc, deux fois la table T_PLN_JOUR_PJR dans la requête :

```
SELECT DATE_DEBUT, DATE_FIN,
       PJR2.PJR_RANG_JOUR - PJR1.PJR_RANG_JOUR AS NOMBRE_JOUR
FROM   TEST_PLN TPN
       INNER JOIN T_PLN_JOUR_PJR PJR1
             ON TPN.DATE_DEBUT = PJR1.PJR_DATE
       INNER JOIN T_PLN_JOUR_PJR PJR2
             ON TPN.DATE_FIN = PJR2.PJR_DATE
```

```
DATE_DEBUT  DATE_FIN    NOMBRE_JOUR
----------- ----------- -----------
2001-01-15  2003-05-18  853
2003-01-15  2003-01-18  3
2004-12-24  2006-12-23  729
2002-11-11  2003-12-12  396
```

Bien entendu, les calculs de nombre de mois, années, etc... sont tout aussi triviaux :

```
SELECT DATE_DEBUT, DATE_FIN,
       PJR2.PJR_RANG_JOUR -      PJR1.PJR_RANG_JOUR      AS NOMBRE_JOUR,
       PJR2.PJR_RANG_AN -        PJR1.PJR_RANG_AN        AS NOMBRE_ANNEE,
       PJR2.PJR_RANG_MOIS -      PJR1.PJR_RANG_MOIS      AS NOMBRE_MOIS,
       PJR2.PJR_RANG_SEMAINE -   PJR1.PJR_RANG_SEMAINE   AS NOMBRE_SEMAINE,
       PJR2.PJR_RANG_TRIMESTRE - PJR1.PJR_RANG_TRIMESTRE AS NOMBRE_TRIMESTRE
FROM   TEST_PLN TPN
       INNER JOIN T_PLN_JOUR_PJR PJR1
             ON TPN.DATE_DEBUT = PJR1.PJR_DATE
       INNER JOIN T_PLN_JOUR_PJR PJR2
             ON TPN.DATE_FIN = PJR2.PJR_DATE
```

```
DATE_DEBUT  DATE_FIN    NOMBRE_JOUR NOMBRE_ANNEE  NOMBRE_MOIS   NOMBRE_SEMAINE
NOMBRE_TRIMESTRE
----------- ----------- ----------- ------------- ------------- -------------- -----
-----------
2001-01-15  2003-05-18  853         2.3369        28.0968       121.8571
9.3609
2003-01-15  2003-01-18  3           .0082         .0968         .4286          .0333
2004-12-24  2006-12-23  729         1.9972        23.9678       105.1429
7.9892
2002-11-11  2003-12-12  396         1.0849        13.0215       56.5714
```

```
4.3369
```

3.3.2. Comment ajouter exactement un deux ou trois mois, et pouvoir retomber sur nos date en les y retranchant ?

L'ajout et la soustraction sont presque aussi simple :

```
-- ajout de 10 jours à DATE DEBUT
SELECT DATE_DEBUT, PJR2.PJR_DATE
FROM   TEST_PLN TPN
       INNER JOIN T_PLN_JOUR_PJR PJR1
             ON TPN.DATE_DEBUT = PJR1.PJR_DATE
       INNER JOIN T_PLN_JOUR_PJR PJR2
             ON PJR2.PJR_RANG_JOUR = PJR1.PJR_RANG_JOUR + 10
```

```
DATE_DEBUT  DATE_PLUS_10_JOUR
----------- -----------------
2001-01-15  2001-01-25
2002-11-11  2002-11-21
2003-01-15  2003-01-25
2004-12-24  2005-01-03
```

```
-- ajout de 5 mois
SELECT DATE_DEBUT, MIN(PJR2.PJR_DATE) AS DATE_PLUS_5_MOIS
FROM   TEST_PLN TPN
       INNER JOIN T_PLN_JOUR_PJR PJR1
             ON TPN.DATE_DEBUT = PJR1.PJR_DATE
       INNER JOIN T_PLN_JOUR_PJR PJR2
             ON PJR2.PJR_RANG_MOIS >= PJR1.PJR_RANG_MOIS + 5
GROUP BY TPN.DATE_DEBUT
```

```
DATE_DEBUT  DATE_PLUS_5_MOIS
----------- -----------------
2001-01-15  2001-06-15
2002-11-11  2003-04-11
2003-01-15  2003-06-15
2004-12-24  2005-05-24
```

```
-- retrait de 3 trimestres
SELECT DATE_DEBUT, MAX(PJR2.PJR_DATE) AS DATE_MOINS_3_TRIMESTRE
FROM   TEST_PLN TPN
       INNER JOIN T_PLN_JOUR_PJR PJR1
             ON TPN.DATE_DEBUT = PJR1.PJR_DATE
       INNER JOIN T_PLN_JOUR_PJR PJR2
             ON PJR2.PJR_RANG_TRIMESTRE < PJR1.PJR_RANG_TRIMESTRE - 3
GROUP BY TPN.DATE_DEBUT
```

```
DATE_DEBUT  DATE_MOINS_3_TRIMESTRE
----------- ----------------------
2001-01-15  2000-04-15
2002-11-11  2002-02-10
2003-01-15  2002-04-15
2004-12-24  2004-03-24
REMARQUE : la différence du jour du mois
dans les dates s'explique par le fait
que les trimestres n'ont pas tous le même
nombre de jours...
```

NOTEZ la différence entre l'ajout de jours, toujours exact parce que sur des entiers, et l'ajout de mois, année, trimestre, etc... qui, opérant sur des nombres réels doit faire l'objet d'une inéquation pour laquelle on recherche le minimum.

Attention donc au signe de cet inéquation et à l'utilisation de l'agrégat :

AJOUT an, mois, semaine, trimestre, semestre	condition >= dans la jointure	condition >= dans la jointure
RETRAIT an, mois, semaine, trimestre, semestre	agrégat MAX dans le SELECT	condition < dans la jointure

Dernier essais, reprenons l'exemple vu avec SQL Server et ses limites de calculs temporels...

```
INSERT INTO TEST_PLN VALUES ('2002-01-31', NULL)
```

```
-- ajout d'un mois
SELECT DATE_DEBUT, MIN(PJR2.PJR_DATE) AS DATE_PLUS_1_MOIS
FROM   TEST_PLN TPN
        INNER JOIN T_PLN_JOUR_PJR PJR1
              ON TPN.DATE_DEBUT = PJR1.PJR_DATE
        INNER JOIN T_PLN_JOUR_PJR PJR2
              ON PJR2.PJR_RANG_MOIS >= PJR1.PJR_RANG_MOIS + 1
GROUP BY TPN.DATE_DEBUT
```

```
DATE_DEBUT  DATE_PLUS_1_MOIS
----------- ----------------
2002-01-31  2002-03-01

REMARQUE : surprise, le mois de février
semble ignoré et l'ajout d'un mois au 31
janvier passe au 1er mars...
```

```
-- retrait d'un mois
SELECT DATE_DEBUT, MAX(PJR2.PJR_DATE) AS DATE_MOINS_1_MOIS
FROM   TEST_PLN TPN
        INNER JOIN T_PLN_JOUR_PJR PJR1
              ON TPN.DATE_DEBUT = PJR1.PJR_DATE
        INNER JOIN T_PLN_JOUR_PJR PJR2
              ON PJR2.PJR_RANG_MOIS < PJR1.PJR_RANG_MOIS - 1
GROUP BY TPN.DATE_DEBUT
```

```
DATE_DEBUT  DATE_MOINS_1_MOIS
----------- -----------------
2002-03-01  2002-01-31

SURPRISE : nous retombons sur nos pattes !
```

3.3.3. Toutes les ... date, mois, années ... entre deux dates

Autre demande qui revient souvent, connaître toutes les dates, les mois, les années entre deux dates :

```
SELECT DISTINCT DATE_DEBUT, PAN_ID, PMS_ID,  DATE_FIN
FROM   TEST_PLN TPN
       INNER JOIN T_PLN_JOUR_PJR PJR
             ON PJR.PJR_DATE BETWEEN TPN.DATE_DEBUT AND TPN.DATE_FIN
ORDER BY DATE_DEBUT, PAN_ID, PMS_ID
```

```
DATE_DEBUT  PAN_ID PMS_ID DATE_FIN
----------- ------ ------ -----------
2001-01-15  2001   1      2003-05-18
2001-01-15  2001   2      2003-05-18
2001-01-15  2001   3      2003-05-18
2001-01-15  2001   4      2003-05-18
2001-01-15  2001   5      2003-05-18
2001-01-15  2001   6      2003-05-18
2001-01-15  2001   7      2003-05-18
2001-01-15  2001   8      2003-05-18
2001-01-15  2001   9      2003-05-18
2001-01-15  2001   10     2003-05-18
2001-01-15  2001   11     2003-05-18
2001-01-15  2001   12     2003-05-18
2001-01-15  2002   1      2003-05-18
2001-01-15  2002   2      2003-05-18
2001-01-15  2002   3      2003-05-18
2001-01-15  2002   4      2003-05-18
2001-01-15  2002   5      2003-05-18
2001-01-15  2002   6      2003-05-18
2001-01-15  2002   7      2003-05-18
2001-01-15  2002   8      2003-05-18
2001-01-15  2002   9      2003-05-18
2001-01-15  2002   10     2003-05-18
2001-01-15  2002   11     2003-05-18
2001-01-15  2002   12     2003-05-18
2001-01-15  2003   1      2003-05-18
2001-01-15  2003   2      2003-05-18
2001-01-15  2003   3      2003-05-18
2001-01-15  2003   4      2003-05-18
2001-01-15  2003   5      2003-05-18
```

```
2002-11-11  2002   11     2003-12-12
2002-11-11  2002   12     2003-12-12
2002-11-11  2003   1      2003-12-12
2002-11-11  2003   2      2003-12-12
2002-11-11  2003   3      2003-12-12
2002-11-11  2003   4      2003-12-12
2002-11-11  2003   5      2003-12-12
2002-11-11  2003   6      2003-12-12
2002-11-11  2003   7      2003-12-12
2002-11-11  2003   8      2003-12-12
2002-11-11  2003   9      2003-12-12
2002-11-11  2003   10     2003-12-12
2002-11-11  2003   11     2003-12-12
2002-11-11  2003   12     2003-12-12

2003-01-15  2003   1      2003-01-18

2004-12-24  2004   12     2006-12-23
2004-12-24  2005   1      2006-12-23
2004-12-24  2005   2      2006-12-23
2004-12-24  2005   3      2006-12-23
2004-12-24  2005   4      2006-12-23
2004-12-24  2005   5      2006-12-23
2004-12-24  2005   6      2006-12-23
2004-12-24  2005   7      2006-12-23
2004-12-24  2005   8      2006-12-23
2004-12-24  2005   9      2006-12-23
2004-12-24  2005   10     2006-12-23
2004-12-24  2005   11     2006-12-23
2004-12-24  2005   12     2006-12-23
2004-12-24  2006   1      2006-12-23
2004-12-24  2006   2      2006-12-23
2004-12-24  2006   3      2006-12-23
2004-12-24  2006   4      2006-12-23
2004-12-24  2006   5      2006-12-23
2004-12-24  2006   6      2006-12-23
2004-12-24  2006   7      2006-12-23
2004-12-24  2006   8      2006-12-23
2004-12-24  2006   9      2006-12-23
2004-12-24  2006   10     2006-12-23
2004-12-24  2006   11     2006-12-23
2004-12-24  2006   12     2006-12-23
```

Cela peut paraître inutile... c'est souvent indispensable.
Voici un exemple qui illustre l'absolu nécessité d'une table de dates. Notre service SAV fait des interventions en principe tous les jours. La table et les données associées sont les suivantes et concerne la semaine allant du lundi 4 mars au vendredi 8 mars :

```
CREATE TABLE T_SAV
(SAV_ID            INTEGER NOT NULL PRIMARY KEY,
 SAV_DATE_INTERV   DATE,
 SAV_NATURE_INTERV VARCHAR(32),
 SAV_RESOLUE       BIT(1) NOT NULL DEFAULT 0)
```

```
INSERT INTO T_SAV VALUES(1, '2002-03-04', 'Imprimante bloquée', 1)
INSERT INTO T_SAV VALUES(2, '2002-03-04', 'Ecran HS', 0)
INSERT INTO T_SAV VALUES(3, '2002-03-05', 'Disque défectueux, changé', 1)
INSERT INTO T_SAV VALUES(4, '2002-03-06', 'Clavier cassé, remplacé', 1)
INSERT INTO T_SAV VALUES(5, '2002-03-06', 'Lecteur CD HS', 0)
INSERT INTO T_SAV VALUES(6, '2002-03-08', 'Base de registre endommagé', 0)
INSERT INTO T_SAV VALUES(7, '2002-03-08', 'Souris encrassée', 1)
INSERT INTO T_SAV VALUES(8, '2002-03-08', 'Lecteur bande encrassé', 1)
```

Notre directeur du SAV veut savoir quel est le volume des interventions pour chaque jour de la semaine, et effectue ma requête suivante :

```
SELECT COUNT(*) AS NOMBRE, SAV_DATE_INTERV
FROM   T_SAV
GROUP BY SAV_DATE_INTERV
```

```
NOMBRE      SAV_DATE_INTERV
----------- --------------
2           2002-03-04
1           2002-03-05
2           2002-03-06
3           2002-03-08
```

Surprise ! La semaine ne compterai que 4 jours du lundi au vendredi ??? Ou est passé le jeudi 7 ? Il n'existe pas dans la table car aucune intervention n'a eût lieu ce jour. Il aurait fallut quand même obtenir cette date avec une valeur 0 pour le nombre d'interventions... Quelque chose comme :

```
NOMBRE      SAV_DATE_INTERV
----------- ----------
2           2002-03-04
1           2002-03-05
2           2002-03-06
0           2002-03-07      <= ligne manquante...
3           2002-03-08
```

Afin de remédier à cette anomalie, il suffit de faire une jointure avec la table des dates :

```
SELECT COUNT(SAV_ID) AS NOMBRE,
       PJR_DATE
FROM   T_SAV
       RIGHT OUTER JOIN T_PLN_JOUR_PJR
            ON PJR_DATE = SAV_DATE_INTERV
WHERE  PJR_DATE BETWEEN ( SELECT MIN(SAV_DATE_INTERV) FROM T_SAV )
                    AND ( SELECT MAX(SAV_DATE_INTERV) FROM T_SAV )
GROUP  BY PJR_DATE
ORDER  BY PJR_DATE
```

```
NOMBRE      PJR_DATE
----------- ----------
2           2002-03-04
1           2002-03-05
2           2002-03-06
0           2002-03-07
3           2002-03-08
```

L'erreur est encore plus criante si l'on tente de mesurer la moyenne du nombre d'intervention par jour :

```
SELECT AVG(CAST(NOMBRE AS FLOAT)) AS MOYENNE_JOUR
FROM   (SELECT COUNT(*) AS NOMBRE, SAV_DATE_INTERV
        FROM   T_SAV
        GROUP BY SAV_DATE_INTERV) T
```

```
MOYENNE_JOUR
------------
2.0
```

Or 8 interventions sur 5 jours, cela représente 1,6 intervention par jour et non 2 ! Une statistique faussée par ce "trou".
Encore une fois la solution nous est fournit par la jointure sur la table des dates :

```
SELECT AVG(CAST(NOMBRE AS FLOAT))
FROM   (SELECT COUNT(SAV_ID) AS NOMBRE,
       PJR_DATE
FROM   T_SAV
       RIGHT OUTER JOIN T_PLN_JOUR_PJR
```

```
MOYENNE_JOUR
------------------
```

ON PJR_DATE = SAV_DATE_INTERV WHERE PJR_DATE BETWEEN (SELECT MIN(SAV_DATE_INTERV) FROM T_SAV) AND (SELECT MAX(SAV_DATE_INTERV) FROM T_SAV) GROUP BY PJR_DATE) T	1.6000000000000001

4. Dates partielles

Il arrive que l'on doive stocker des dates dont certaines parties sont inconnues ou imprécises. Plusieurs solutions sont envisageables : l'utilisation de données séparées pour les éléments composant les dates ou bien l'utilisation d'une fourchette de dates.

4.1. Dates composites

Cette solution consiste à représenter les dates sous la forme de 3 colonnes : AN, MOIS et JOUR. L'important est de spécifier que ces colonnes peuvent être vide... Autrement dit il ne faut pas construire ces colonnes avec l'option NOT NULL.

Exemples :

```
CREATE TABLE T_DATES_PARTIELLES_DTP
(...
 DTP_AN INTEGER,
 DPT_MOIS INTEGER,
 DPT_JOUR INTEGER)
```

A l'insertion comme lors des mises à jour on veillera a ne stocker que les éléments connus des dates.

Voici maintenant comment les requêtes doivent se présenter dans différents cas de figure :

Rechercher les zigzornifles datés du 4/5/1990 :

```
SELECT ...
FROM ...
WHERE DPT_JOUR = 4
  AND DPT_MOIS = 5
  AND DPT_AN = 1990
```

Les zigzornifles datant de mai 1990 :

```
SELECT ...
FROM ...
WHERE DPT_MOIS = 5
  AND DPT_AN = 1990
```

Les zigzornifles de l'an 1990 :

```
SELECT ...
FROM ...
WHERE DPT_AN = 1990
```

Les zigzornifles entre le 4/5/1990 et le 8/7/1990 :

```
SELECT ...
FROM ...
WHERE (COALESCE(DPT_JOUR, 0),
       COALESCE(DPT_MOIS, 0),
       COALESCE(DPT_AN, 0)) >= (4, 5, 1990)
  AND (COALESCE(DPT_JOUR, 0),
       COALESCE(DPT_MOIS, 0),
       COALESCE(DPT_AN, 0)) <= (8, 7, 1990)
```

Mais si votre SGBDR ne supporte pas le constructeur de lignes valuées, alors il faut écrire un équivalent SQL.

```
SELECT ...
FROM ...
WHERE (COALESCE(DPT_JOUR, 0) >= 4
   OR (COALESCE(DPT_JOUR, 0) = 4
      AND COALESCE(DPT_MOIS, 0) >= 5)
   OR (COALESCE(DPT_JOUR, 0) = 4
      AND COALESCE(DPT_MOIS, 0) = 5
      AND COALESCE(DPT_AN, 0)) >= 1990))
  AND (COALESCE(DPT_JOUR, 0) <= 8
   OR (COALESCE(DPT_JOUR, 0) = 8
      AND COALESCE(DPT_MOIS, 0) <= 7)
   OR (COALESCE(DPT_JOUR, 0) = 8
      AND COALESCE(DPT_MOIS, 0) = 7
      AND COALESCE(DPT_AN, 0)) <= 1990))
```

Sur le constructeur de lignes valuées, lire : Constructeur de ligne valuées (ROW VALUE CONSTRUCTOR)

4.2. Fourchette de date

La seconde technique consiste à créer deux colonnes de date afin de définir un intervalle. Voici un exemple du modèle de table :

Exemples :

```
CREATE TABLE T_DATES_INCOMPLETE_DTI
(...
 DTI_DATE_MIN DATE,
 DTI_DATE_MAX DATE)
```

Si la date est complète, on veillera à la recopier dans les deux colonnes. Pour cela on eut s'aider d'un trigger pour qu'en cas de présence du NULL dans la seconde date on

reprenne la valeur de la première date.

A l'insertion, comme à la mise à jour, la partie de date qui est inconnue doit prendre la plage maximale des valeurs. Par exemple si nous voulons insérer un brandouillon daté de juin 1990 sans que nous en connaissions le jour, il convient d'insérer de la manière suivante :

```
INSERT INTO T_DATES_INCOMPLETE_DTI (..., DTI_DATE_MIN DATE, DTI_DATE_MAX DATE)
                            VALUES (..., '1990-06-01'     , '1990-06-30'     )
```

Voici maintenant comment les requêtes doivent se présenter dans différents cas de figure :

Rechercher les brandouillons datés du 4/5/1990 :

```
SELECT ...
FROM ...
WHERE DPI_DATE_MIN = '1990-05-04'
  AND DPT_DATE_MIN = DATE_MAX
```

```
SELECT ...
FROM ...
WHERE DPI_DATE_MAX = '1990-05-04'
  AND DPT_DATE_MIN = DATE_MAX
```

Les deux requêtes devant donner les mêmes résultats.

Les brandouillons datant de mai 1990 :

```
SELECT ...
FROM ...
WHERE DPI_DATE_MIN >= '1990-05-01'
  AND DPI_DATE_MAX <= '1990-05-31'
```

Les brandouillons de l'an 1990 :

```
SELECT ...
FROM ...
WHERE DPI_DATE_MIN >= '1990-01-01'
  AND DPI_DATE_MAX <= '1990-12-31'
```

Les brandouillons entre le 4/5/1990 et le 8/7/1990 :

```
SELECT ...
FROM ...
WHERE DPI_DATE_MIN >= '1990-05-04'
  AND DPI_DATE_MAX <= '1990-07-08'
```

4.3. Affinage

Ces deux modèles possèdent néanmoins un inconvénient... En effet, si notre utilisateur veut reprendre dans sa requête les bidules situées entre le 4/5/1990 et le 8/7/1990, il convient de se demander si un zirzornifle ou un brandouillon daté de mai 1990 sans précision de jour doit figurer dans le résultat !
Car l'abscence de la connaissance du jour, ne signifie nullement qu'il est obligatoirement inférieur au 4 mai 1990...

Dans ce cas, que l'on pourrait apeller "critère externe", lever cette ambiguité peut être fait de différentes manières...

Dans la première représentation (dates composites) une simple modification de la requête suffit :

```
SELECT ...
FROM ...
WHERE (COALESCE(DPT_JOUR, 31),
       COALESCE(DPT_MOIS, 5),
       COALESCE(DPT_AN, 0)) >= (4, 5, 1990)
  AND (COALESCE(DPT_JOUR, 1),
       COALESCE(DPT_MOIS, 7),
       COALESCE(DPT_AN, 0)) <= (8, 7, 1990)
```

Les valeurs du coalesce sont, dans le premier prédicat de filtrage le dernier jour du mois et le mois de la borne basse du filtre, dans le second prédicat le premier jour du mois et le mois de la borne haute du filtre.

Le problème est plus complexe concernat la seconde représentation. En effet l'indication de l'imprécision de la date nous est donné par le fait que DTI_DATE_MIN est différent de DTI_DATE_MAX.
Il faut alors réaliser une requête plus complexe :

```
SELECT ...
FROM ...
-- date exacte
WHERE (DPI_DATE_MIN >= '1990-05-04'
  AND DPI_DATE_MAX <= '1990-07-08'
  AND DPI_DATE_MIN = DPI_DATE_MAX)
-- date partielle
OR (DPI_DATE_MIN >= '1990-05-01'
  AND DPI_DATE_MAX <= '1990-07-31'
  AND DPI_DATE_MIN <> DPI_DATE_MAX)
```

Dans le second prédicat, on reprend les dates de début de mois de la borne basse et de fin de mois de la borne haute si les dates min et max sont différentes.

5. Conclusion

Voici un ensemble de données et ses règles qui marchent de manière irréprochable afin de traiter tous les cas de figure de manipulation portant sur des dates. Bien entendu vous pouvez transformer ces requêtes en procédures stockées voire en fonction

utilisateurs si votre SGBDR en est doté.

Mais, peut être le volume de données à manipuler vous effraye t-il ?
A titre d'indication, sous SQL server v7 le volume des données pour 30 années de dates (2000 à 2030), soit plus de 11 300 lignes, représente : 6 Mo index compris... Pour 130 années, c'est à dire les dates de 1900 à 2030 le volume des données est de 15Mo... (plus de 47 500 lignes). Quand son sait que la clef de cette table est une date et se trouve organisée généralement en cluster, on ne manipule jamais plus de 1 Mo de données si l'on ne fait pas de calculs sur une plage de dates de plus de 5 à 6 ans.

6. ANNEXE - les scripts de création du modèle de planning...

```
/* ============================================================ */
/*   Nom de la base   :  PLN_PLANNING                           */
/*   Nom de SGBD      :  SQL 2 - standard ISO 1992              */
/*   Date de création :  04/08/2002  17:50                      */
/* ============================================================ */

-- version avec domaines

CREATE DOMAIN D_PLN_ANNEE SMALLINT
CONSTRAINT CKD_ANNEE CHECK (VALUE BETWEEN 1 AND 9999)
;

CREATE DOMAIN D_PLN_BOOLEEN bit(1)
;

CREATE DOMAIN D_PLN_DATE DATE
;

CREATE DOMAIN D_PLN_ENTIER_AUTOINC INTEGER
CONSTRAINT CKD_ENTIER CHECK (VALUE >= 1)
;

CREATE DOMAIN D_PLN_ENTIER_NAT_LONG INTEGER
CONSTRAINT CKD_ENTIER_LONG CHECK (VALUE >= 1)
;

CREATE DOMAIN D_PLN_HEURE SMALLINT
CONSTRAINT CKD_HEURE CHECK (VALUE BETWEEN 0 AND 24)
;

CREATE DOMAIN D_PLN_HEURE_DECIMALE FLOAT
CONSTRAINT CKD_HEURE_DEC CHECK (VALUE BETWEEN 0 AND 24)
;

CREATE DOMAIN D_PLN_JOUR_AN SMALLINT
CONSTRAINT CKD_JOUR_AN CHECK (VALUE BETWEEN 1 AND 366)
;

CREATE DOMAIN D_PLN_JOUR_MOIS SMALLINT
CONSTRAINT CKD_JOUR_MOIS CHECK (VALUE BETWEEN 1 AND 31)
;

CREATE DOMAIN D_PLN_JOUR_SEMAINE SMALLINT
CONSTRAINT CKD_JOUR_SEMAINE CHECK (VALUE 1 AND 7)
;

CREATE DOMAIN D_PLN_LIB_COURT CHAR(32)
;

CREATE DOMAIN D_PLN_LIB_LONG VARCHAR(256)
;

CREATE DOMAIN D_PLN_MINUTE SMALLINT
CONSTRAINT CKD_MINUTE CHECK (VALUE BETWEEN 0 AND 60)
;

CREATE DOMAIN D_PLN_MOIS SMALLINT
CONSTRAINT CKD_MOIS CHECK (VALUE BETWEEN 1 AND 12)
;

CREATE DOMAIN D_PLN_SEMAINE SMALLINT
CONSTRAINT CKD_SEMAINE CHECK (VALUE BETWEEN 1 AND 53)
;

CREATE DOMAIN D_PLN_SEMESTRE SMALLINT
CONSTRAINT CKD_SEMESTRE CHECK (VALUE BETWEEN 1 AND 2)
;

CREATE DOMAIN D_PLN_TRIMESTRE SMALLINT
CONSTRAINT CKD_TRIMESTRE  CHECK (VALUE BETWEEN 1 AND 4)
;

CREATE DOMAIN D_PLN_RANG DECIMAL (10,4)
CONSTRAINT CKD_RANG  CHECK (VALUE > 0)
;


/* ============================================================ */
/*   Table : TR_PLN_ANNEE_PAN                                   */
/* ============================================================ */
create table TR_PLN_ANNEE_PAN
(
    PAN_ID              T_D_PLN_ANNEE         not null,
    PAN_BISEXTILE       T_D_PLN_BOOLEEN       not null,
    constraint PK_TR_PLN_ANNEE_PAN primary key (PAN_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_MOIS_PMS                                    */
/* ============================================================ */
create table TR_PLN_MOIS_PMS
(
    PMS_ID              T_D_PLN_MOIS          not null,
    PMS_NOM_MOIS        T_D_PLN_LIB_COURT     not null,
    constraint PK_TR_PLN_MOIS_PMS primary key (PMS_ID)
)
;
```

```
/* ============================================================ */
/*   Table : TR_PLN_JOUR_MOIS_PJM                               */
/* ============================================================ */
create table TR_PLN_JOUR_MOIS_PJM
(
    PJM_ID              T_D_PLN_JOUR_MOIS     not null,
    constraint PK_TR_PLN_JOUR_MOIS_PJM primary key (PJM_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_SEMAINE_PSM                                 */
/* ============================================================ */
create table TR_PLN_SEMAINE_PSM
(
    PSM_ID              T_D_PLN_SEMAINE       not null,
    constraint PK_TR_PLN_SEMAINE_PSM primary key (PSM_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_FERIE_MOBILE_PFM                       */
/* ============================================================ */
create table TR_PLN_JOUR_FERIE_MOBILE_PFM
(
    PFM_ID              T_D_PLN_ENTIER_AUTOINC not null,
    PFM_LIBELLE         char(64)              not null,
    constraint PK_TR_PLN_JOUR_FERIE_MOBILE_PF primary key (PFM_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_SEMAINE_PJS                            */
/* ============================================================ */
create table TR_PLN_JOUR_SEMAINE_PJS
(
    PJS_ID              T_D_PLN_JOUR_SEMAINE  not null,
    PJS_LIBELLE         T_D_PLN_LIB_COURT     not null,
    constraint PK_TR_PLN_JOUR_SEMAINE_PJS primary key (PJS_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_ANNEE_PJA                              */
/* ============================================================ */
create table TR_PLN_JOUR_ANNEE_PJA
(
    PJA_ID              T_D_PLN_JOUR_AN       not null,
    constraint PK_TR_PLN_JOUR_ANNEE_PJA primary key (PJA_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_TRIMESTRE_PTR                               */
/* ============================================================ */
create table TR_PLN_TRIMESTRE_PTR
(
    PTR_ID              T_D_PLN_TRIMESTRE     not null,
    PTR_LIBELLE         T_D_PLN_LIB_COURT     not null,
    constraint PK_TR_PLN_TRIMESTRE_PTR primary key (PTR_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_SEMESTRE_PST                                */
/* ============================================================ */
create table TR_PLN_SEMESTRE_PST
(
    PST_ID              T_D_PLN_SEMESTRE      not null,
    PST_LIBELLE         T_D_PLN_LIB_COURT     not null,
    constraint PK_TR_PLN_SEMESTRE_PST primary key (PST_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_INTERVALLE_TRAVAIL_PIT                      */
/* ============================================================ */
create table TR_PLN_INTERVALLE_TRAVAIL_PIT
(
    PIT_HEURE_DEBUT     T_D_PLN_HEURE         not null,
    PIT_MINUTE_DEBUT    T_D_PLN_MINUTE        not null
        default 0,
    PIT_HEURE_FIN       T_D_PLN_HEURE         not null,
    PIT_MINUTE_FIN      T_D_PLN_MINUTE        not null
        default 0,
    constraint PK_TR_PLN_INTERVALLE_TRAVAIL_P primary key (PIT_HEURE_DEBUT,
PIT_MINUTE_DEBUT)
)
;

/* ============================================================ */
/*   Table : T_PLN_JOUR_PJR                                     */
/* ============================================================ */
create table T_PLN_JOUR_PJR
(
    PJR_DATE            T_D_PLN_DATE          not null,
    PAN_ID              T_D_PLN_ANNEE         null    ,
    PMS_ID              T_D_PLN_MOIS          null    ,
    PJM_ID              T_D_PLN_JOUR_MOIS     null    ,
    PSM_ID              T_D_PLN_SEMAINE       null    ,
    PJS_ID              T_D_PLN_JOUR_SEMAINE  null    ,
    PJA_ID              T_D_PLN_JOUR_AN       null    ,
    PTR_ID              T_D_PLN_TRIMESTRE     null    ,
    PST_ID              T_D_PLN_SEMESTRE      null    ,
    PJR_RANG_JOUR       T_D_PLN_ENTIER_NAT_LONG not null,
    PJR_RANG_SEMAINE    T_D_PLN_RANG          null    ,
    PJR_RANG_MOIS       T_D_PLN_RANG          null    ,
    PJR_RANG_TRIMESTRE  T_D_PLN_RANG          null    ,
    PJR_RANG_SEMESTRE   T_D_PLN_RANG          null    ,
    PJR_RANG_AN         T_D_PLN_RANG          null    ,
    PJR_ALEA            T_D_PLN_ENTIER_NAT_LONG not null,
    constraint PK_T_PLN_JOUR_PJR primary key (PJR_DATE)
)
;
```

```
/* ============================================================ */
/*   Table : TR_PLN_JOUR_FERIE_FIXE_PJF                         */
/* ============================================================ */
create table TR_PLN_JOUR_FERIE_FIXE_PJF
(
    PMS_ID              T_D_PLN_MOIS          not null,
    PJM_ID              T_D_PLN_JOUR_MOIS     not null,
    PJF_LIBELLE         T_D_PLN_LIB_LONG      not null,
    constraint PK_TR_PLN_JOUR_FERIE_FIXE_PJF primary key (PMS_ID, PJM_ID)
)
;

/* ============================================================ */
/*   Table : TJ_PJRPFM                                          */
/* ============================================================ */
create table TJ_PJRPFM
(
    PJR_DATE            T_D_PLN_DATE          not null,
    PFM_ID              T_D_PLN_ENTIER_AUTOINC not null,
    constraint PK_TJ_PJRPFM primary key (PJR_DATE, PFM_ID)
)
;

/* ============================================================ */
/*   Table : TJ_PJSPIT                                          */
/* ============================================================ */
create table TJ_PJSPIT
(
    PJS_ID              T_D_PLN_JOUR_SEMAINE  not null,
    PIT_HEURE_DEBUT     T_D_PLN_HEURE         not null,
    PIT_MINUTE_DEBUT    T_D_PLN_MINUTE        not null
        default 0,
    constraint PK_TJ_PJSPIT primary key (PJS_ID, PIT_HEURE_DEBUT, PIT_MINUTE_DEBUT)
)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPAN_TR_PLN_A foreign key  (PAN_ID)
       references TR_PLN_ANNEE_PAN (PAN_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPMS_TR_PLN_M foreign key  (PMS_ID)
       references TR_PLN_MOIS_PMS (PMS_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPJM_TR_PLN_J foreign key  (PJM_ID)
       references TR_PLN_JOUR_MOIS_PJM (PJM_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPSM__TR_PLN_S foreign key  (PSM_ID)
       references TR_PLN_SEMAINE_PSM (PSM_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPJS_TR_PLN_J foreign key  (PJS_ID)
       references TR_PLN_JOUR_SEMAINE_PJS (PJS_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPJA_TR_PLN_J foreign key  (PJA_ID)
       references TR_PLN_JOUR_ANNEE_PJA (PJA_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPTR_TR_PLN_T foreign key  (PTR_ID)
       references TR_PLN_TRIMESTRE_PTR (PTR_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPST_TR_PLN_S foreign key  (PST_ID)
       references TR_PLN_SEMESTRE_PST (PST_ID)
;

alter table TR_PLN_JOUR_FERIE_FIXE_PJF
    add constraint FK_TR_PLN_J_L_PJFPJM_TR_PLN_J foreign key  (PJM_ID)
       references TR_PLN_JOUR_MOIS_PJM (PJM_ID)
;

alter table TR_PLN_JOUR_FERIE_FIXE_PJF
    add constraint FK_TR_PLN_J_L_PJFPMS_TR_PLN_M foreign key  (PMS_ID)
       references TR_PLN_MOIS_PMS (PMS_ID)
;

alter table TJ_PJRPFM
    add constraint FK_TJ_PJRPF_L_PJRPFM_T_PLN_JO foreign key  (PJR_DATE)
       references T_PLN_JOUR_PJR (PJR_DATE)
;

alter table TJ_PJRPFM
    add constraint FK_TJ_PJRPF_L_PFMPJR_TR_PLN_J foreign key  (PFM_ID)
       references TR_PLN_JOUR_FERIE_MOBILE_PFM (PFM_ID)
;

alter table TJ_PJSPIT
    add constraint FK_TJ_PJSPI_L_PJSPIT_TR_PLN_J foreign key  (PJS_ID)
       references TR_PLN_JOUR_SEMAINE_PJS (PJS_ID)
;

alter table TJ_PJSPIT
    add constraint FK_TJ_PJSPI_L_PITPJS_TR_PLN_I foreign key  (PIT_HEURE_DEBUT,
PIT_MINUTE_DEBUT)
       references TR_PLN_INTERVALLE_TRAVAIL_PIT (PIT_HEURE_DEBUT, PIT_MINUTE_DEBUT)
;

/* n'oubliez pas d'indexer toutes les colonnes constituant clefs primaires et clefs
étrangères !!! */

/* ============================================================ */
/*   Nom de la base   :  PLN_PLANNING                           */
/*   Nom de SGBD      :  SQL 2 - standard ISO 1992              */
/*   Date de création :  04/08/2002  17:50                      */
/* ============================================================ */
```

```
-- version sans les domaines (les contraintes de domaine sont
-- reportées en contraintes de table)

/* ============================================================ */
/*   Table : TR_PLN_ANNEE_PAN                                   */
/* ============================================================ */
create table TR_PLN_ANNEE_PAN
(
    PAN_ID              smallint              not null
        constraint CKC_PAN_ID_TR_PLN_A check (PAN_ID between 1 and 9999),
    PAN_BISEXTILE       bit                   not null,
    constraint PK_TR_PLN_ANNEE_PAN primary key (PAN_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_MOIS_PMS                                    */
/* ============================================================ */
create table TR_PLN_MOIS_PMS
(
    PMS_ID              smallint              not null
        constraint CKC_PMS_ID_TR_PLN_M check (PMS_ID between 1 and 12),
    PMS_NOM_MOIS        char(32)              not null,
    constraint PK_TR_PLN_MOIS_PMS primary key (PMS_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_MOIS_PJM                               */
/* ============================================================ */
create table TR_PLN_JOUR_MOIS_PJM
(
    PJM_ID              smallint              not null
        constraint CKC_PJM_ID_TR_PLN_J check (PJM_ID between 1 and 31),
    constraint PK_TR_PLN_JOUR_MOIS_PJM primary key (PJM_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_SEMAINE_PSM                                 */
/* ============================================================ */
create table TR_PLN_SEMAINE_PSM
(
    PSM_ID              smallint              not null
        constraint CKC_PSM_ID_TR_PLN_S check (PSM_ID between 1 and 53),
    constraint PK_TR_PLN_SEMAINE_PSM primary key (PSM_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_FERIE_MOBILE_PFM                       */
/* ============================================================ */
create table TR_PLN_JOUR_FERIE_MOBILE_PFM
(
    PFM_ID              integer               not null
        constraint CKC_PFM_ID_TR_PLN_J check (PFM_ID >= 1),
    PFM_LIBELLE         char(64)              not null,
    constraint PK_TR_PLN_JOUR_FERIE_MOBILE_PF primary key (PFM_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_SEMAINE_PJS                            */
/* ============================================================ */
create table TR_PLN_JOUR_SEMAINE_PJS
(
    PJS_ID              smallint              not null
        constraint CKC_PJS_ID_TR_PLN_J check (PJS_ID between 1 and 7),
    PJS_LIBELLE         char(32)              not null,
    constraint PK_TR_PLN_JOUR_SEMAINE_PJS primary key (PJS_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_JOUR_ANNEE_PJA                              */
/* ============================================================ */
create table TR_PLN_JOUR_ANNEE_PJA
(
    PJA_ID              smallint              not null
        constraint CKC_PJA_ID_TR_PLN_J check (PJA_ID between 1 and 366),
    constraint PK_TR_PLN_JOUR_ANNEE_PJA primary key (PJA_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_TRIMESTRE_PTR                               */
/* ============================================================ */
create table TR_PLN_TRIMESTRE_PTR
(
    PTR_ID              smallint              not null
        constraint CKC_PTR_ID_TR_PLN_T check (PTR_ID between 1 and 4),
    PTR_LIBELLE         char(32)              not null,
    constraint PK_TR_PLN_TRIMESTRE_PTR primary key (PTR_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_SEMESTRE_PST                                */
/* ============================================================ */
create table TR_PLN_SEMESTRE_PST
(
    PST_ID              smallint              not null
        constraint CKC_PST_ID_TR_PLN_S check (PST_ID between 1 and 2),
    PST_LIBELLE         char(32)              not null,
    constraint PK_TR_PLN_SEMESTRE_PST primary key (PST_ID)
)
;

/* ============================================================ */
/*   Table : TR_PLN_INTERVALLE_TRAVAIL_PIT                      */
/* ============================================================ */
create table TR_PLN_INTERVALLE_TRAVAIL_PIT
(
    PIT_HEURE_DEBUT     smallint              not null
        constraint CKC_PIT_HEURE_DEBUT_TR_PLN_I check (PIT_HEURE_DEBUT between 0
and 24),
    PIT_MINUTE_DEBUT    smallint              not null
```

```
        default 0
        constraint CKC_PIT_MINUTE_DEBUT_TR_PLN_I check (PIT_MINUTE_DEBUT between 0
and 60),
    PIT_HEURE_FIN       smallint              not null
        constraint CKC_PIT_HEURE_FIN_TR_PLN_I check (PIT_HEURE_FIN between 0 and
24),
    PIT_MINUTE_FIN      smallint              not null
        default 0
        constraint CKC_PIT_MINUTE_FIN_TR_PLN_I check (PIT_MINUTE_FIN between 0 and
60),
    constraint PK_TR_PLN_INTERVALLE_TRAVAIL_P primary key (PIT_HEURE_DEBUT,
PIT_MINUTE_DEBUT)
)
;

/* ============================================================ */
/*   Table : T_PLN_JOUR_PJR                                     */
/* ============================================================ */
create table T_PLN_JOUR_PJR
(
    PJR_DATE            datetime              not null,
    PAN_ID              smallint              null
        constraint CKC_PAN_ID_T_PLN_JO check (PAN_ID between 1 and 9999),
    PMS_ID              smallint              null
        constraint CKC_PMS_ID_T_PLN_JO check (PMS_ID between 1 and 12),
    PJM_ID              smallint              null
        constraint CKC_PJM_ID_T_PLN_JO check (PJM_ID between 1 and 31),
    PSM_ID              smallint              null
        constraint CKC_PSM_ID_T_PLN_JO check (PSM_ID between 1 and 53),
    PJS_ID              smallint              null
        constraint CKC_PJS_ID_T_PLN_JO check (PJS_ID between 1 and 7),
    PJA_ID              smallint              null
        constraint CKC_PJA_ID_T_PLN_JO check (PJA_ID between 1 and 366),
    PTR_ID              smallint              null
        constraint CKC_PTR_ID_T_PLN_JO check (PTR_ID between 1 and 4),
    PST_ID              smallint              null
        constraint CKC_PST_ID_T_PLN_JO check (PST_ID between 1 and 2),
    PJR_RANG            int                   not null
        constraint CKC_PJR_RANG_T_PLN_JO check (PJR_RANG >= 1),
    PJR_RANG_SEMAINE    decimal(10,4)         null    ,
    PJR_RANG_MOIS       decimal(10,4)         null    ,
    PJR_RANG_TRIMESTRE  decimal(10,4)         null    ,
    PJR_RANG_SEMESTRE   decimal(10,4)         null    ,
    PJR_RANG_AN         decimal(10,4)         null    ,
    PJR_ALEA            int                   not null
        constraint CKC_PJR_ALEA_T_PLN_JO check (PJR_ALEA >= 1),
    constraint PK_T_PLN_JOUR_PJR primary key (PJR_DATE)
)
;


/* ============================================================ */
/*   Table : TR_PLN_JOUR_FERIE_FIXE_PJF                         */
/* ============================================================ */
create table TR_PLN_JOUR_FERIE_FIXE_PJF
(
    PMS_ID              smallint              not null
        constraint CKC_PMS_ID_TR_PLN_J check (PMS_ID between 1 and 12),
    PJM_ID              smallint              not null
        constraint CKC_PJM_ID_TR_PLN_J check (PJM_ID between 1 and 31),
    PJF_LIBELLE         varchar(256)          not null,
    constraint PK_TR_PLN_JOUR_FERIE_FIXE_PJF primary key (PMS_ID, PJM_ID)
)
;

/* ============================================================ */
/*   Table : TJ_PJRPFM                                          */
/* ============================================================ */
create table TJ_PJRPFM
(
    PJR_DATE            datetime              not null,
    PFM_ID              integer               not null
        constraint CKC_PFM_ID_TJ_PJRPF check (PFM_ID >= 1),
    constraint PK_TJ_PJRPFM primary key (PJR_DATE, PFM_ID)
)
;

/* ============================================================ */
/*   Table : TJ_PJSPIT                                          */
/* ============================================================ */
create table TJ_PJSPIT
(
    PJS_ID              smallint              not null
        constraint CKC_PJS_ID_TJ_PJSPI check (PJS_ID between 1 and 7),
    PIT_HEURE_DEBUT     smallint              not null
        constraint CKC_PIT_HEURE_DEBUT_TJ_PJSPI check (PIT_HEURE_DEBUT between 0
and 24),
    PIT_MINUTE_DEBUT    smallint              not null
        default 0
        constraint CKC_PIT_MINUTE_DEBUT_TJ_PJSPI check (PIT_MINUTE_DEBUT between 0
and 60),
    constraint PK_TJ_PJSPIT primary key (PJS_ID, PIT_HEURE_DEBUT, PIT_MINUTE_DEBUT)
)
;


alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPAN_TR_PLN_A foreign key  (PAN_ID)
       references TR_PLN_ANNEE_PAN (PAN_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPMS_TR_PLN_M foreign key  (PMS_ID)
       references TR_PLN_MOIS_PMS (PMS_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPJM_TR_PLN_J foreign key  (PJM_ID)
       references TR_PLN_JOUR_MOIS_PJM (PJM_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPSM__TR_PLN_S foreign key  (PSM_ID)
       references TR_PLN_SEMAINE_PSM (PSM_ID)
;

alter table T_PLN_JOUR_PJR
```

```
    add constraint FK_T_PLN_JO_L_PJRPJS_TR_PLN_J foreign key  (PJS_ID)
       references TR_PLN_JOUR_SEMAINE_PJS (PJS_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPJA_TR_PLN_J foreign key  (PJA_ID)
       references TR_PLN_JOUR_ANNEE_PJA (PJA_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPTR_TR_PLN_T foreign key  (PTR_ID)
       references TR_PLN_TRIMESTRE_PTR (PTR_ID)
;

alter table T_PLN_JOUR_PJR
    add constraint FK_T_PLN_JO_L_PJRPST_TR_PLN_S foreign key  (PST_ID)
       references TR_PLN_SEMESTRE_PST (PST_ID)
;

alter table TR_PLN_JOUR_FERIE_FIXE_PJF
    add constraint FK_TR_PLN_J_L_PJFPJM_TR_PLN_J foreign key  (PJM_ID)
       references TR_PLN_JOUR_MOIS_PJM (PJM_ID)
;

alter table TR_PLN_JOUR_FERIE_FIXE_PJF
    add constraint FK_TR_PLN_J_L_PJFPMS_TR_PLN_M foreign key  (PMS_ID)
       references TR_PLN_MOIS_PMS (PMS_ID)
;

alter table TJ_PJRPFM
    add constraint FK_TJ_PJRPF_L_PJRPFM_T_PLN_JO foreign key  (PJR_DATE)
       references T_PLN_JOUR_PJR (PJR_DATE)
;

alter table TJ_PJRPFM
    add constraint FK_TJ_PJRPF_L_PFMPJR_TR_PLN_J foreign key  (PFM_ID)
       references TR_PLN_JOUR_FERIE_MOBILE_PFM (PFM_ID)
;

alter table TJ_PJSPIT
    add constraint FK_TJ_PJSPI_L_PJSPIT_TR_PLN_J foreign key  (PJS_ID)
       references TR_PLN_JOUR_SEMAINE_PJS (PJS_ID)
;

alter table TJ_PJSPIT
    add constraint FK_TJ_PJSPI_L_PITPJS_TR_PLN_I foreign key  (PIT_HEURE_DEBUT,
PIT_MINUTE_DEBUT)
       references TR_PLN_INTERVALLE_TRAVAIL_PIT (PIT_HEURE_DEBUT, PIT_MINUTE_DEBUT)
;

/* n'oubliez pas d'indexer toutes les colonnes constituant clefs primaires et clefs
étrangères !!! */

/* ============================================================ */
/*   ALIMENTATION DES REFERENCES DE LA BASE PLANNING            */
/* ============================================================ */
/* ============================================================ */
/*   ALIMENTATION DES REFERENCES DE LA BASE PLANNING            */
/* ============================================================ */

/* insertion des données table TR_PLN_MOIS_PMS            */
INSERT INTO TR_PLN_MOIS_PMS VALUES (1, 'janvier')
INSERT INTO TR_PLN_MOIS_PMS VALUES (2, 'février')
INSERT INTO TR_PLN_MOIS_PMS VALUES (3, 'mars')
INSERT INTO TR_PLN_MOIS_PMS VALUES (4, 'avril')
INSERT INTO TR_PLN_MOIS_PMS VALUES (5, 'mai')
INSERT INTO TR_PLN_MOIS_PMS VALUES (6, 'juin')
INSERT INTO TR_PLN_MOIS_PMS VALUES (7, 'juillet')
INSERT INTO TR_PLN_MOIS_PMS VALUES (8, 'août')
INSERT INTO TR_PLN_MOIS_PMS VALUES (9, 'septembre')
INSERT INTO TR_PLN_MOIS_PMS VALUES (10, 'octobre')
INSERT INTO TR_PLN_MOIS_PMS VALUES (11, 'novembre')
INSERT INTO TR_PLN_MOIS_PMS VALUES (12, 'décembre')


/* insertion des données table TR_PLN_JOUR_SEMAINE_JSM      */

INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (1, 'lundi')
INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (2, 'mardi')
INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (3, 'mercredi')
INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (4, 'jeudi')
INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (5, 'vendredi')
INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (6, 'samedi')
INSERT INTO TR_PLN_JOUR_SEMAINE_PJS VALUES (7, 'dimanche')

/* insertion des données table TR_PLN_JOUR_MOIS_PJM          */
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (1)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (2)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (3)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (4)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (5)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (6)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (7)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (8)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (9)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (10)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (11)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (12)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (13)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (14)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (15)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (16)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (17)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (18)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (19)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (20)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (21)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (22)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (23)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (24)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (25)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (26)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (27)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (28)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (29)
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (30)
```

```
INSERT INTO TR_PLN_JOUR_MOIS_PJM VALUES (31)

/* insertion des données table TR_PLN_JOUR_FERIE_MOBILE_PFM     */
INSERT INTO TR_PLN_JOUR_FERIE_MOBILE_PFM (PFM_ID, PFM_LIBELLE) VALUES (1, 'Pâques')
INSERT INTO TR_PLN_JOUR_FERIE_MOBILE_PFM (PFM_ID, PFM_LIBELLE) VALUES
(2, 'Ascension')
INSERT INTO TR_PLN_JOUR_FERIE_MOBILE_PFM (PFM_ID, PFM_LIBELLE) VALUES
(3, 'Pentecôte')

/* insertion des données table  TR_PLN_JOUR_FERIE_FIXE_PJF      */
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (1,  1,  'jour de l''an')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (5,  1,  'fête du travail')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (5,  8,  'armistice 1945')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (7,  14, 'fête nationale')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (8,  15, 'Assomption')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (11, 1,  'Toussaint (fête des morts)')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (11, 11, 'armistice de 1918')
INSERT INTO TR_PLN_JOUR_FERIE_FIXE_PJF VALUES (12, 25, 'Noël')

/* insertion des données table  TR_PLN_TRIMESTRE_PTR       */
INSERT INTO TR_PLN_TRIMESTRE_PTR VALUES (1,'Premier trimestre')
INSERT INTO TR_PLN_TRIMESTRE_PTR VALUES (2,'Second trimestre')
INSERT INTO TR_PLN_TRIMESTRE_PTR VALUES (3,'Troisième trimestre')
INSERT INTO TR_PLN_TRIMESTRE_PTR VALUES (4,'Quatrième trimestre')

/* insertion des données table  TR_PLN_SEMESTRE_PST       */
INSERT INTO TR_PLN_SEMESTRE_PST VALUES (1,'Premier semestre')
INSERT INTO TR_PLN_SEMESTRE_PST VALUES (2,'Second semestre')

/*************************************************************************
 SCRIPTS TRANSACT SQL DES PROCÉDURES STOCKÉES POUR ALIMENTATION DES DONNÉES
*************************************************************************/

CREATE PROCEDURE SP_PLN_CREATE_ONE_YEAR @AN INTEGER
AS

/*************************************************************************
  TITRE    : PROCEDURE STOCKÉE MS SQL Server "SP_PLN_CREATE_ONE_YEAR"
  AUTEUR   : Frédéric BROUARD - 2002-09-06
  ARGUMENT : @AN INTEGER entier représentant l'année du calendrier dont les
             dates sont à stocker
  APPELÉE  : par la procédure stockée SP_PLN_CREATE_YEAR qui vérifie les
             années à créer
*************************************************************************/

-- NOTA les dates sont calculées pour une seule année en débordant
-- éventuellement de quelques jours pour stocker des semaines entières

DECLARE @JOUR DATETIME
DECLARE @JOUR_DE_LAN DATETIME
DECLARE @JOUR_DEBUT DATETIME
DECLARE @JOUR_FIN DATETIME
DECLARE @i integer
DECLARE @JOUR_FIN_SEMAINE DATETIME
DECLARE @leapYear bit
DECLARE @ALEA INTEGER

SET NOCOUNT ON

-- indique de commencer la numérotation des jours de semaine à lundi
SET DATEFIRST 1

-- foamt ISO des dates
SET DATEFORMAT YMD

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION TRAN_INS_DATES

-- création des dates
SET @JOUR_DE_LAN = CAST(CAST(@AN AS VARCHAR(4)) + '-01-01' AS DATETIME)
SET @JOUR_DEBUT = DATEADD(DAY, -6,  @JOUR_DE_LAN) -- soit le 26 décembre an - 1
SET @JOUR_FIN = CAST(CAST(@AN + 1 AS VARCHAR(4)) + '-01-01' AS DATETIME)
SET @JOUR_FIN = DATEADD(DAY, 5,  @JOUR_FIN)    -- soit le 6 janvier an + 1
SET @JOUR = @JOUR_DEBUT

-- vérification de présence des années pour débordement des dates
-- l'année avant @AN existe t-elle ?
SET @i = DATEPART(YEAR, @JOUR_DEBUT)
IF NOT EXISTS(SELECT *
              FROM   TR_PLN_ANNEE_PAN
              WHERE  PAN_ID =  @i)
BEGIN
   IF @i % 4 = 0
      SET @leapYear = 1
   ELSE
      SET @leapYear = 0
   INSERT INTO TR_PLN_ANNEE_PAN
   VALUES (@i, @leapYear)
   IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END
END
-- l'année après @AN existe t-elle ?
SET @i = DATEPART(YEAR, @JOUR_FIN)
IF NOT EXISTS(SELECT *
              FROM   TR_PLN_ANNEE_PAN
              WHERE  PAN_ID =  @i)
BEGIN
   IF @i % 4 = 0
      SET @leapYear = 1
   ELSE
      SET @leapYear = 0
   INSERT INTO TR_PLN_ANNEE_PAN
   VALUES (@i, @leapYear)
   IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END
END

-- création de l'année @AN
IF NOT EXISTS(SELECT *
              FROM   TR_PLN_ANNEE_PAN
              WHERE  PAN_ID = @AN)
BEGIN
   IF @AN % 4 = 0
      SET @leapYear = 1
   ELSE
      SET @leapYear = 0
   INSERT INTO TR_PLN_ANNEE_PAN
   VALUES (@AN, @leapYear)
```

```
   IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END
END


WHILE @JOUR < @JOUR_FIN
BEGIN
   SET @ALEA = RAND() * 100000
   IF NOT EXISTS (SELECT *
                  FROM T_PLN_JOUR_PJR
                  WHERE PJR_DATE = @JOUR)
   BEGIN
      INSERT INTO T_PLN_JOUR_PJR (PJR_DATE, PAN_ID, PMS_ID, PJM_ID, PJS_ID, PJA_ID,
PJR_RANG_JOUR, PJR_ALEA)
      VALUES (@JOUR,
              DATEPART(YEAR, @JOUR),
              DATEPART(MONTH, @JOUR),
              DATEPART(DAY, @JOUR),
              DATEPART(WEEKDAY, @JOUR),
              DATEPART(DAYOFYEAR, @JOUR),
              CAST(@JOUR AS INTEGER),
              @ALEA)
      IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END
   END
   SET @JOUR = DATEADD(DAY, 1,  @JOUR)
END

/* Update des trimestres et semestres */

UPDATE T_PLN_JOUR_PJR
SET    PTR_ID = 1, PST_ID = 1
WHERE  DATEPART(MONTH, PJR_DATE) BETWEEN 1 AND 3
  AND  PAN_ID = @AN AND PTR_ID IS NULL
IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END

UPDATE T_PLN_JOUR_PJR
SET    PTR_ID = 2, PST_ID = 1
WHERE  DATEPART(MONTH, PJR_DATE) BETWEEN 4 AND 6
  AND  PAN_ID = @AN AND PTR_ID IS NULL
IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END

UPDATE T_PLN_JOUR_PJR
SET    PTR_ID = 3, PST_ID = 2
WHERE  DATEPART(MONTH, PJR_DATE) BETWEEN 7 AND 9
  AND  PAN_ID = @AN AND PTR_ID IS NULL
IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END

UPDATE T_PLN_JOUR_PJR
SET    PTR_ID = 4, PST_ID = 2
WHERE  DATEPART(MONTH, PJR_DATE) BETWEEN 10 AND 12
  AND  PAN_ID = @AN AND PTR_ID IS NULL
IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END


-- création des semaines de l'année
-- calage premier jour de semaine
SET @JOUR = @JOUR_DE_LAN
IF DATEPART(WEEKDAY, @JOUR) > 4
BEGIN
   WHILE DATEPART(WEEKDAY, @JOUR) <> 1
      SET @JOUR = DATEADD(DAY, +1, @JOUR)
END
ELSE
BEGIN
   WHILE DATEPART(WEEKDAY, @JOUR) <> 1
      SET @JOUR = DATEADD(DAY, -1, @JOUR)
END
SET @i = 1
WHILE DATEADD(DAY, 7, @JOUR) < @JOUR_FIN
BEGIN
   UPDATE T_PLN_JOUR_PJR
   SET PSM_ID = @i
   WHERE PJR_DATE BETWEEN @JOUR AND DATEADD(DAY, 6,  @JOUR)
   IF @@ERROR <> 0 BEGIN GOTO LBL_ERROR RETURN END
   SET @JOUR = DATEADD(DAY, 7,  @JOUR)
   SET @i = @i + 1
END

SET NOCOUNT OFF
COMMIT TRANSACTION TRAN_INS_DATES

RETURN

LBL_ERROR:
ROLLBACK TRANSACTION TRAN_INS_DATES

GO


/*************************************************************************
  TITRE    : PROCEDURE STOCKÉE MS SQL Server "SP_PLN_CREATE_YEAR"
  AUTEUR   : Frédéric BROUARD - 2002-09-06
  ARGUMENT : @AN INTEGER entier dans les limtes 1900 / 200 représentant
             l'année du calendrier dont les dates sont à stocker
  APPELS   : la procédure stockée SP_PLN_CREATE_ONE_YEAR qui créé une année
             de date
*************************************************************************/

CREATE PROCEDURE SP_PLN_CREATE_YEAR @AN INTEGER
AS

-- alimente les tables :
--    TR_PLN_JOUR_ANNEE_PJA
--    TR_PLN_ANNEE_PAN
--    TR_PLN_JOUR_PJR
--    TR_PLN_SEMAINE_PSM


-- vérifaction des limites d'utilisation
-- limite de calcul : année comprise entre 1601 et 2399

IF @AN < 1901 OR @AN > 2099
 BEGIN
    DECLARE @TXT_ERROR VARCHAR(300)
    SET @TXT_ERROR = 'Il n''est pas possible de calculer le calendrier pour des
années hors de la plage 1900 / 2100.'
                +' En l''occurence vous avez tené de calculer les dates de l''année
%d.'
```

```
    RAISERROR (@TXT_ERROR, 16, 1, @AN)
    RETURN
 END

--DECLARE @JOUR_DE_LAN DATETIME
DECLARE @MIN_AN INTEGER
DECLARE @MAX_AN INTEGER
DECLARE @i integer

--DECLARE @JOUR_FIN_SEMAINE DATETIME

--DECLARE @leapYear bit

SET NOCOUNT ON
SET DATEFORMAT YMD

--SET @JOUR_DE_LAN = CAST(CAST(@AN AS VARCHAR(4)) + '-01-01' AS DATETIME)

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION

-- création des jours de l'année de 1 à 366 si cela n'est pas déjà fait.
IF NOT EXISTS(SELECT 1
              FROM   TR_PLN_JOUR_ANNEE_PJA
              HAVING COUNT(*) = 366)
BEGIN
   SET @i = 1
   WHILE @i <= 366
   BEGIN
      INSERT INTO TR_PLN_JOUR_ANNEE_PJA
      VALUES (@i)
      IF @@ERROR <> 0 BEGIN ROLLBACK RETURN END
      SET @i = @i + 1
   END
END

-- création des numéro de semaine de 1 à 53 si cela n'est pas déjà fait.
IF NOT EXISTS(SELECT 1
              FROM   TR_PLN_SEMAINE_PSM
              HAVING COUNT(*) = 53)
BEGIN
   SET @i = 1
   WHILE @i <= 53
   BEGIN
      INSERT INTO TR_PLN_SEMAINE_PSM
      VALUES (@i)
      IF @@ERROR <> 0 BEGIN ROLLBACK RETURN END
      SET @i = @i + 1
   END
END

COMMIT TRANSACTION


-- l'année à créer a t-elle déjà été créée ?
IF EXISTS (SELECT 1
           FROM   T_PLN_JOUR_PJR
           GROUP  BY PJR_DATE
           HAVING COUNT(*) >= 365)
-- oui, alors retour
RETURN

-- non, alors on regarde si d'autres années ont été créées
SELECT @MIN_AN = MIN(PAN_ID), @MAX_AN = MAX(PAN_ID)
FROM   T_PLN_JOUR_PJR
GROUP  BY PAN_ID
HAVING COUNT(*) >= 365

-- le calendrier est vide : une seule année est à créer
IF @MIN_AN IS NULL AND @MAX_AN IS NULL
BEGIN
   EXEC SP_PLN_CREATE_ONE_YEAR @AN
   RETURN
END

-- le calendrier est déjà rempli et l'année à créer si situe
-- avant les années déjà créées
IF @AN < @MIN_AN
BEGIN
   WHILE @AN < @MIN_AN
   BEGIN
      EXEC SP_PLN_CREATE_ONE_YEAR @AN
      SET @AN = @AN + 1
   END
   RETURN
END

-- le calendrier est déjà rempli et l'année à créer si situe
-- après les années déjà créées
IF @AN > @MAX_AN
BEGIN
   WHILE @AN > @MIN_AN
   BEGIN
      EXEC SP_PLN_CREATE_ONE_YEAR @AN
      SET @AN = @AN - 1
   END
   RETURN
END

GO

Exemple d'appel du script :
SP_PLN_CREATE_YEAR 2000 => création des dates de l'année 2000 du 26/12/1999 au
6/1/2001
SP_PLN_CREATE_YEAR 2010 => création des dates de toutes les années entre 2001 et
2010

REQUÊTE DE MISE A JOUR DES RANGS
********************************

-- maj du rang des années
UPDATE T_PLN_JOUR_PJR
SET PJR_RANG_AN =
       CAST(
       (CAST(PAN_ID AS FLOAT) - 1899.0)
     + (CAST(PJR_RANG_JOUR AS FLOAT)
```

```
     - (SELECT CAST(PJR2.PJR_RANG_JOUR AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR2
        WHERE  PJR2.PJM_ID = 1
          AND  PJR2.PMS_ID = 1
          AND  PJR2.PAN_ID = PJR.PAN_ID +1))
     / (SELECT CAST(COUNT(*) AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR3
        WHERE  PJR3.PAN_ID = PJR.PAN_ID)
        AS DECIMAL(10,4))
FROM T_PLN_JOUR_PJR PJR

-- maj du rang des mois
UPDATE T_PLN_JOUR_PJR
SET PJR_RANG_MOIS =
       CAST(
       (CAST(PAN_ID AS FLOAT) - 1900.0) * 12 + PMS_ID + 1
     + (CAST(PJR_RANG_JOUR AS FLOAT)
     - (SELECT CAST(PJR2.PJR_RANG_JOUR AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR2
        WHERE  PJR2.PJM_ID = 1
          AND  PJR2.PMS_ID = (PJR.PMS_ID  % 12) + 1
          AND  PJR2.PAN_ID =
               CASE
               WHEN PJR.PMS_ID + 1 = 13
                    THEN PJR.PAN_ID +1
                   ELSE PJR.PAN_ID
               END))
     / (SELECT CAST(COUNT(*) AS FLOAT)
        FROM   T_PLN_JOUR_PJR PJR3
        WHERE  PJR3.PMS_ID = PJR.PMS_ID
          AND  PJR3.PAN_ID = PJR.PAN_ID)
       AS DECIMAL(10,4))
FROM T_PLN_JOUR_PJR PJR

-- maj du rang des semaines
UPDATE T_PLN_JOUR_PJR
SET PJR_RANG_SEMAINE =
        CAST(
        CAST(PSM_ID AS FLOAT) + 51
     + (SELECT SUM(MAX_PSM_ID)
        FROM   (SELECT MAX(PSM_ID) AS MAX_PSM_ID
                FROM   T_PLN_JOUR_PJR
                WHERE  PAN_ID < PJR.PAN_ID
                GROUP  BY PAN_ID)
        T )
    + (CAST(PJS_ID AS FLOAT) - 1) / 7 )
       AS DECIMAL(10,4))
FROM T_PLN_JOUR_PJR PJR

-- maj du rang des trimestres
UPDATE T_PLN_JOUR_PJR
SET PJR_RANG_TRIMESTRE =
CAST((CAST(PAN_ID AS FLOAT) - 1900) * 4 + PTR_ID
-- + nombre de jours écoulé
     + (CAST((SELECT COUNT(*)
              FROM   T_PLN_JOUR_PJR PJR2
              WHERE  PJR2.PTR_ID = PJR.PTR_ID
                 AND PJR2.PAN_ID = PJR.PAN_ID
                 AND PJR2.PJR_DATE <= PJR.PJR_DATE) AS FLOAT) - 1)
-- divisé par le total de jours
     / (CAST((SELECT COUNT(*)
              FROM   T_PLN_JOUR_PJR PJR3
              WHERE  PJR3.PTR_ID = PJR.PTR_ID
                 AND PJR3.PAN_ID = PJR.PAN_ID) AS FLOAT))
       AS DECIMAL(10,4))
FROM T_PLN_JOUR_PJR PJR

-- maj du rang des semestres
UPDATE T_PLN_JOUR_PJR
SET PJR_RANG_SEMESTRE =
      CAST((CAST(PAN_ID AS FLOAT) - 1900) * 2 + PST_ID
-- + nombre de jours écoulé
     + (CAST((SELECT COUNT(*)
              FROM   T_PLN_JOUR_PJR PJR2
              WHERE  PJR2.PST_ID = PJR.PST_ID
                 AND PJR2.PAN_ID = PJR.PAN_ID
                 AND PJR2.PJR_DATE <= PJR.PJR_DATE) AS FLOAT) - 1)

-- divisé par le total de jours
     / (CAST((SELECT COUNT(*)
              FROM   T_PLN_JOUR_PJR PJR3
              WHERE  PJR3.PST_ID = PJR.PST_ID
                 AND PJR3.PAN_ID = PJR.PAN_ID) AS FLOAT))
       AS DECIMAL(10,4))
FROM T_PLN_JOUR_PJR PJR

A titre indicatif voici les nombre de semaines dans les années allant de 1900 à 2030
ainsi que le cumul du nombre de semaines depuis le 1er janvier 1900

ANNEE  JOUR_DE_LAN JOUR_SEMAINE NOMBRE_SEMAINE CUMUL_SEMAINE
------ ---------- ------------ -------------- -------------
1900   1900-01-01  lundi        52             52
1901   1901-01-01  mardi        52             104
1902   1902-01-01  mercredi     52             156
1903   1903-01-01  jeudi        53             209
1904   1904-01-01  vendredi     53             262
1905   1905-01-01  dimanche     52             314
1906   1906-01-01  lundi        52             366
1907   1907-01-01  mardi        52             418
1908   1908-01-01  mercredi     53             471
1909   1909-01-01  vendredi     53             524
1910   1910-01-01  samedi       52             576
1911   1911-01-01  dimanche     52             628
1912   1912-01-01  lundi        52             680
1913   1913-01-01  mercredi     52             732
1914   1914-01-01  jeudi        53             785
1915   1915-01-01  vendredi     53             838
1916   1916-01-01  samedi       52             890
1917   1917-01-01  lundi        52             942
1918   1918-01-01  mardi        52             994
1919   1919-01-01  mercredi     52             1046
1920   1920-01-01  jeudi        53             1099
1921   1921-01-01  samedi       53             1152
1922   1922-01-01  dimanche     52             1204
1923   1923-01-01  lundi        52             1256
1924   1924-01-01  mardi        52             1308
```

1925	1925-01-01	jeudi	53	1361
1926	1926-01-01	vendredi	53	1414
1927	1927-01-01	samedi	52	1466
1928	1928-01-01	dimanche	52	1518
1929	1929-01-01	mardi	52	1570
1930	1930-01-01	mercredi	52	1622
1931	1931-01-01	jeudi	53	1675
1932	1932-01-01	vendredi	53	1728
1933	1933-01-01	dimanche	52	1780
1934	1934-01-01	lundi	52	1832
1935	1935-01-01	mardi	52	1884
1936	1936-01-01	mercredi	53	1937
1937	1937-01-01	vendredi	53	1990
1938	1938-01-01	samedi	52	2042
1939	1939-01-01	dimanche	52	2094
1940	1940-01-01	lundi	52	2146
1941	1941-01-01	mercredi	52	2198
1942	1942-01-01	jeudi	53	2251
1943	1943-01-01	vendredi	53	2304
1944	1944-01-01	samedi	52	2356
1945	1945-01-01	lundi	52	2408
1946	1946-01-01	mardi	52	2460
1947	1947-01-01	mercredi	52	2512
1948	1948-01-01	jeudi	53	2565
1949	1949-01-01	samedi	53	2618
1950	1950-01-01	dimanche	52	2670
1951	1951-01-01	lundi	52	2722
1952	1952-01-01	mardi	52	2774
1953	1953-01-01	jeudi	53	2827
1954	1954-01-01	vendredi	53	2880
1955	1955-01-01	samedi	52	2932
1956	1956-01-01	dimanche	52	2984
1957	1957-01-01	mardi	52	3036
1958	1958-01-01	mercredi	52	3088
1959	1959-01-01	jeudi	53	3141
1960	1960-01-01	vendredi	53	3194
1961	1961-01-01	dimanche	52	3246
1962	1962-01-01	lundi	52	3298
1963	1963-01-01	mardi	52	3350
1964	1964-01-01	mercredi	53	3403
1965	1965-01-01	vendredi	53	3456
1966	1966-01-01	samedi	52	3508
1967	1967-01-01	dimanche	52	3560
1968	1968-01-01	lundi	52	3612
1969	1969-01-01	mercredi	52	3664
1970	1970-01-01	jeudi	53	3717
1971	1971-01-01	vendredi	53	3770
1972	1972-01-01	samedi	52	3822
1973	1973-01-01	lundi	52	3874
1974	1974-01-01	mardi	52	3926
1975	1975-01-01	mercredi	52	3978
1976	1976-01-01	jeudi	53	4031
1977	1977-01-01	samedi	53	4084
1978	1978-01-01	dimanche	52	4136
1979	1979-01-01	lundi	52	4188
1980	1980-01-01	mardi	52	4240
1981	1981-01-01	jeudi	53	4293
1982	1982-01-01	vendredi	53	4346
1983	1983-01-01	samedi	52	4398
1984	1984-01-01	dimanche	52	4450
1985	1985-01-01	mardi	52	4502
1986	1986-01-01	mercredi	52	4554
1987	1987-01-01	jeudi	53	4607
1988	1988-01-01	vendredi	53	4660
1989	1989-01-01	dimanche	52	4712
1990	1990-01-01	lundi	52	4764
1991	1991-01-01	mardi	52	4816
1992	1992-01-01	mercredi	53	4869
1993	1993-01-01	vendredi	53	4922
1994	1994-01-01	samedi	52	4974
1995	1995-01-01	dimanche	52	5026
1996	1996-01-01	lundi	52	5078
1997	1997-01-01	mercredi	52	5130
1998	1998-01-01	jeudi	53	5183
1999	1999-01-01	vendredi	53	5236
2000	2000-01-01	samedi	52	5288
2001	2001-01-01	lundi	52	5340
2002	2002-01-01	mardi	52	5392
2003	2003-01-01	mercredi	52	5444
2004	2004-01-01	jeudi	53	5497
2005	2005-01-01	samedi	53	5550
2006	2006-01-01	dimanche	52	5602
2007	2007-01-01	lundi	52	5654
2008	2008-01-01	mardi	52	5706
2009	2009-01-01	jeudi	53	5759
2010	2010-01-01	vendredi	53	5812
2011	2011-01-01	samedi	52	5864
2012	2012-01-01	dimanche	52	5916
2013	2013-01-01	mardi	52	5968
2014	2014-01-01	mercredi	52	6020
2015	2015-01-01	jeudi	53	6073
2016	2016-01-01	vendredi	53	6126
2017	2017-01-01	dimanche	52	6178
2018	2018-01-01	lundi	52	6230
2019	2019-01-01	mardi	52	6282
2020	2020-01-01	mercredi	53	6335
2021	2021-01-01	vendredi	53	6388
2022	2022-01-01	samedi	52	6440
2023	2023-01-01	dimanche	52	6492
2024	2024-01-01	lundi	52	6544
2025	2025-01-01	mercredi	52	6596
2026	2026-01-01	jeudi	53	6649
2027	2027-01-01	vendredi	53	6702
2028	2028-01-01	samedi	52	6754
2029	2029-01-01	lundi	52	6806
2030	2030-01-01	mardi	52	6858

En prime, voici comment calculer les dates fériées mobiles des fêtes chrétiennes, le programme est une procédure écrite pour Transact SQL :

```
*************************************************************************
calcul des dates fériées chrétiennes dans le calendrier grégorien
*************************************************************************
Algorithme conçu par Claus Tøndering .
```

```
Version 2.0 - 11 Nov 1998

Copyright and disclaimer
------------------------
        This document is Copyright (C) 1998 by Claus Tondering.
        E-mail: claus@tondering.dk.
        The document may be freely distributed, provided this
        copyright notice is included and no money is charged for
        the document.
        This document is provided "as is". No warranties are made as
        to its correctness.

This algorithm is based in part on the algorithm of Oudin (1940) as
quoted in "Explanatory Supplement to the Astronomical Almanac",
P. Kenneth Seidelmann, editor.

People who want to dig into the workings of this algorithm, may be
interested to know that
    G is the Golden Number-1
    H is 23-Epact (modulo 30)
    I is the number of days from 21 March to the Paschal full moon
    J is the weekday for the Paschal full moon (0=Sunday, 1=Monday,
      etc.)
    L is the number of days from 21 March to the Sunday on or before
      the Paschal full moon (a number between -6 and 28
*******************************************************************************
Le lundi de Pâques est déterminé par :
   Dimanche de Pâques + 1 jour
Le Jeudi de l'ascension est déterminé par :
   Dimanche de Pâques + 39 jours
Le Lundi de pentecôte est déterminé par :
   Dimanche de Pâques + 50 jours


CREATE PROCEDURE SP_PLN_ADD_JOUR_FERIE_MOBILE_CHRETIEN @AN INT
AS

SET DATEFORMAT YMD

DECLARE @G INT
DECLARE @I INT
DECLARE @J INT
DECLARE @C INT
DECLARE @H INT
DECLARE @L INT
DECLARE @JourPaque INT
DECLARE @MoisPaque INT
DECLARE @DimPaque DATETIME
DECLARE @LunPaque DATETIME
DECLARE @JeuAscension DATETIME
DECLARE @LunPentecote DATETIME

SET @G = @AN % 19
SET @C = @AN / 100
SET @H = (@C - @C / 4 - (8 * @C + 13) / 25 + 19 * @G + 15) % 30
SET @I = @H - (@H / 28) * (1 - (@H / 28) * (29 / (@H + 1)) * ((21 - @G) / 11))
SET @J = (@AN + @AN / 4 + @I + 2 - @C + @C / 4) % 7

SET @L = @I - @J
SET @MoisPaque = 3 + (@L + 40) / 44
SET @JourPaque = @L + 28 - 31 * (@MoisPaque / 4)

SET DATEFORMAT YMD

SET @DimPaque = CAST(CAST(@AN AS VARCHAR(4)) + '-'
                + CAST(@MoisPaque AS VARCHAR(2)) + '-'
                + CAST(@JourPaque AS VARCHAR(2)) AS DATETIME)
SET @LunPaque = DATEADD(DAY, 1, @DimPaque)
SET @JeuAscension = DATEADD(DAY, 39, @DimPaque)
SET @LunPentecote = DATEADD(DAY, 50, @DimPaque)

SELECT 'Dimanche' AS JOUR, 'Pâques' AS FETE, @DimPaque AS DATE_FETE
UNION
SELECT 'Lundi' AS JOUR, 'Pâques' AS FETE, @LunPaque AS DATE_FETE
UNION
SELECT 'Jeudi' AS JOUR, 'Ascension' AS FETE, @JeuAscension AS DATE_FETE
UNION
SELECT 'Lundi' AS JOUR, 'Pentecôte AS FETE, @LunPentecote AS DATE_FETE

GO
```

7. Pour en savoir plus sur le sujet

LIVRES :

Developping Time-Oriented Database Applications in SQL - Ricahrds T. Snodgrass - Morgan Kauffmann - 2000
Temporal Data and the Relational Model - C. J. Date, Hugh Darwen, Nikos A. Lorentzos - Morgan Kauffmann - 2000

WEB :

http://membres.lycos.fr/urbainmartin/temps.et.calendriers/
http://pchapelin.free.fr/calrep/index.htm
http://perso.easynet.fr/~cerf/calendar/calendar.htm http://www.bdl.fr/minitel/
http://www.geocities.com/Paris/Louvre/9647/histoire.htm
http://lwh.free.fr/pages/algo/calendriers/calendriers.htm
http://www.altcal.com/propcal.html http://www.cite-sciences.fr/francais/ala_cite/act_educ/education/createurs/temps/tintrod.htm
http://perso.wanadoo.fr/b.villemin/bissexti.html

Livres
SQL - développement
SQL - le cours de référence sur le langage SQL
Avant d'aborder le SQL

Définitions
SGBDR fichier ou client/serveur ?
La base de données exemple (gestion d'un hôtel)
Modélisation MERISE
Mots réservés du SQL
Le SQL de A à Z
Les fondements
Le simple (?) SELECT
Les jointures, ou comment interroger plusieurs tables
Groupages, ensembles et sous-ensembles
Les sous-requêtes
Insérer, modifier, supprimer
Création des bases
Gérer les privilèges ("droits")
Toutes les fonctions de SQL
Les techniques des SGBDR
Les erreur les plus fréquentes en SQL
Les petits papiers de SQLPro
Conférence Borland 2003
L'héritage des données
Données et normes
Modélisation par méta données
Optimisez votre SGBDR et vos requêtes SQL
Le temps, sa mesure, ses calculs
QBE, le langage de ZLOOF
Des images dans ma base
La jointure manquante
Clefs auto incrémentées
L'indexation textuelle
L'art des "Soundex"
Une seule colonne, plusieurs données
La division relationnelle, mythe ou réalité ?
Gestion d'arborescence en SQL
L'avenir de SQL
Méthodes et standards
Les doublons
SQL Server
Eviter les curseurs
Un aperçu de TRANSACT SQL V 2000
SQL Server 2000 et les collations
Sécurisation des accès aux bases de données SQL Server
Des UDF pour SQL Server
SQL Server et le fichier de log...
Paradox
De vieux articles publiés entre 1995 et 1999 dans la défunte revue Point DBF

(1)	A cette date arbitraire, on trouve : - discours de Pompidou à Aurillac - naissance de Gérald Gardrinier dit De Palmas, chanteur - naissance d'Alain Roche footballeur - publication au JO du décret n° 67-896 du 6 octobre 1967 - fondation du judo club de Nangis - décès de Marcel Aymé - création de la Ligue du Limousin de Voile - grand prix moto du Japon vainqueur Mike HAILWOOD sur 350 cc (HONDA)