## 5.7 – Jointure partitionnée

La jointure partitionnée permet de partitionner la table à joindre avant d'effectuer la jointure de manière à faire apparaître plus clairement certaines lignes manquantes. Elle n'a d'intérêt que combinée avec une jointure externe, et remplace bien souvent des valeurs qui disparaitraient d'une jointure classique.

La syntaxe est la suivante :

```
FROM <table1> [ [ AS ] alias1 ] PARTITION BY ( <liste_expression> )
     { LEFT | RIGHT | FULL [ OUTER]
        JOIN <table2> [ [ AS ] alias2 ]
           ON <predicat_de_jointure>
```

Ou encore :

```
FROM   <table1> [ [ AS ] alias1 ]
       { LEFT | RIGHT | FULL [ OUTER]
          JOIN <table2> [ [ AS ] alias2 ] PARTITION BY ( <liste_expression> )
             ON <predicat_de_jointure>
```

Avec :

```
<liste_expression> ::= <expression1>
                          [, <expression2>
                             [, ...
                                [, <liste_expressionN> ] ] ]
```

De manière logique, ce type de jointure découpe l'exécution des correspondances entre les tables en autant de partitions qu'exprimé dans la liste des expressions de partitionnement.
Voici une syntaxe équivalente pour une partition de la table 1 ne comprenant qu'une colonne *col* partitionnée en *valeur1, valeur2, ... valeur N* :

La requête

```
SELECT ...
FROM   <table1> [ [ AS ] alias1 ] PARTITION BY ( col )
       {RIGHT OUTER JOIN <table2> [ [ AS ] alias2 ]
            ON <predicat_de_jointure>
```

Peut être récrite :

```
SELECT ...
FROM   <table1> [ [ AS ] alias1 ]
       RIGHT OUTER] JOIN <table2> [ [ AS ] alias2 ]
           ON <predicat_de_jointure> AND col = valeur1
UNION ALL
SELECT ...
FROM   <table1> [ [ AS ] alias1 ]
       RIGHT OUTER] JOIN <table2> [ [ AS ] alias2 ]
           ON <predicat_de_jointure> AND col = valeur2
UNION ALL

...

UNION ALL
SELECT ...
FROM   <table1> [ [ AS ] alias1 ]
              RIGHT OUTER] JOIN <table2> [ [ AS ] alias2 ]
           ON <predicat_de_jointure> AND col = valeurN
```

L'inconvénient de ce découpage est qu'il faut connaître à l'avance toutes les valeurs du domaine de la liste d'expression du partitionnement et d'établir les requêtes une par une pour chaque valeur ! Bien entendu il existe d'autres équivalences plus simples notamment avec deux jointures : l'une en produit cartésien et l'autre en jointure externe.

Pour bien comprendre ce type de jointure voici un exemple complet. Deux tables comportent, l'une, la liste des entrepôts (T_ENTREPOT_ETP) et l'autre une liste des chargement des camions par date à ces différents entrepôts (T_CHARGEMENT_CHG). La structure des tables et les données sont les suivantes :

***Exemple 5.22*** – *Illustration de la jointure partitionnée*

***Exemple 5.22.1*** – *Création des tables*

```
CREATE TABLE T_CHARGEMENT_CHG
(CHG_ID             INT IDENTITY PRIMARY KEY,
 CHG_DATE           DATE NOT NULL,
 CHG_ENTREPOT       VARCHAR(32) NOT NULL,
 CHG_NOMBRE         SMALLINT NOT NULL);

CREATE TABLE T_ENTREPOT_ETP
(ETP_ENTREPOT varchar(32));
```

***Exemple 5.22.2*** – *Insertion des données*

```
INSERT INTO T_ENTREPOT_ETP ( ETP_ENTREPOT ) VALUES ( 'Marseille' );
INSERT INTO T_ENTREPOT_ETP ( ETP_ENTREPOT ) VALUES ( 'Paris' );
INSERT INTO T_ENTREPOT_ETP ( ETP_ENTREPOT ) VALUES ( 'Lyon' );

INSERT INTO T_CHARGEMENT_CHG  VALUES (  '2023-09-17', 'Marseille', 11 ) ;
INSERT INTO T_CHARGEMENT_CHG  VALUES (  '2023-09-15', 'Marseille', 10 ) ;
INSERT INTO T_CHARGEMENT_CHG  VALUES (  '2023-09-16', 'Paris', 20 ) ;
INSERT INTO T_CHARGEMENT_CHG  VALUES (  '2023-09-17', 'Lyon', 31 ) ;
INSERT INTO T_CHARGEMENT_CHG  VALUES (  '2023-09-16', 'Lyon', 30 ) ;
```

***Exemple 5.22.3*** – *Requête donnant des résultats incomplets*

```
SELECT S.CHG_DATE, L.ETP_ENTREPOT, S.CHG_NOMBRE
FROM   T_ENTREPOT_ETP AS L
       RIGHT OUTER JOIN T_CHARGEMENT_CHG S
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT;

SELECT S.CHG_DATE, L.ETP_ENTREPOT, S.CHG_NOMBRE
FROM   T_ENTREPOT_ETP AS L
       LEFT OUTER JOIN T_CHARGEMENT_CHG S
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT;

SELECT S.CHG_DATE, L.ETP_ENTREPOT, S.CHG_NOMBRE
FROM   T_ENTREPOT_ETP AS L
       INNER JOIN T_CHARGEMENT_CHG S
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT;
```

Ces requêtes donnent toutes le même résultat :

```
CHG_DATE   ETP_ENTREPOT    CHG_NOMBRE
---------- --------------- ----------
2023-09-17 Marseille       11
2023-09-15 Marseille       10
2023-09-16 Paris           20
2023-09-17 Lyon            31
2023-09-16 Lyon            30
```

Aucune de ces requêtes ne fait apparaître un résultat complet. Il manque les lignes suivantes :

```
CHG_DATE   ETP_ENTREPOT    CHG_NOMBRE
---------- --------------- ----------
2023-09-16 Marseille       NULL
2023-09-15 Paris           NULL
2023-09-17 Paris           NULL
2023-09-15 Lyon            NULL
```

***Exemple 5.22.4*** – *Jointure partitionnée présentant les lignes manquantes*

```
SELECT S.CHG_DATE, S.CHG_NOMBRE, L.ETP_ENTREPOT
FROM   T_CHARGEMENT_CHG AS S PARTITION BY (CHG_DATE)
       RIGHT OUTER JOIN T_ENTREPOT_ETP L
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT
ORDER BY 1, 2;
```

La jointure partitionnée résous élégamment ce problème en faisant figurer les manques :

```
CHG_DATE   ETP_ENTREPOT     CHG_NOMBRE
---------- ---------------- -----------
2019-01-15 Boston           NULL
2019-01-15 London           10
2019-01-15 Paris            NULL
2019-01-16 Boston           30
2019-01-16 London           NULL
2019-01-16 Paris            20
2019-01-17 Boston           31
2019-01-17 London           11
2019-01-17 Paris            NULL
```

Les équivalences ne manquent pas... Voici tout d'abord la pire qui concatène différentes requêtes via l'opérateur d'ensembles UNION ALL[9] :

***Exemple 5.22.5*** – *Équivalence de la jointure partitionnée avec une succession d'UNION ALL et des requêtes filtrées pour chacune des dates*

```
SELECT CAST('2023-09-15' AS DATE) AS CHG_DATE, L.ETP_ENTREPOT, S.CHG_NOMBRE
FROM   T_ENTREPOT_ETP AS L
       RIGHT OUTER JOIN T_CHARGEMENT_CHG AS S
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT AND S.CHG_DATE = '2023-09-15'
UNION ALL
SELECT CAST('2023-09-16' AS DATE)  AS CHG_DATE, L.ETP_ENTREPOT, S.CHG_NOMBRE
FROM   T_CHARGEMENT_CHG AS S
       RIGHT OUTER JOIN T_ENTREPOT_ETP AS L
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT AND S.CHG_DATE = '2023-09-16'
UNION ALL
SELECT CAST('2023-09-17' AS DATE)  AS CHG_DATE, L.ETP_ENTREPOT, S.CHG_NOMBRE
FROM   T_CHARGEMENT_CHG AS S
       RIGHT OUTER JOIN T_ENTREPOT_ETP AS L
          ON S.CHG_ENTREPOT = L.ETP_ENTREPOT  AND S.CHG_DATE = '2023-09-17'
ORDER BY 1, 2;
```

Comme nous l'avons mentionnée, l'écriture d'une, telle requête nécessiterait la mise au point de nombreuses requêtes chacune reprenant l'un des valeurs du domaine

[9] Bien que nous n'ayons pas encore vu cet opérateur qui sera décrit au chapitre 6, il semble assez évident que les lignes résultantes de l'opération UNION ALL est constitué de l'assemblage de toutes les lignes des différentes requêtes au sein d'un même jeu de résultat.

partitionné. Un tel code serait fastidieux à programmé et la requête pourrait être d'une longueur que certains SGBD Relationnels n'accepterait pas...
D'autres formes, d'écriture par équivalence sont bien plus concises. Elles procèdent avec plusieurs jointures dont un produit cartésien et des sous-requêtes que nous verrons au chapitre 7. En voici quatre différentes.

***Exemple 5.22.6*** – *Différentes équivalence de la jointure partitionnée*

```
-- forme n°1
SELECT CE.*, C.CHG_NOMBRE
FROM   (SELECT *
        FROM   (SELECT DISTINCT CHG_DATE
                FROM   T_CHARGEMENT_CHG) AS T1
               CROSS JOIN (SELECT DISTINCT ETP_ENTREPOT
                           FROM   T_ENTREPOT_ETP) AS T2
        ) AS CE
        LEFT OUTER JOIN (SELECT CHG_DATE, CHG_ENTREPOT, CHG_NOMBRE
                         FROM   T_CHARGEMENT_CHG) AS C
           ON CE.ETP_ENTREPOT = C.CHG_ENTREPOT
              AND CE.CHG_DATE = C.CHG_DATE
ORDER BY 1, 2;

-- forme n°2
WITH T AS
(SELECT C.CHG_DATE, E.ETP_ENTREPOT, C.CHG_NOMBRE
 FROM   T_CHARGEMENT_CHG AS C
        INNER JOIN T_ENTREPOT_ETP AS E
           ON C.CHG_ENTREPOT = E.ETP_ENTREPOT)
SELECT *
FROM   T
UNION  ALL
SELECT DISTINCT C.CHG_DATE, E.ETP_ENTREPOT, NULL
FROM   T_CHARGEMENT_CHG AS C
       CROSS JOIN T_ENTREPOT_ETP AS E
WHERE  NOT EXISTS(SELECT *
                  FROM   T
                  WHERE  T.CHG_DATE = C.CHG_DATE
                         AND E.ETP_ENTREPOT = T.ETP_ENTREPOT)
ORDER  BY 1, 2;

-- forme n°3
WITH
T AS (SELECT DISTINCT CHG_DATE, X.CHG_ENTREPOT
      FROM   T_CHARGEMENT_CHG
             CROSS JOIN (SELECT CHG_ENTREPOT
                         FROM   T_CHARGEMENT_CHG) AS X)
SELECT T.CHG_DATE, T.CHG_ENTREPOT, S.CHG_NOMBRE
FROM   T
       LEFT OUTER JOIN T_CHARGEMENT_CHG AS S
          ON T.CHG_DATE = S.CHG_DATE
             AND T.CHG_ENTREPOT = S.CHG_ENTREPOT
          LEFT OUTER JOIN T_ENTREPOT_ETP AS L
             ON S.CHG_ENTREPOT = L.ETP_ENTREPOT
ORDER BY 1, 2;

-- forme n°4
WITH
T AS
(
SELECT DISTINCT C.CHG_DATE, E.ETP_ENTREPOT
FROM   T_CHARGEMENT_CHG AS C
       CROSS JOIN T_ENTREPOT_ETP AS E
)
```

```
SELECT T.CHG_DATE, T.ETP_ENTREPOT, C.CHG_NOMBRE
FROM   T
       LEFT OUTER JOIN T_CHARGEMENT_CHG AS C
          ON T.CHG_DATE = C.CHG_DATE
             AND T.ETP_ENTREPOT = C.CHG_ENTREPOT
ORDER BY 1, 2;
```

Nous avons mis en évidence le produit cartésien en gras et surligné les sous requêtes. Notez que la forme 3 utilise deux sous-requêtes imbriquées, l'une en jaune et une autre à l'intérieur de la première en bleu.

Je laisse au lecteur la sagacité de comprendre comment l'articulation des différentes jointures entre ces requêtes permet de résoudre cette jointure partitionnée...