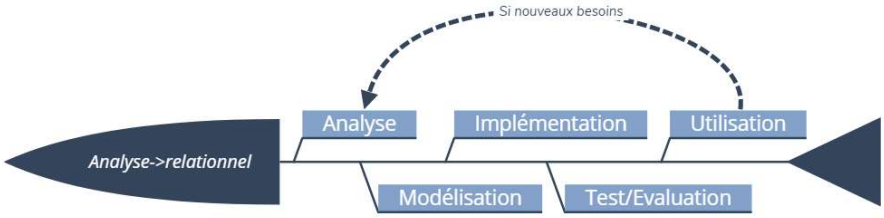
SQL déclaratif

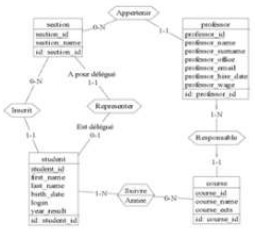
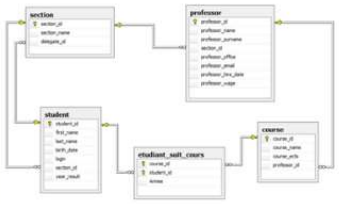
Table des matières

Partie 1 : Introduction aux concepts

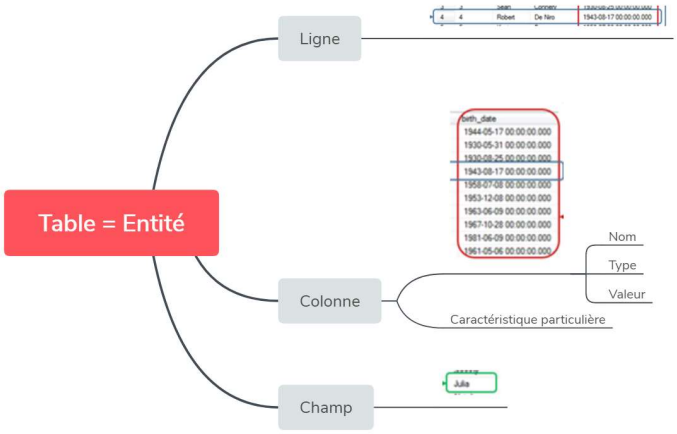
Base de données et SGBD

De l'analyse au relationnel

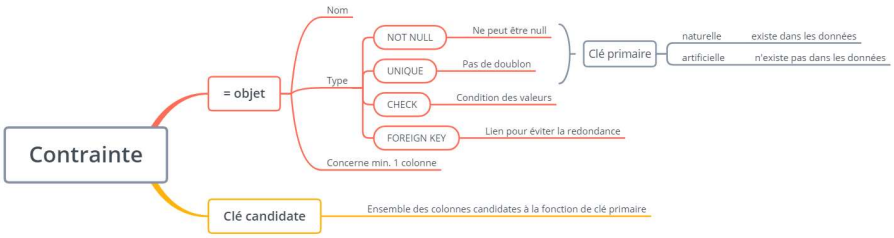


De l'analyse au relationnel		
	Schéma EA	Schéma Relationnel
BUT	Modélisation	Plan de la BD
Acteur	Entité	Table
Attribut	Attribut	Colonne
Nom Interaction	Association	Contraintes d'intégrité référentielles
Type Interaction	Cardinalité	Clé Etrangère : O-M / O-O Table : M-M
Exemple		

Notions de tables



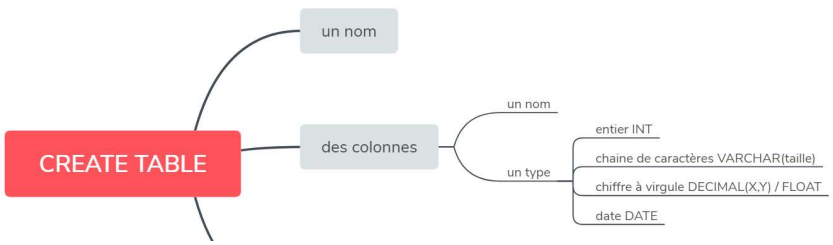
Contraintes



Partie 2 : DDL : Data Definition Language (Créer la base de données)

CREATE TABLE : créer une table

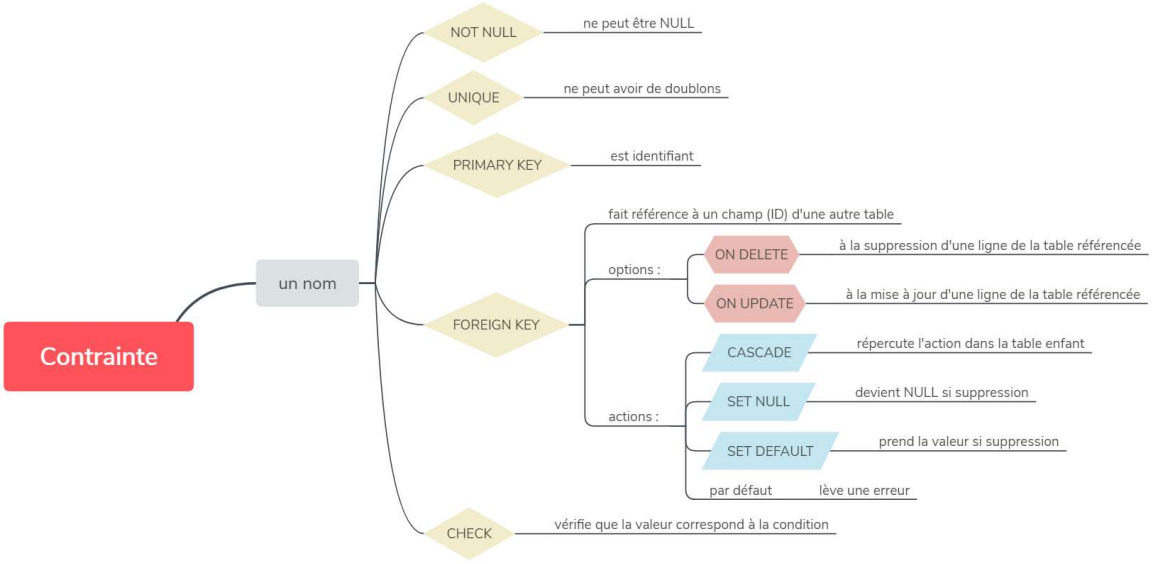
```
CREATE TABLE nom_table (
nom_colonne1 TYPE,
nom_colonne2 TYPE,
nom_colonne3 TYPE
)
```



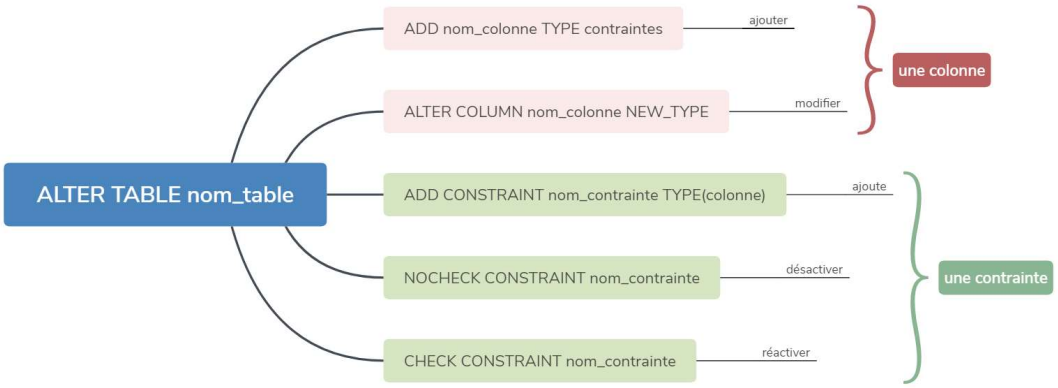
IDENTITY et DEFAULT

Contraintes

CREATE TABLE *nom_table* **(** *nom_colonne1* **TYPE,** *nom_colonne2* **TYPE,** **CONSTRAINT** *nom_contrainte* **TYPE** **CONTRAINTE** *(colonne_concernée)* **)**	**CREATE TABLE** *nom_table* **(** *nom_colonne1* **TYPE,** *nom_colonne2* **TYPE** **CONSTRAINT** *nom_contrainte* **TYPE** **CONTRAINTE,** *nom_colonne3* **TYPE** **)**	**CREATE TABLE** *nom_table* **(** *nom_colonne1* **TYPE,** *nom_colonne2* **TYPE TYPE** **CONTRAINTE,** *nom_colonne3* **TYPE** **)**



ALTER TABLE : modifier la structure d'une table



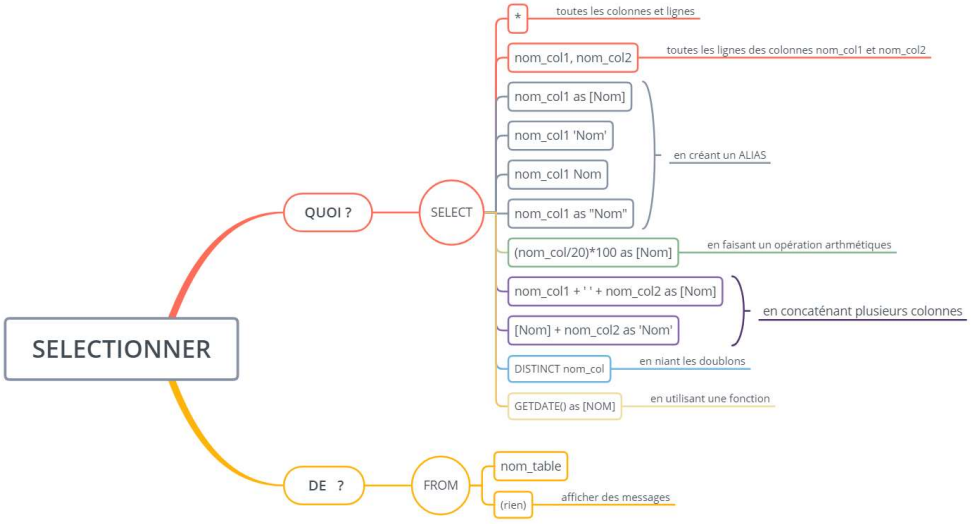
TRUNCATE TABLE nom_table : vider la table

DROP TABLE nom_table : supprimer la table

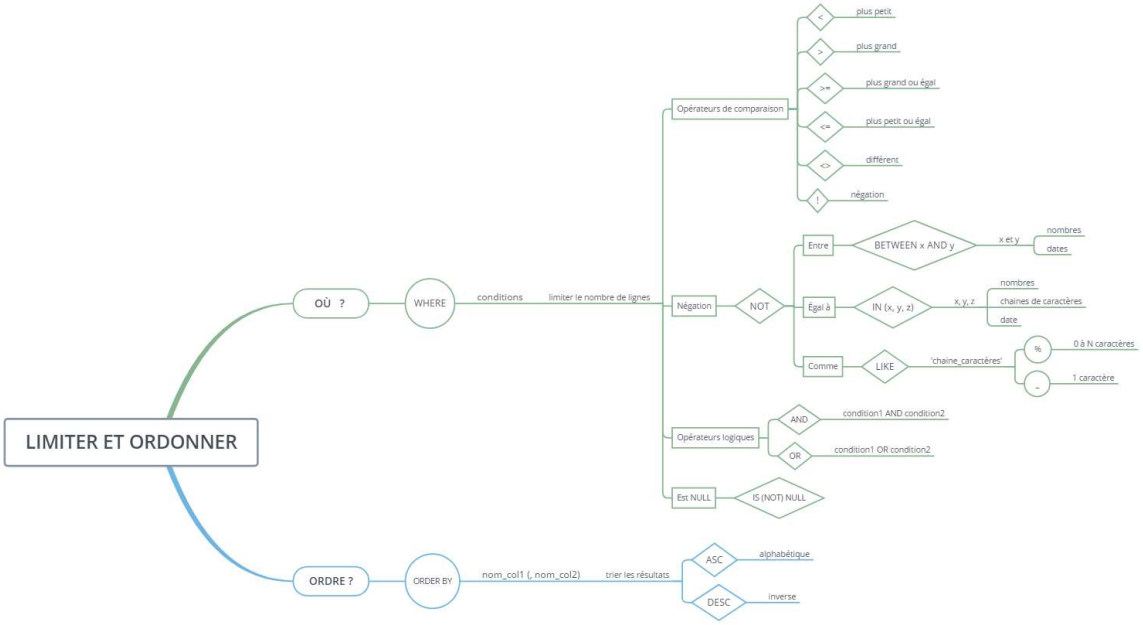
Partie 3 : DRL : Data Retrieval Language (Sélectionner des données)

SELECT

```
SELECT colonne1, colonne2, colonne3, …
FROM nom_table
```



Limiter et ordonner



Les fonctions

Concaténation

+ ou CONCAT	concaténer	**CONCAT(**nom_col1, ' ' , nom_col2**)**

Conversion

CONVERT	convertir	**CONVERT (NOUVEAU_TYPE,** *valeur_à_convertir***)** **CONVERT (TYPE_CHAINE_DE_CARACTÈRES,** *date_à_convertir*, **format_date)**
CAST	convertir	**CAST(**valeur_à_convertir AS **NOUVEAU TYPE)**

Date

GETDATE	récupérer la date et l'heure	**GETDATE()**
DATEPART	extraire une partie d'une date	**DATEPART (***partie_de_date_à_extraire***,** *date_traitée***)**
DATEADD	additionner des dates	`DATEADD (datepart , number , date )`
DATEDIFF	soustraire des dates	`DATEDIFF ( datepart , startdate , enddate )`
DAY	retourne le jour	`DAY ( date )`
MONTH	retrouve le mois	`MONTH (date)`
YEAR	retourne l'année	`YEAR(date)`

Chaines de caractères

CHARINDEX	récupérer la position d'un car.	**CHARINDEX (***chaine_de_caractères_recherchée***,** *valeur_à_évaluer***)**
LEN	récupérer le nombre de car.	**LEN (***chaine_de_caractères_à_mesurer***)**
SUBSTRING	couper une chaine	***SUBSTRING (chaine_de_caractères, position_départ, nombre_caractères)***
UPPER	mettre en MAJUSCULE	***UPPER (chaine_de_caractères)***
LOWER	mettre en minuscule	***LOWER (chaine_de_caractères)***
REPLACE	remplacer des car.	***REPLACE (chaine_de_caractères_traitée, caract_à_remplacer, nouveau_caract)***
LTRIM	retirer les espaces blancs avant	***LTRIM (chaine_de_caractères)***
RTRIM	retirer les espaces blancs après	***RTRIM (chaine_de_caractères)***
LEFT	récupérer le X car. à gauche	***LEFT(****nom_col, X)*
RIGHT	récupérer le X car. à droite	***RIGHT(****nom_col, X)*

Mathématiques

ABS	récupérer la valeur absolue	**ABS (***nombre***)**
%	récupérer le modulo	***dividende % diviseur***
COUNT	renvoyer le nombre de champs	***COUNT (*)***
MAX	renvoyer le nombre max	***MAX (colonne)***
MIN	renvoyer le nombre min	***MIN (colonne)***
SUM	renvoyer la somme	***SUM (colonne)***
AVG	renvoyer la moyenne	***AVG (colonne)***

Structures conditionnelles

CASE	modifier l'affichage	***CASE*** ***WHEN colonne_à_évaluer + condition THEN valeur1*** ***…*** ***ELSE valeur_par_défaut*** ***END***
		CASE colonne_à_évaluer ***WHEN valeur_de_comparaison1 THEN valeur1*** ***…*** ***ELSE valeur_par_défaut*** ***END***
NULLIF	mettre null si	***NULLIF (colonne_considérée, valeur_à_mettre_à_NULL)***
COALESCE	renvoyer 1er valeur non NULL	***COALESCE (colonne1, colonne2, …, colonneN)***

GROUP BY : grouper par nom_col

La colonne doit être sélectionnée dans le select.

HAVING : respectant telle condition

```
SELECT section_id, course_id, AVG(year_result)
  FROM student
 WHERE section_id IN (1010, 1020)
GROUP BY section_id, course_id
  HAVING SUM(year_result) >= 2
ORDER BY section_id
```

La colonne du HAVING ne doit pas forcément être sélectionnée.

ROLL UP : crée une ligne de sous-total en fonction des colonnes sous-groupes sélectionnées

```
SELECT colonnes, fonction_agrégation(colonne)
FROM table
GROUP BY ROLLUP (sous_groupes_agrégation)
```

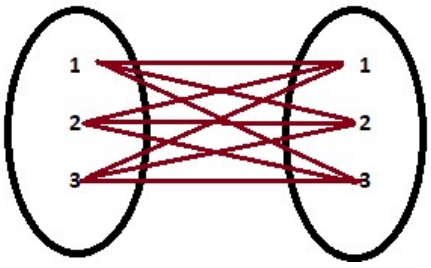
CUBE : crée une ligne de sous-total en fonction de toutes les colonnes sous-groupes sélectionnées = rollup + col2

```
SELECT colonnes, fonction_agrégation(colonne)
FROM table
GROUP BY CUBE (sous_groupes_agrégation)
```

Jointures

CROSS JOIN : produit cartésien des lignes de chaque table.

```
SELECT A.c1, B.c1
FROM A
CROSS JOIN B
```

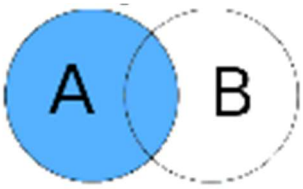


A B

INNER JOIN : là où il y a une correspondance entre les champs d'une colonne dans 2 tables

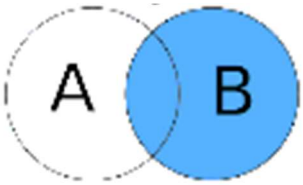

```
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
```

LEFT OUTER JOIN : affiche toutes les lignes de A + les correspondances avec B



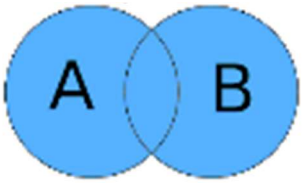

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

RIGHT OUTER JOIN : affiche toutes les lignes de B + les correspondances avec A




```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

FULL OUTER JOIN : affiche toutes les lignes de A et les correspondances avec B + toutes les lignes de B




```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
```

EQUI-JOIN : idem que inner join mais uniquement le signe =

```
SELECT C.course_name, P.professor_name, S.section_name
  FROM (course C JOIN professor P
        ON C.professor_id = P.professor_id)
          JOIN section S
          ON P.section_id = S.section_id
```

NON EQUI-JOIN : idem inner mais en utilisant autre chose qu'une égalité stricte.

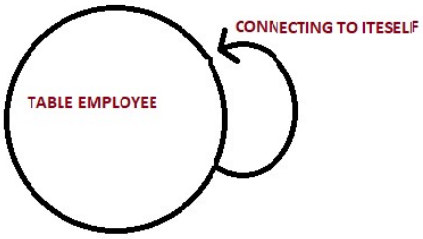
```
SELECT S.last_name, S.year_result, G.Grade
  FROM Grade G JOIN student S
       ON S.year_result BETWEEN G.Lower_bound AND G.Upper_bound
```

SELF-JOIN : comparer les éléments d'une même table

```
SELECT *

FROM table1 T1

JOIN table1 T2 ON T1.col1 = T2.col1
```

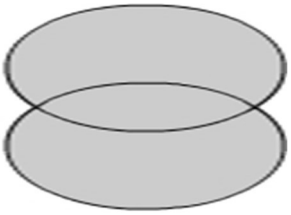


Jointures verticales : comparer 2 requêtes indépendantes

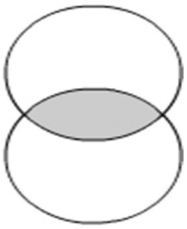
```
SELECT … FROM … WHERE … GROUP BY …
opérateur_comparaison_requêtes
SELECT … FROM … WHERE … GROUP BY …
ORDER BY …
```

UNION : applique un distinct [ALL] : affiche toutes les lignes même les doublons.

```
SELECT col1 as [colonne1_table1], col2 as [colonne2_table1]
FROM table1
UNION
SELECT col1 as [colonne1_table2], col2 as [colonne2_table2]
FROM table2
ORDER BY [colonne2_table1]
```

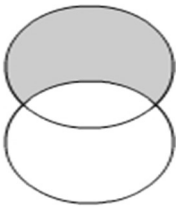


INTERSECT : affiche uniquement ceux compris dans A et B de manière DISTINCT



```
SELECT * FROM table1
INTERSECT
SELECT * FROM table2
```

EXCEPT : affiche uniquement ceux compris dans A mais non dans B de manière DISTINCT



```
SELECT * FROM table1
EXCEPT
SELECT * FROM table2
```

Sous-requêtes : imbriquer une requête dans une requête.

Dans le WHERE

1 valeur

```
SELECT … FROM …
WHERE col <comparaison> (SELECT … FROM … WHERE … GROUP BY … ORDER BY …)
GROUP BY … ORDER BY …
```

[NOT] IN

1 colonne

```
SELECT … FROM …
WHERE col IN (SELECT … FROM … WHERE … GROUP BY … ORDER BY …)
GROUP BY … ORDER BY …
```

ANY / SOME

1 colonne

```
SELECT … FROM …
WHERE col <comparaison> ANY (SELECT … FROM … WHERE … GROUP BY … ORDER BY …)
GROUP BY … ORDER BY …
```

ALL

1 colonne

```
SELECT … FROM …
WHERE col <comparaison> ALL (SELECT … FROM … WHERE … GROUP BY … ORDER BY …)
GROUP BY … ORDER BY …
```

Corrélation : corrélation entre le résultat de la requête et la ligne actuellement traitée

```
SELECT … FROM table1 as T1
WHERE col <comparaison> (SELECT … FROM table1 as T2 WHERE T1.col1 = T2.col1 …) …
GROUP BY … ORDER BY …
```

1 colonne équivalente

[NOT] EXISTS : affiche le résultat uniquement si un résultat est corrélée

```
SELECT … FROM table1 as T1
WHERE EXISTS (SELECT … FROM table2 as T2 WHERE T2.col = T1.col)
```

1 colonne équivalente

Dans le FROM

1 table

```
SELECT …
FROM (SELECT … FROM … WHERE … GROUP BY … ORDER BY …) AS T1
WHERE … GROUP BY … ORDER BY …
```

WITH est équivalent au FROM

```
WITH table_CTE (nom_col1, nom_col2, nom_col3, …, nom_colN)
AS
(SELECT … FROM … WHERE … GROUP BY … ORDER BY …)
SELECT … FROM table_CTE WHERE … GROUP BY … ORDER BY …
```

Dans le HAVING

1 valeur

```
SELECT … FROM … WHERE …
GROUP BY … HAVING val <comparaison> (SELECT … FROM … WHERE … GROUP BY … ORDER BY …)
GROUP BY … ORDER BY …
```

Partie 4 : DML : Data Manipulation Language (Insérer, mettre à jour, suppression de données)

Insertion de données

```
INSERT INTO table (col1, col2, …, colN) VALUES
(valeur1_col1, valeur1_col2, …, valeur1_colN),
(valeur2_col1, valeur2_col2, …, valeur2_colN), …
```

ou

```
INSERT INTO table (col2, col1) VALUES (valeur1_col2, valeur1_col1)
```

ou

```
INSERT INTO table VALUES (valeur1_col1, valeur1_col2, …, valeur1_colN)
```

DEFAULT : prend la valeur par défaut à insérer.

```
INSERT INTO table VALUES (valeur1_col1, DEFAULT, …, DEFAULT)
```

SELECT dans INSERT

```
INSERT INTO table1 (col1, col2, col3)
VALUES (valeur1_cal 1, DEFAULT, (SELECT col FROM table2 WHERE coltable2
like 'truc'))
```

ou

```
INSERT INTO table1 (col1, col2)
SELECT DISTINCT col1, col2 FROM table2 T2 JOIN table3 T3 ON T2.col = T3.col
```

Mise à jour de données

```
UPDATE table
SET col1 = nouvelle_valeur_col1, col2 = nouvelle_valeur_col2, …
WHERE …
```

SELECT dans UPDATE

```
UPDATE table
SET col1 = (SELECT col FROM table2 WHERE <condition>), col2 =
nouvelle_valeur_col2, …
WHERE …
```

Suppression de de données

```
DELETE FROM table
WHERE …
```

OUTPUT : récupère une table temporaire lors d'un INSERT

Partie 5 : Notions avancées

Gestion des transactions

```
BEGIN TRANSACTION [;]  // Débuter de la transaction
COMMIT [ TRAN | TRANSACTION | WORK ] [;]  //valider les requêtes
ROLLBACK [ TRAN | TRANSACTION | WORK ] [;] //annuler les requêtes
```

Fusion de données

```
MERGE INTO table_cible AS alias_table_cible
USING (données_à_comparer) AS alias_table_source
ON alias_table_cible.colonne_comparée = alias_table_source.colonne_comparée
WHEN MATCHED THEN …
WHEN NOT MATCHED THEN …
```