Module 06 - Transactions

Objectifs

- Principe de transaction
- Common Table Expression
- Sous-requête

Transaction

- Une transaction regroupe une ou plusieurs opérations de modifications de données
- Une transaction permet d'éviter qu'un traitement s'exécute partiellement
- Une transaction doit être une unité de travail ('unit of work')
- Une unité de travail doit être exécuté à partir d'une base de données qui se trouve dans un état cohérent. Après son exécution, la base de données doit rester dans un état cohérent
- Exemple : transfert d'argent d'un compte à l'autre
  - On doit ajouter une transaction de débit dans un compte et ajouter une transaction de crédit dans un autre compte. Les deux opérations sont liées pour ne pas perdre d'argent

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql

Transaction

- Si la transaction est validée, les données sont modifiées de façon permanente dans la base de données
- S'il y a des erreurs, les modifications peuvent être annulées et les modifications sont ignorées
- Par défaut, SQL Server crée une transaction par commande SQL qui modifie les données. Si la commande s'exécute correctement, elle fait une auto validation (auto-commit)
  - Ce comportement peut être modifié pour rendre la transaction implicite (le développeur doit alors faire un COMMIT) ou explicite (Voir ce qui suit)

Transaction – SQL

```
BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable }
      [ WITH MARK [ 'description' ] ]
    ]
[ ; ]
```

Début d'une transaction

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }
[ ; ]
```

Création d'un marqueur dans la transaction

```
ROLLBACK { TRAN | TRANSACTION }
    [ transaction_name | @tran_name_variable
    | savepoint_name | @savepoint_variable ]
[ ; ]
```

Annulation des modifications de données jusqu'au marqueur ou du début de la transaction

```
COMMIT [ { TRAN | TRANSACTION } [ transaction_name | @tran_name_variable ] ]
[ ; ]
```

Validation des modifications

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/begin-transaction-transact-sql
https://learn.microsoft.com/en-us/sql/t-sql/language-elements/save-transaction-transact-sql
https://learn.microsoft.com/en-us/sql/t-sql/language-elements/rollback-transaction-transact-sql
https://learn.microsoft.com/en-us/sql/t-sql/language-elements/commit-transaction-transact-sql

Common table expression

- Les CTE permettent de garder des résultats temporaires et de les utiliser dans un requête spécifique
- Peut être suivi d'une instruction SELECT / INSERT / UPDATE / DELETE

```
[ WITH <common_table_expression> [ ,...n ] ]

<common_table_expression>::=
    expression_name [ ( column_name [ ,...n ] ) ]
    AS
    ( CTE_query_definition )
```

https://learn.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql

Common table expression – Exemple

```
SELECT
   ROW_NUMBER() OVER (ORDER BY DATEPART(YEAR, i.InvoiceDate), DATEPART(MONTH, i.InvoiceDate), SUM(il.Quantity * il.UnitPrice) DESC) AS NumeroDeLigne
  DATEPART(YEAR, i.InvoiceDate) AS AnneeVentes, DATEPART(MONTH, i.InvoiceDate) AS MoisVentes
  , DENSE_RANK() OVER (PARTITION BY DATEPART(YEAR, i.InvoiceDate), DATEPART(MONTH, i.InvoiceDate) ORDER BY SUM(il.Quantity * il.UnitPrice) DESC) AS Classement
  , NTILE(3) OVER (PARTITION BY DATEPART(YEAR, i.InvoiceDate), DATEPART(MONTH, i.InvoiceDate) ORDER BY SUM(il.Quantity * il.UnitPrice) DESC) AS Tiers
  ,p.FullName
  , FORMAT(SUM(il.Quantity * il.UnitPrice), 'C') AS MontantVentes
FROM Sales.Invoices i
INNER JOIN Application.People p ON i.SalespersonPersonID = p.PersonID
INNER JOIN Sales.InvoiceLines il ON il.InvoiceID = i.InvoiceID
GROUP BY p.PersonID, p.FullName, DATEPART(YEAR, i.InvoiceDate), DATEPART(MONTH, i.InvoiceDate)
ORDER BY AnneeVentes, MoisVentes, MontantVentes DESC, FullName
```

```
WITH VolumeVentesParVendeurAnneeMois
AS (
        SELECT
           DATEPART(YEAR, i.InvoiceDate) AS AnneeVentes, DATEPART(MONTH, i.InvoiceDate) AS MoisVentes, p.FullName
          , SUM(il.Quantity * il.UnitPrice) AS MontantVentes
        FROM Sales.Invoices i
        INNER JOIN Application.People p ON i.SalespersonPersonID = p.PersonID INNER JOIN Sales.InvoiceLines il ON il.InvoiceID = i.InvoiceID
        GROUP BY p.PersonID, p.FullName, DATEPART(YEAR, i.InvoiceDate), DATEPART(MONTH, i.InvoiceDate)
)
SELECT
   ROW_NUMBER() OVER (ORDER BY vvpam.AnneeVentes, vvpam.MoisVentes, vvpam.MontantVentes DESC) AS NumeroDeLigne
  , vvpam.AnneeVentes
  , vvpam.MoisVentes
  , DENSE_RANK() OVER (PARTITION BY vvpam.AnneeVentes, vvpam.MoisVentes ORDER BY vvpam.MontantVentes DESC) AS Classement
  , NTILE(3) OVER (PARTITION BY vvpam.AnneeVentes, vvpam.MoisVentes ORDER BY vvpam.MontantVentes DESC) AS Tiers
  , vvpam.FullName
  , FORMAT(vvpam.MontantVentes, 'C') AS MontantVentes
FROM VolumeVentesParVendeurAnneeMois vvpam
ORDER BY AnneeVentes, MoisVentes, MontantVentes DESC, FullName
```

Sous-requêtes

- Un sous-requête ou requête imbriquée est une requête de sélection de données dans une requête de type SELECT (projection / filtrage / source) / UPDATE / DELETE
- **La plupart des sous-requêtes peuvent être exprimées/remplacées avec/par des jointures**

```
/* SELECT statement built using a subquery. */
SELECT [Name]
FROM Production.Product
WHERE ListPrice =
    (SELECT ListPrice
     FROM Production.Product
     WHERE [Name] = 'Chainring Bolts' );
GO

/* SELECT statement built using a join that returns
   the same result set. */
SELECT Prd1.[Name]
FROM Production.Product AS Prd1
     JOIN Production.Product AS Prd2
       ON (Prd1.ListPrice = Prd2.ListPrice)
WHERE Prd2.[Name] = 'Chainring Bolts';
GO
```

https://learn.microsoft.com/en-us/sql/relational-databases/performance/subqueries

Sous-requêtes

- Au niveau des performances, il y a peu d'impact entre les deux approches. sauf dans le cas de validations d'existences d'enregistrements ou de sous-requêtes à traiter par enregistrement (ie sous requête interne dépendante d'une colonne de la requête externe)

```
SELECT DISTINCT c.LastName, c.FirstName, e.BusinessEntityID
FROM Person.Person AS c JOIN HumanResources.Employee AS e
ON e.BusinessEntityID = c.BusinessEntityID
WHERE 5000.00 IN
    (SELECT Bonus
    FROM Sales.SalesPerson sp
    WHERE e.BusinessEntityID = sp.BusinessEntityID) ;
GO
```

Sous-requête synchronisée
(Correlated subquery)

https://learn.microsoft.com/en-us/sql/relational-databases/performance/subqueries

Sous-requêtes – Opérateurs booléens

Une sous-requête peut renvoyer de 0 à n enregistrements qui peuvent être utilisés aussi avec des opérateurs logiques

Opérateur	Utilisation
IN (v1, v2, …, vn) IN (sous requête)	Vrai si la valeur testée est dans l'ensemble. IN est équivalent à « = ANY ».
<valeur> <opérateurComparaison> ANY (…)	Vrai si la comparaison est vraie pour au moins une valeur de l'ensemble (donc faux si l'ensemble est vide).
<valeur> <opérateurComparaison> ALL (…)	Vrai si la comparaison est vraie pour toutes les valeurs de l'ensemble (donc vrai si l'ensemble est vide).
EXISTS (…)	Si la sous-interrogation renvoie un résultat (au moins une ligne), la valeur retournée est Vrai sinon la valeur Faux est retournée. Équivalent à « <colonne> = ANY (…) » ou « <colonne> IN (…)
NOT …	Inverse le résultat d'un autre prédicat. NOT IN est équivalent à « != ALL (…) »

```
3 < ALL (SELECT ID FROM T1)
```

```
SELECT a.FirstName, a.LastName
FROM Person.Person AS a
WHERE EXISTS
(SELECT *
    FROM HumanResources.Employee AS b
    WHERE a.BusinessEntityID = b.BusinessEntityID
    AND a.LastName = 'Johnson') ;
GO
```

```
SELECT DISTINCT s.Name
FROM Sales.Store AS s
WHERE EXISTS
(SELECT *
    FROM Purchasing.Vendor AS v
    WHERE s.Name = v.Name) ;
GO
```

```
SELECT DISTINCT s.Name
FROM Sales.Store AS s
WHERE s.Name = ANY
(SELECT v.Name
    FROM Purchasing.Vendor AS v ) ;
GO
```

Aller plus loin

- Problèmes de concurrence de transactions :
  - https://learn.microsoft.com/en-us/sql/t-sql/language-elements/transaction-isolation-levels
  - https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide
- Approches optimiste / pessimiste : https://learn.microsoft.com/en-us/sql/connect/ado-net/optimistic-concurrency