Transact SQL

Langage SQL

Participants et pré requis

- **Participants**
- Chargé de reporting ou d'analyse, assistant(e), toute personne ayant des besoins d'interrogation ou de mises à jour simples d'une base de données avec le langage SQL.

- **Prérequis**
- Aucune connaissance particulière.
- Formation commune à toutes les bases relationnelles (Oracle, SQL Server, DB2, PostGreSQL, MySQL, Access, SQL Lite...).

Langage SQL

La Formatrice

- Laure BERENGUER, consultante et formatrice indépendante

- Page du site internet (exercices et support) :

http://berenguer-formation-conseil.fr/langage-sql/

- laure@berenguer.onmicrosoft.com
- http://www.linkedin.com/in/laure-berenguer38/

Langage SQL

Objectif de la formation

- Comprendre le principe et le contenu d'une base de données relationnelle
- Créer des requêtes pour extraire des données suivant différents critères
- Utiliser des calculs simples et des agrégations de données
- Réaliser des requêtes avec des jointures, pour restituer les informations de plusieurs tables
- Combiner les résultats de plusieurs requêtes

Langage SQL

Le Sommaire

Langage SQL

INTRODUCTION AUX BASES DE DONNÉES

Eléments constitutifs

Requêtes mono-tables

- Les tables
- Les champs (colonnes) et les types de données
- Les enregistrements (lignes)

Requêtes multi-tables :

- Le modèle Physique de données (MPD) **indispensable**
- Les clés : clés primaires et clés étrangères
- Les relations entre les tables

Qu'est-ce qu'un serveur et une base de données ?

- Un serveur de gestion de base de données (SGBD) est un logiciel qui permet de stocker et gérer les bases de données sur un serveur.

- Une base de données est un ensemble structuré et organisé en tables permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation : ajout, mise à jour, recherche et analyse des données.

Qu'est-ce qu'une table et des champs ?

- Une **table** correspondant à une **entité particulière** : client, produit, commande, employé, etc.
- Chaque **enregistrement/ligne d'une table** correspondant à un **individu statistique différent.**
- Chaque enregistrement contient un **identifiant** – souvent la première colonne/champ de la table – puis **d'autres colonnes** qui apportent des informations relatives à l'individu/enregistrement en question.

IdClient	Nom	Prénom	Ville
1	DUPONT	JOHN	PARIS
2	DUPONT	PAUL	LYON
3	BART	ELODIE	MARSEILLE

Qu'est-ce que les clés et les relations ?

- Une **clé primaire (Primary Key = PK)** est l'identifiant de l'enregistrement d'une table : l'identifiant/numéro du client de la table client. Chaque clé primaire **est unique** et il ne peut pas y avoir deux fois le même client dans la table.

- Une **clé étrangère** (**FK**) est l'identifiant d'un enregistrement provenant d'une autre table. Elle fait toujours référence à une PK dans une autre table : l'identifiant du client dans la table commande. Elle peut donc être en doublons si le client a commandé plusieurs fois par exemple.

- C'est la relation d'égalité entre les deux qui permet la relation entre la table client et la table commande.

IdClient : clé primaire	Nom	Prénom	Ville
1	DUPONT	JOHN	PARIS
2	DUPONT	PAUL	LYON
3	BART	ELODIE	MARSEILLE

NumCommande	Idlcient : clé étrangère	Date
12	1	30-déc-21
13	1	31-déc-21
14	2	31-déc-21

Base de données : Structure

Les types de données courants

Texte

- Char(n) 0 à 255
- Char(10) : 10 octets sur le disque /et en mémoire même si la valeur du champ est « Paris »
- Varchar(10) : 5 octets sur disque/mémoire si le champ contient « Paris »

- **Numérique (entier ou décimal)**
- Int : integer (Entier)
- Decimal (x,y) : Decimal (5,2) indique 5 chiffres avant la virgule et une précision à 2 chiffres après la virgule
- Numeric (x,y) : Identique à Decimal
- Float : Réservé aux calculs scientifiques (très gourmand en mémoire)

- **Date**
  - DateTime (Année, mois, jour, heures, minutes, secondes, ms)
  - Date (Année, mois, jour)

Base de données :
Conclusion

Une base de données est :

- Un ensemble de données organisées et reliées
- Dans (et entre) différentes tables composées
- De champs (dont certains clés primaires)
- De types numériques, textes ou dates

Qui nous permet d'enregistrer, modifier et d'extraire de l'information !

Langage SQL

EXTRAIRE LES DONNEES D'UNE TABLE

Extraire les données d'une table

Pour extraire les données d'une (ou plusieurs) table en SQL il faut :

Bien connaître le modèle physique (MPD) du sous système de base de données, le contenu, les règles métiers..

Et bien savoir ce que l'on veut !!

SELECT et la syntaxe SQL

SELECT est la commande de base du SQL destinée à **extraire des données**

3 mots clés :

```
SELECT
        Champ1,
        Champ2
 FROM
        Table
 WHERE
        Champ3 ='test'
```

Cette requête SQL va **sélectionner et afficher** (SELECT) le champ « nom_du_champ »

provenant (FROM) du tableau appelé « nom_du_tableau ».

Contenant (**condition**) quand le «nom_du_champ» est

SELECT et la syntaxe SQL (2)

3 types de select:

SELECT *
FROM

Retourne et affiche l'ensemble des champs (colonnes) et des lignes. (A éviter)

SELECT TOP 100 *
FROM

Retourne l'ensemble des champs (colonnes) et seulement les 100 premières lignes : permet de visualiser les données et ce que contient la table sans problématique de performance.

Filtrage des données :
La clause DISTINCT

```
use comptoir

select Distinct Clientpays
From T_Client
```

Clientpays
Argentina
Austria
Belgium
Brazil
Canada
Denmark
DEUTSCHLAND
Finland
FRANCE

Le DISTINCT se fait sur l'ensemble des champs du Select

Il permet de supprimer les doublons sur les lignes.

Les commentaires en SQL

- Introduits par « -- » (une seule ligne
- Entourés par /*,,,,,,,,,*/ (plusieurs lignes)

```
/*
Afficher les 100 premieres lignes
Nom, region, Pays à partir de la table client
*/
-- Triées par pays (croissant)
use comptoir
select TOP 100
ClientNom,
ClientRegion,
ClientPays

from T_Client
order by ClientPays
```

La commande SELECT :

La clause TOP à travers différents SGBD

```
-- SQL SERVEUR
Select TOP 10 *
From DimCustomer

-- ORACLE
Select *
From DimCustomer
WHERE ROWNUM <=10

-- MYSQL
Select *
From DimCustomer
LIMIT 10
```

La commande SELECT :

Sensibilité à la casse (majuscule / minuscule)

Ces 2 requêtes produisent un résultat **différent** si le serveur (ou la base) est sensible a la casse

```
use COMPTOIR_OLTP
select * from
T_Client
Where ClientVille ='paris'
```

ou

```
use comptoir_OLTP
select * from
t_CLIENT
Where ClientVille ='Paris'
```

Renommer les colonnes

```
-- Il existe deux manières de renommer les colonnes
:

use comptoir
Select CLIST as 'Nom Client',
   TILL as 'Tel Client'
From T_Client



Select 'Nom Client'= CLIST,
   'Tel Client'= TILL
From T_Client
```

La commande SELECT :

La clause ORDER BY

```
-- La clause ORDER BY permet de trier les données. C'est
la dernière clause dans l'ordre de la syntaxe SQL :

use comptoir
Select ClientNom, ClientRegion, clientpays
From T_Client
order by ClientPays, ClientVille desc

-- ASC signifie ascendant : pas besoin de l'écrire dans
SQL SERVER.
-- DESC signifie descendant : besoin de le mentionner car
par défaut, la clause ORDER BY trie par ordre ascendant.
```

Attention, pour les autres SGBD, mettre ***ORDER BY clientpays ASC***

La commande SELECT :

La clause WHERE : Les critères simples

```
-- Lister les clients non parisiens dont le nom
commence par P ou F

SELECT * FROM T_Client
WHERE ClientPays = 'France' AND
  ClientVille <> 'Paris' AND
  LEFT(clientnom,1) IN ('P','F')
```

LEFT : Je me positionne à gauche du champ clientnom et je prends le 1er caractère. S'il correspond à P ou à F alors je le sélectionne.

Texte :

='France'

<>'Spain'

Numérique :

='25'
>'50'
< '50'
<>'25'

La commande SELECT :

La clause WHERE : (not) IN

```
select * from t_client
where ClientPays
Not in ('France','italy')
```

Je sélectionne toutes les colonnes et lignes de ma table t_client.
Quand le pays du client n'est pas, ni l'italy, ni la France.

```
Select *

from Orders

where
Year(orderdate) in (2008,2009,2012)
and
Month(orderdate)  in (4,10)
```

Year & Month : fonctions intégrées présentées dans la section ***Fonctions intégrées de date.***
Elles permettent d'aller chercher une année dans une date de type DateTime ou Date… Ici je sélectionne les années 2008,2009 et 2012 dans le champ order date puis les mois d'avril et d'octobre.

La commande SELECT :

La clause WHERE : (not)LIKE

```
-- La commande LIKE permet de sélectionner ce qui contient :

SELECT * FROM T_Client
WHERE ClientNom like 'B%'
-- Ce qui commence par B car ensuite pourcentage qui
signifie "qu'importe ce qu'il y a après".

SELECT * FROM T_Client
WHERE ClientNom like '%S'
-- Ce qui finit par B car ensuite pourcentage qui signifie
"qu'importe ce qu'il y a avant".

SELECT * FROM T_Client
WHERE ClientNom like '%S%'
-- Ce qui commence par B car ensuite pourcentage qui
signifie "qu'importe ce qu'il y a avant et après".
```

La commande SELECT :

La clause WHERE : Between Vs > & < (Entre)

```
-- La commande Between va permettre de gérer des
intervalles et éviter d'écrire deux conditions
(plus grand et plus petit) :

select *
From T_Commande
where CdeDate >= '20040705'
AND CdeDate < '20040715'

select *
From T_Commande
where CdeDate between '20040705' AND '20040714'

-- La commande Between prendra 3 fois moins de
temps à l'exécution que l'exemple au dessus.
```

CdeNum	ClientID	EmployeID	CdeDate
10249	TOMSP	6	05/07/2004
10250	HANAR	4	08/07/2004
10251	VICTE	3	08/07/2004
10252	SUPRD	4	09/07/2004
10253	HANAR	3	10/07/2004
10254	CHOPS	5	11/07/2004
10255	RICSU	9	12/07/2004

La commande SELECT :

La clause WHERE : Synthèse

= <>	exactement égal à une seule valeur *where ClientPays <> 'France'*
IN / NOT IN	exactement égal à un ensemble de valeurs (remplace un or) *where ClientPays not in ('France',' Espagne')* ~~*where (ClientPays <> 'France' or ClientPays <> 'Espagne)*~~
LIKE / NOT LIKE	contient (et commence) par telle(s) valeur(s) *where ClientPays like '%rance%'* *where (ClientPays not like 'Franc%'* *or* *ClientPays not like '%pagne')*
Between .. and…	entre telle et telle valeur *where nombers between 5 and 7*

La commande SELECT :

Règles d'écritures : (erreurs surlignées en rouge)
Les erreurs de code sont souvent les mêmes

```
use ContosoRetailDW
select
FirstName,
LastName,
Education
From DimCustomer
Where Education = 'Bachelor'

use comptoir
Select
YEAR(Cdedate) as Année,
COUNT(Cdenum) as 'Nombre de
commandes'
From T_commande
Group by YEAR(cdedate)
```

Les champs sélectionnés doivent être séparés d'une virgule (comme lorsque l'on énumère plusieurs exemples (sauf pour le dernier qui clôture)

Lorsque l'on renomme un champ, calcul ou lorsque l'on met une condition : ne pas oublier de mettre des cotes

Lorsque l'on fait un calcul (utilisation d'une fonction d'agrégation) = il faut bien remettre les autres champs du select dans le Group By

La commande SELECT :

La clause CASE

Le CASE permet de **créer une nouvelle colonne** à partir de **conditions sur d'autres colonnes.**

Si dans la colonne genre, il y a la valeur M, alors Homme dans nouvelle colonne…

Si le montant des ventes par client est supérieur a tant, alors bon client…

```
select CdeNum, SUM(qte_vente) as
                        'somme des ventes',
    CASE
WHEN SUM(Qte_vente) < 50 THEN 'Faible'
WHEN SUM(Qte_vente) > 50 THEN 'Elevé'
ELSE 'Modéré'
    END as 'Qualification du montant de
ventes'

From T_DetaillerCommande
Group by CdeNum
```

CdeNum	somme des ventes	Qualification du montant de ventes
10411	74	Elevé
10743	28	Faible
11075	42	Faible
10388	75	Elevé
10720	29	Faible
11052	40	Faible
10457	36	Faible
10789	93	Elevé
10434	24	Faible
10766	115	Elevé
10835	17	Faible
10906	15	Faible
10912	100	Elevé
10265	50	Modéré

La commande SELECT :

La clause CASE (exemples dates)

```
-- Combien de commandes par type d'envoi ?
select COUNT(*) as 'nombre de commandes',

CASE
WHEN CommandeShippedDate IS NULL THEN 'Commande non
                                       envoyée'
WHEN DATEDIFF(Day,cdedate,CommandeShippedDate) < 5 THEN
                                       'Envoi rapide'
WHEN DATEDIFF(Day,cdedate,CommandeShippedDate) < 10 THEN
                                       'Envoi moyen'
ELSE 'Envoi lent'
   END as 'Qualification du montant de ventes’

From T_Commande

group by
CASE WHEN CommandeShippedDate IS NULL THEN 'Commande non envoyée'
WHEN DATEDIFF(Day,cdedate,CommandeShippedDate) < 5 THEN 'Envoi rapide'
WHEN DATEDIFF(Day,cdedate,CommandeShippedDate) < 10 THEN 'Envoi moyen'
ELSE 'Envoi lent’ END
```

La commande SELECT :

La clause CASE (Oracle)

```
-- On peut mettre autant de conditions sur autant de colonnes
que l'on souhaite pour créer notre colonne conditionnelle :

select COUNT(customerkey) as 'nombre de clients',
   CASE
WHEN gender IS NULL THEN 'Entreprise'
WHEN Gender='M' AND MaritalStatus='M' THEN 'Homme marié'
WHEN Gender='F' AND MaritalStatus='M' THEN 'Femme mariée'
WHEN Gender='M' AND MaritalStatus='S' THEN 'Homme
                                           célibataire'
WHEN Gender='F' AND MaritalStatus='S' THEN 'Femme
                                           célibataire'
   END as 'genre et situation familiale'
From DimCustomer

Group by Gender, MaritalStatus
```

Langage SQL

LES CALCULS ET FONCTIONS INTÉGRÉES

Les calculs / Fonctions intégrées

- Calculs simples
- Fonctions d'agrégations
- Fonctions DATE
- Fonctions TEXTE
- Fonctions CONVERT

Les calculs / fonctions intégrées

Champs calculé simple

```
-- Possibilité de faire des soustractions,
multiplications et divisions assez facilement :

select top 10
SalesAmount as 'vente',
DiscountAmount as 'remise',
SalesAmount - DiscountAmount as 'hors remise'
From FactOnlineSales
```

vente	remise	hors remise
10,36	2,59	7,77
3,984	0,996	2,988
39,968	9,992	29,976
23,992	5,998	17,994
207,992	51,998	155,994
319,2	79,8	239,4
3,792	0,948	2,844
0	0,948	-0,948
166,4	41,6	124,8
0	41,6	-41,6

Les calculs / fonctions intégrées

Les fonctions d'agrégations

Les instructions d'agrégation permettent des opérations comme le comptage ou les sommes.

Les principales fonctions d'agrégation sont :

- AVG(<champs>), COUNT(<champs>), SUM (<champs>), MIN et MAX.

```
-- nombres de commandes passées en 2004
Select COUNT(cdenum) as 'nombre de commandes'
from t_commande
where YEAR(cdedate) = 2004

-- Le poids moyen des commandes par ville
Select AVG(CommandePoids) as 'Moy poids',ShipCity
from T_Commande
group by ShipCity

-- Chiffre d'affaires global
select SUM(Qte_vente * PU_Vente) as CA
from T_DetaillerCommande
```

Les calculs / fonctions intégrées

Les fonctions d'agrégations → GROUP BY : Where et Having

```
select ColorName,
   COUNT(productkey) as 'nombre de produits'
from DimProduct
where ProductKey between 2 and 1216
-- la clause where est en 3ème position dans la
syntaxe, AVANT le regroupement.
-- Il s'effectue sur un CHAMP de la base de données.
→ Quand l'identifiant du produit est égal à un numéro
entre 2 et 1216.
group by colorname

having COUNT(productkey) > 50
-- la clause having est en 5ème position dans la
syntaxe, APRES le regroupement.
-- Il s'effectue sur une FONCTION D'AGREGATION, calcul
après regroupement.
→ Quand le nombre de produit - count(productkey) – PAR
couleur est supérieur à 50.
```

Les calculs / fonctions intégrées

Les fonctions d'agrégations

GROUP BY ou La clause OVER (partition order by)

```
SELECT
 salesorderID,
 sum(orderqty) as 'total',
 avg(orderqty) as 'moyenne',
 count(Orderqty) as 'nombre',
 min(orderqty) as 'min',
 max(Orderqty) as 'max'
 FROM Sales.SalesOrderDetail
 WHERE SalesOrderID IN(43659,43664)
 group by salesorderId
 order by SalesOrderID
```

Utilisation du group by : résultat

Mais nous pouvons aussi utiliser un partitionnement : Clause OVER

	SalesOrderID	ProductID	OrderQty	Total	Avg	Count	Min	Max
1	43659	776	1	26	2	12	1	6
2	43659	777	3	26	2	12	1	6
3	43659	778	1	26	2	12	1	6
4	43659	771	1	26	2	12	1	6
5	43659	772	1	26	2	12	1	6
6	43659	773	2	26	2	12	1	6
7	43659	774	1	26	2	12	1	6
8	43659	714	3	26	2	12	1	6
9	43659	716	1	26	2	12	1	6
10	43659	709	6	26	2	12	1	6
11	43659	712	2	26	2	12	1	6
12	43659	711	4	26	2	12	1	6

```
SELECT
     SalesOrderID,
     ProductID,
     OrderQty
     ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID order by salesorderID) AS 'Total'
     ,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Avg'
     ,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Count'
     ,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Min'
     ,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Max'
 FROM Sales.SalesOrderDetail
 WHERE SalesOrderID IN(43659,43664);
```

Les calculs / fonctions intégrées

Les fonctions d'agrégations

GROUP BY ou La clause OVER (partition order by)

Les clauses Row_number et rank

```
select ClientNom, ClientID
from T_Client
where ClientID like 'A%'
order by ClientNom asc

select
   ROW_NUMBER() OVER(ORDER BY clientnom ASC) AS Row#, clientNom, clientID
from T_Client
where ClientID like 'A%'
```

150 %

Résultats | Messages

	Row#	clientNom	clientID
1	1	Alfreds Futterkiste	ALFKI
2	2	Ana Trujillo Emparedados y helados	ANATR
3	3	Antonio Moreno Taquería	ANTON
4	4	Around the Horn	AROUT

Les calculs / fonctions intégrées

Les fonctions : Date (SQL Server)

```
select
CdeNum,
CdeDate,
GETDATE() as 'date jour serveur',
year(cdedate) as 'year',
month(cdedate) as 'mois',
Datepart("weekday", cdedate) as 'jour semaine',
Datepart("Year", cdedate) as 'année',
Datepart("QUARTER", cdedate) as 'trimestre',
DATEDIFF(day,cdedate,GETDATE()) as 'delai en jours',
DATEDIFF(Month,cdedate,GETDATE()) as 'delai en mois',
DATEDIFF(YEAR,cdedate,GETDATE()) as 'delai en année'
From T_Commande
```

CdeNum	CdeDate	date jour serveur	year	mois	jour semaine	année	trimestre	delai en jours	délai en mois	délai en année
10249	05/07/2004	2021-12-31 14:13:13.653	2004	7	1	2004	3	6388	209	17
10250	08/07/2004	2021-12-31 14:13:13.653	2004	7	4	2004	3	6385	209	17
10319	02/10/2004	2021-12-31 14:13:13.653	2004	10	6	2004	4	6299	206	17
10875	06/02/2006	2021-12-31 14:13:13.653	2006	2	1	2006	1	5807	190	15

Les calculs / fonctions intégrées

Les fonctions : Date (SQL Server) : Exemples

```
-- Que permettent de faire les requêtes suivantes ?
select top 100 Laminate_Id, Start_Time
From T_Commande
Where DATEDIFF(MINUTE, Start_Time,getdate())< 60

select CdeNum, DATEDIFF(day, CdeDate,
CommandeShippedDate) as délai
From T_Commande
where YEAR(cdedate) = 2006

select employenom
From T_Employe
where DATEDIFF(year, employedteentree, getdate()) > 15

-- La commande datediff permet de calculer un
délai/différence entre 2 dates.
-- Il faut préciser 3 éléments : L'intervalle (année,
mois, minutes, etc.), la date de début, la date de fin.
```

Les calculs / fonctions intégrées

Chaînes de caractères (SQL Server)

```
select
LEFT(ClientPays, 3) as '3ères car gch',
SUBSTRING(clientpays,3,2) as 'extrait du texte',
UPPER(clientpays) as 'en maj',
LOWER(clientpays) as 'en min',
LEN(clientpays) as 'nb caractères’
clientcontactfonction + clientid, -- espaces du champ de
type CHAR à droite…
Rtrim(clientcontactfonction) + ' ' + clientid as 'supp
                                    espaces à droite’

From T_Client
```

3ères car gch	extrait du texte	en maj	en min	nb caractères	(No column name)		supp espaces drte
DEU	UT	DEUTSCHLAND	deutschland	11	Sales Representative	ALFKI	Sales Representative ALFKI
Mex	xi	MEXICO	mexico	6	Owner	ANATR	Owner ANATR
Mex	xi	MEXICO	mexico	6	Owner	ANTON	Owner ANTON
UK		UK	uk	2	Sales Representative	AROUT	Sales Representative AROUT

Les calculs / fonctions intégrées

Convertir : La fonction CAST

```
select
cast(25.2 as int) + cast(15.2 as int) + cast(33.6 as int) as
result
→ 73: l'arrondi est calculé au nombre inférieur

----------------------------------------
select getdate() as 'date entière',
convert (date, getdate()) as 'date sans heures minutes
secondes',
-- autres types de conversions mais en TEXTE attention :
convert(varchar(20),getdate(),108) 'date format 108',
convert(varchar(20),getdate(),107) 'date format 107',
convert(varchar(20),getdate(),113) 'date format 113',
convert(varchar(20),getdate(),103) 'date format 103'
```

date entière	date sans heures minutes secondes	date format 108	date format 107	date format 113	date format 103
2021-12-31 14:44:16.620	31/12/2021	14:44:16	déc 31, 2021	31/12/2021 14:44	31/12/2021

Langage SQL

LES JOINTURES

Structure d'une table : les clés

- Une **clé primaire (Primary Key = PK)** est l'identifiant de l'enregistrement d'une table. Une **clé étrangère** (**FK**) est l'identifiant d'un enregistrement provenant d'une autre table.
- C'est la relation d'égalité entre les deux qui permet la relation entre la table client et la table commande.

- La cardinalité permet de comprendre la relation entre clé primaire et clé étrangère : 1- N signifiant une ligne dans table1(client – PK) peut correspondre à plusieurs lignes (max) dans table2 (commande – FK).

IdClient : clé primaire PK	Nom	Prénom	Ville
1	DUPONT	JOHN	PARIS
2	DUPONT	PAUL	LYON
3	BART	ELODIE	MARSEILLE

NumCommande	Idlcient : clé étrangère FK	Date
12	1	30-déc-21
13	1	31-déc-21
14	2	31-déc-21

Les jointures

Rôle des clés

CLIENT

Idclt	Prénom
1	John
2	Paul
3	Elodie

COMMANDE

Id cmd	Date	Idclt	Idp
12	30-déc-21	1	2
13	31-déc-21	1	1
14	31-déc-21	2	3

PRODUIT

IDp	Produit
1	Vélo
2	Trotinette
3	Skate

PK (clés primaires) en rouge. *FK (clés étrangères) en vert.*

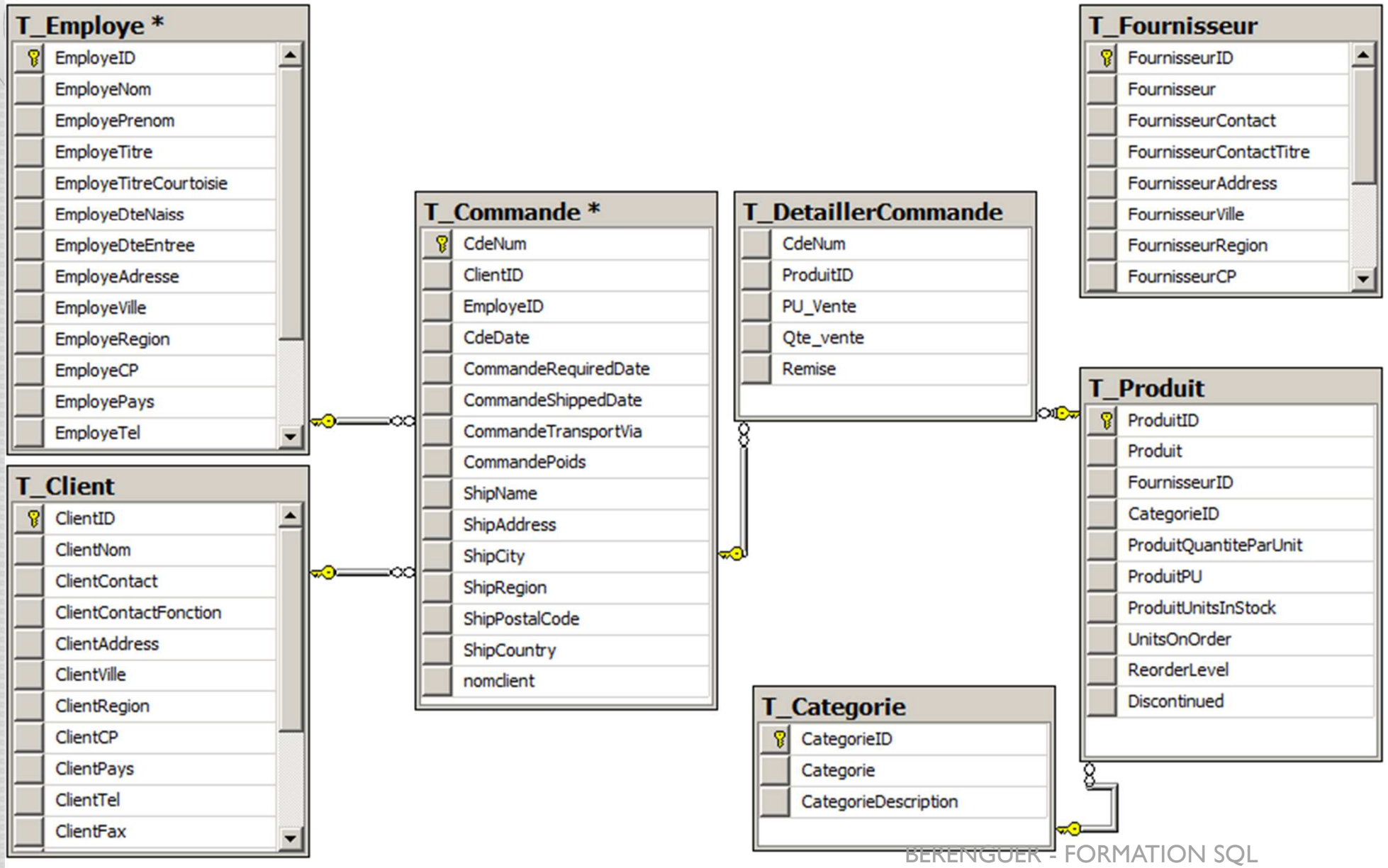
- L'égalité a retranscrire est entre la clé primaire et la clé étrangère.
- Si les deux champs contenant ses clés ont le même nom alors il faudra préciser le nom de la table afin que le code soit le plus précis possible sinon cela ne marchera pas.

Nomtable1 . nomchampPK = Nomtable2 . nomchampFK

CLIENT.idclt = COMMANDE.idclt

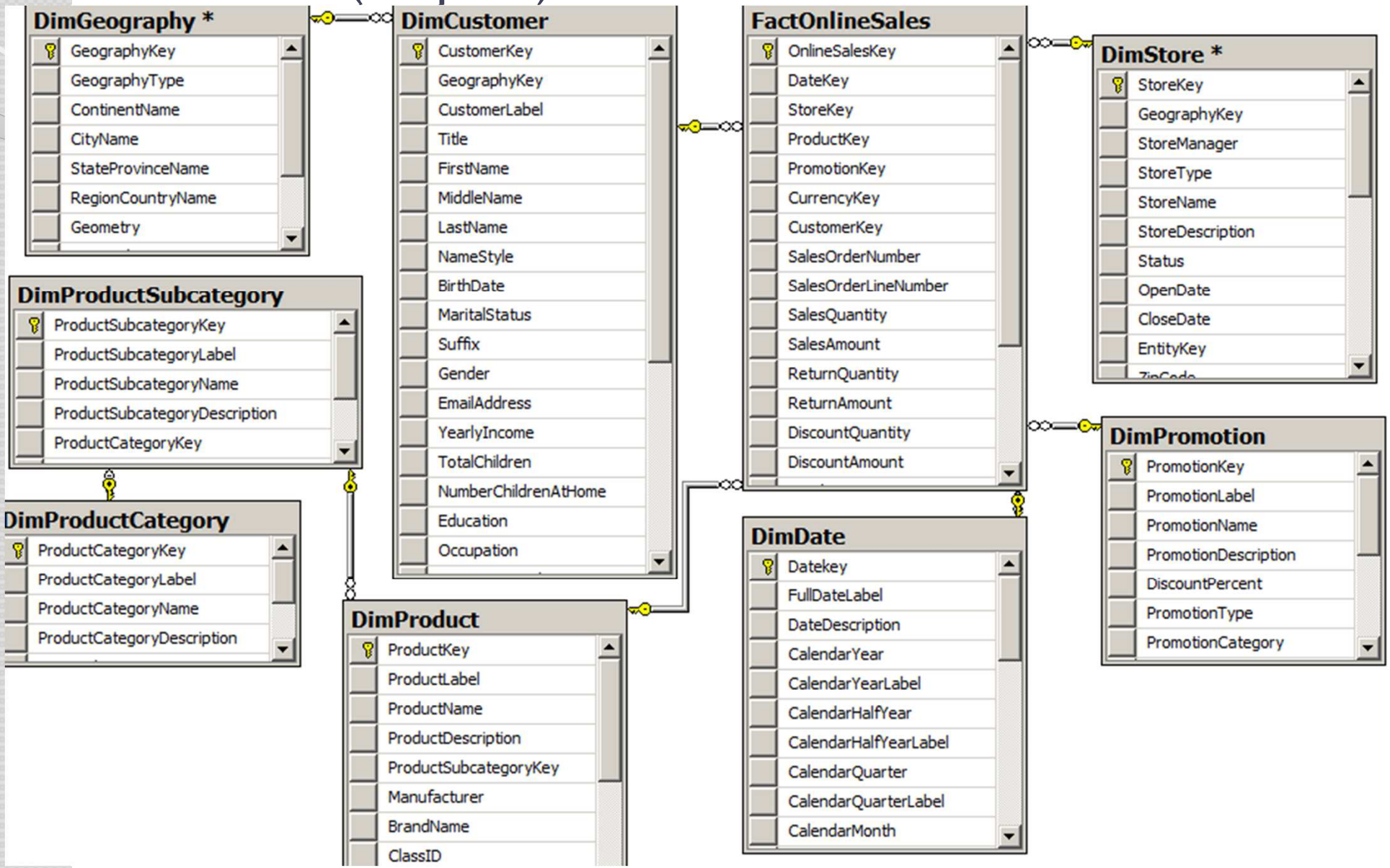
Les jointures

Lire un modèle relationnel (ou physique / MPD)



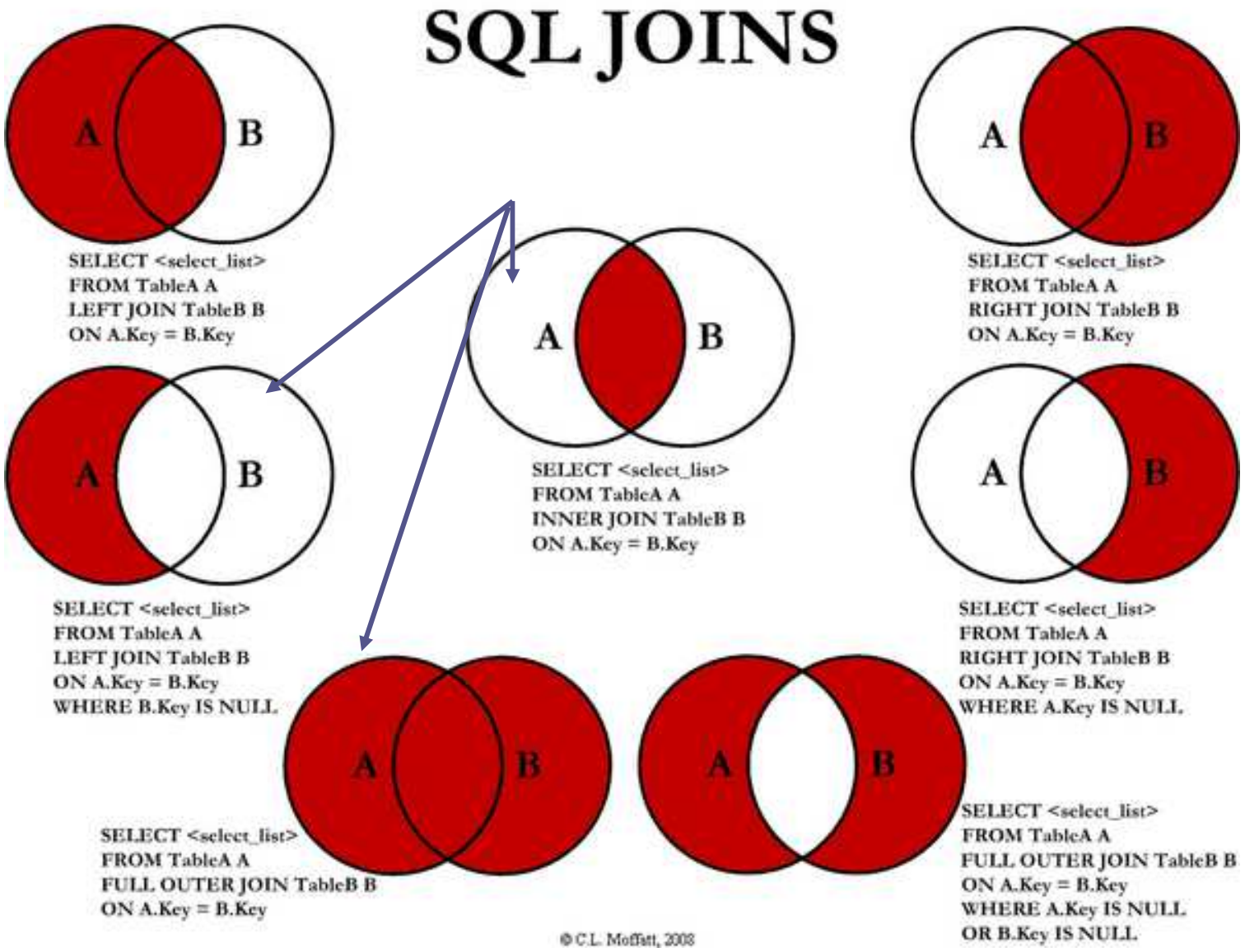
Les jointures

Le MPD (simplifié) de la base ContosoRetailDW



Les jointures

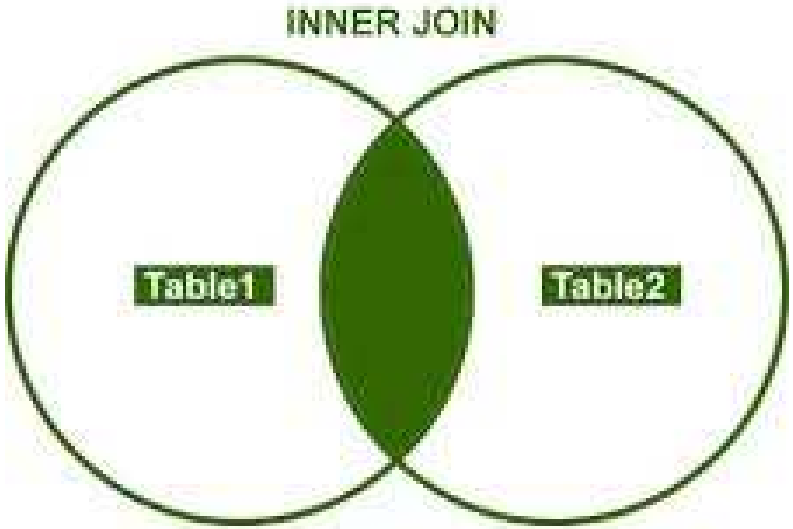
Les jointures Vue d'ensemble



Les jointures

INNER JOIN (schéma) Inclusion

C'est **LA** Jointure **par défaut** qui compare deux tables et retourne tous les enregistrements comportant une concordance (en règle **quasi générale**) PK vers FK



Les jointures

INNER JOIN (schéma) Inclusion

Figure 2. INNER JOIN Example

```
SELECT PROJNO, PROJNAME, P.DEPTNO, D.DEPTNO, DEPTNAME
FROM  PROJECT P INNER JOIN DEPARTMENT D
ON      P.DEPTNO = D.DEPTNO
```

PROJNO	PROJNAME	DEPTNO
AD3100	ADMIN SERVICES	D01
IF1000	QUERY SERVICES	C01
IF2000	USER EDUCATION	E01
MA2100	WELD LINE AUTOMATION	D01
PL2100	WELD LINE PLANNING	B01

DEPTNO	DEPTNAME
A00	SPIFFY COMPUTER SERVICE DIV.
B01	PLANNING
C01	INFORMATION CENTER
D01	DEVELOPMENT CENTER

ON

PROJNO	PROJNAME	P.DEPTNO	D.DEPTNO	DEPTNAME
AD3100	ADMIN SERVICES	D01	D01	DEVELOPMENT CENTER
IF1000	QUERY SERVICES	C01	C01	INFORMATION CENTER
MA2100	WELD LINE AUTOMATION	D01	D01	DEVELOPMENT CENTER
PL2100	WELD LINE PLANNING	B01	B01	PLANNING

Les jointures

Renommer les tables (alias)

```
select DAT.CalendarYear, STO.StoreName,
sum(FAC.SalesAmount) as 'total montant ventes'

From FactSales as FAC
        JOIN DimProduct as PROD
  ON FAC.ProductKey = PROD.ProductKey
        JOIN DimPromotion as PROM
  ON PROM.PromotionKey = FAC.PromotionKey
        JOIN DimDate as DAT
  ON DAT.Datekey = FAC.DateKey
        JOIN DimStore as STO
  ON STO.StoreKey = FAC.StoreKey

where MONTH(dat.datekey) = 3
Group by DAT.CalendarYear, STO.StoreName
ORDER BY DAT.CalendarYear, STO.StoreName
```

Exercice :

- Dessiner le schéma (tables et relations),
- Indiquer et cocher les clés,
- Lister champs dans la requête,
- Dites-moi s'il y a une erreur dans le code

Les jointures

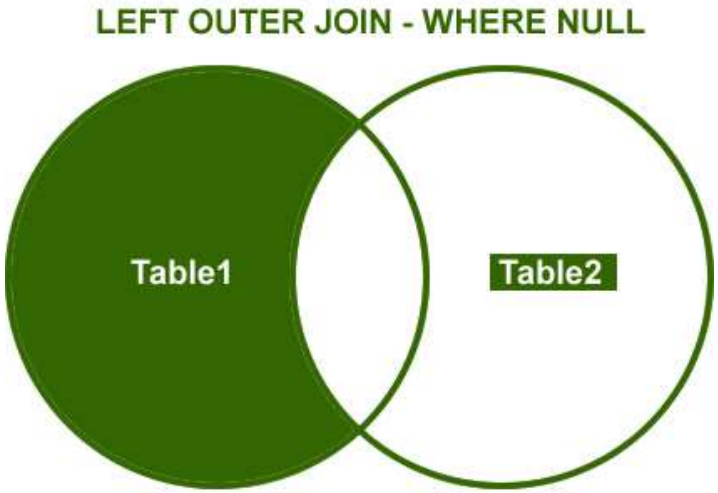
Renommer les tables (alias « métier »)

```
Select
A.NUM, B.SDHNUM, C.DET

From SINVOICE A
JOIN SVOICE B
   ON A.NUM=B.NUM
JOIN SDELIVERYD C
   ON B.SDHNUM = C.SDHNUM
  AND B.SDDLIN = C.SDDLIN

WHERE A.NUM = 'KELMKE'
```

Les jointures

LEFTJOIN : l'exclusion

La commande LEFT JOIN (aussi appelée LEFT OUTER JOIN) est un type de jointure entre 2 tables. Cela permet de lister tous les résultats de la table de gauche (left = gauche) s'il n'y a pas de correspondance dans la deuxième tables.



Les jointures

LEFT (outer) JOIN (exemple)

Figure 3. LEFT OUTER JOIN Example

```
SELECT PROJNO, PROJNAME, P.DEPTNO, D.DEPTNO, DEPTNAME
FROM  PROJECT P LEFT OUTER JOIN DEPARTMENT D
ON      P.DEPTNO = D.DEPTNO
```

PROJNO	PROJNAME	DEPTNO
AD3100	ADMIN SERVICES	D01
IF1000	QUERY SERVICES	C01
IF2000	USER EDUCATION	E01
MA2100	WELD LINE AUTOMATION	D01
PL2100	WELD LINE PLANNING	B01

DEPTNO	DEPTNAME
A00	SPIFFY COMPUTER SERVICE DIV.
B01	PLANNING
C01	INFORMATION CENTER
D01	DEVELOPMENT CENTER

Preserved Row Table

ON

NULL-Supplying Table

PROJNO	PROJNAME	P.DEPTNO	D.DEPTNO	DEPTNAME
AD3100	ADMIN SERVICES	D01	D01	DEVELOPMENT CENTER
IF1000	QUERY SERVICES	C01	C01	INFORMATION CENTER
IF2000	USER EDUCATION	E01	-	-
MA2100	WELD LINE AUTOMATION	D01	D01	DEVELOPMENT CENTER
PL2100	WELD LINE PLANNING	B01	B01	PLANNING

Les jointures

LEFT JOIN : Exemple

```
Select

A.SOHNUM as 'numéro de commande sur table commande',

B.SOHNUM as 'numéro de commande sur table livraison'


From SORDER A LEFT JOIN DELIVERYD B
        ON A.SOHNUM = B.SOHNUM

WHERE B.SOHNUM IS NULL
```

SELECT : J'affiche des champs provenant des deux tables.

FROM : Je prends toutes les lignes de la table A (ordres de commandes) : LEFT indiquant de prendre ce qui est à gauche du JOIN.

WHERE : Quand le champs SOHNUM est inexistant (NULL) dans la table b (livraisons)

→ Les commandes qui n'ont pas été livrées !

Les jointures

« Old School »

```
Select A.ClientNom, B.CdeDate, C.employenom

From t_client A, T_commande B, T_employe C
-- les tables sont toutes appelées les unes après les
autres dans le From, séparées de virgule

where A.ClientID=B.ClientID And
B.EmployeID=C.EmployeID And
A.ClientPays = 'France’
-- ce qui est dans la clause ON, qui permet d'établir
l'égalité entre clés se trouve içi dans le WHERE
mélangé au reste
```

Les jointures
Synthèse et Méthodologie (1/2)

1. Sur quelles tables sont les informations nécessaires pour mon extraction

(A afficher et pour mes conditions) ?

FROM : Je fais mes jointures (JOIN par défaut) et je crée mes alias pour relier mes tables.

2. Est-ce que j'ai des conditions particulières de sélection des données ?

WHERE : Sur quels critères je souhaite avoir des informations ? Je n'oublie pas de mettre l'alias de la table avant le champ. J'exécute un SELECT simple pour vérifier que mes conditions fonctionnent.

Les jointures
Synthèse et Méthodologie (2/2)

3. Qu'est-ce que je souhaite afficher ? Quelles colonnes / champs ?

SELECT : je liste mes champs en n'oubliant pas de remettre l'alias de la table avant

5. Est-ce que je veux exclure ou inclure des données d'une table par rapport à une autre ?

Est-ce une jointure d'inclusion ? → Je laisse le JOIN

Est-ce une jointure d'exclusion ? → LEFT JOIN ou RIGHT JOIN selon l'ordre d'écriture des tables.

Ne pas oublier ensuite la condition NULL (dans la table où l'on ne souhaite pas retrouver les données.

La commande SELECT :
SELECT UNION / ALL / INTERSECT / EXCEPT

```
select clientid from t_client
                UNION
Select clientid from t_commande
```

UNION → permet de récupérer les lignes des deux requêtes, tout en supprimant les doublons si des éléments (client) apparaissent dans les deux tables.

UNION ALL → permet de faire exactement la même chose, tout en conservant les doublons.

INTERSECT → permet de récupérer que les éléments (client) qui sont dans la première mais aussi la seconde table. Si une ville n'existe que dans la table customers, alors la commande ne permet pas de l'afficher.

EXCEPT → permet de récupérer que les éléments (client) qui sont dans la requête qui précède EXCEPT (celle du haut) mais qui ne sont pas dans la suivante (commande).

Langage SQL

LES SOUS REQUÊTES

Utilisation de sous-requêtes

- Écriture de sous-requêtes simples
- Écriture de sous-requêtes corrélées
- Utilisation du prédicat Exists avec les sous-requêtes
- Utilisation du IN, ALL, ANY, SOME

Sous requêtes :

Dans la clause WHERE : In ou NOT IN (requête simples)
L'inclusion / l'exclusion ET/OU comparer des listings

```
-- Comparaison de listing d'une table sur l'autre donc
possibilité d'écrire une sous-requête dans le where :
Select clientnom
from t_client
where ClientID IN (SELECT distinct ClientID from T_Commande)

-- En version Jointure :
SELECT distinct ClientNom
From T_client JOIN t_commande ON
T_client.clientID=T_commande.clientID
```

-- Seulement dans le cas d'une comparaison entre une table et une autre où il n'y a pas besoin de jointure (et donc peut être remplacé par une jointure) MAIS POUR TOUS LES AUTRES CAS, ce sera obligatoirement une SOUS-REQUETE et possiblement des jointures à l'intérieur ☺

Les performances : De quel côté sont-elles favorables ?

Sous requêtes :

Dans la clause WHERE/Having : = > < et ALL / ANY

```
-- Afficher le nombre de commandes en 2006 que si plus
importante qu'en 2004

-- nombre de commandes en 2006
Select COUNT(cdenum) as 'nombre de commandes'
from T_Commande
where YEAR(cdedate) = 2006
having COUNT(cdenum) >

-- nombre de commandes en 2004
(Select COUNT(cdenum) as 'nombre de commandes'
from T_Commande
where YEAR(cdedate) = 2004)
```

- On peut rajouter ALL : signifie si supérieur à toutes les valeurs de la sous-requête.

- Ou ANY = signifie supérieur à au moins une seule valeur.

Sous requêtes :

La sous-requête corrélée

-- Faire remonter les pays qui avaient plus de commandes en 2006 qu'en 2004 :

```
Select shipcountry,
COUNT(cdenum) as 'nombre de commandes'
from T_Commande
where YEAR(cdedate) = 2006
group by shipcountry
having COUNT(cdenum) >

-- nombre de commandes en 2004
(Select COUNT(cdenum) as 'nombre de commandes'
from T_Commande cmd – Renommer table pour être précis
 where YEAR(cdedate) = 2004
AND cmd.ShipCountry=T_Commande.ShipCountry – Faire une
jointure entre le country de la sous-requête (cmd) et
celui de la requête principale (t_commande)
group by shipcountry)
```

Sous requêtes :

Dans la clause WHERE : EXISTS (requêtes corrélées)

```
SELECT reference_art
  FROM ARTICLES
  WHERE NOT EXISTS(SELECT *
    FROM LIGNES_CDE
    WHERE LIGNES_CDE.reference_art= ARTICLES.reference_art);
```

<u>3 différences avec les requêtes simples :</u>

-Dans la condition where et le prédicat Exists : aucun champ car on va lier les requêtes par les tables entre la 1ère requête et la seconde : requêtes corrélées,

- Dans la sous-requête, on va utiliser l'ancienne syntaxe des jointures : tables énumérées dans le From et clés reliées dans le where,

- Dans le where de la sous-requête : on lie la clé de la table de la sous-requête à la clé de la table de la première requête (corrélation des requêtes)

--> Permet d'avoir plusieurs choses dans le select de la sous-requête

Sous requêtes :

Dans la clause FROM

```
select AVG([nombre de produits]) as 'moyenne'
From

(
Select COUNT(productkey) as 'nombre de produits'
From DimProduct
Group by ColorName
) as toto
```

- Il faut **décomposer le calcul** (étape 1, sous-requête Puis étape 2/ finale : requête principale.

- Pour que **la sous-requête soit considérée comme une table**, cela nécessite :
  - 1: des parenthèses,
  - 2: un nom de table,
  - 3: des noms/entêtes de colonnes

Sous requêtes :

Dans le SELECT

```
select
DC.CustomerKey,
DC.LastName,
avg(salesamount) as 'moyenne cus’,

(select avg(salesamount) from FactOnlineSales)  as
'moy_gnrle_VentesEnLigne',
(select avg(salesamount) from FactSales)  as
'moy_gnrle_VentesMagasin’


from FactOnlineSales FOS join DimCustomer DC
          on FOS.CustomerKey=DC.CustomerKey
group by DC.CustomerKey, DC.LastName
```

Les sous-requêtes
Synthèse et Méthodologie

1.Est-ce que je souhaite comparer des listings ? Exclure des données de l'un par rapport à l'autre ?

Je vais devoir créer une requête imbriquée dans le WHERE.

Attention : est-ce que je peux remplacer cette requête par une jointure ?

2. Est-ce que j'ai besoin de préparer des requêtes intermédiaires afin d'effectuer des requêtes dessus (fonction d'agrégation sur une autre fonction d'agrégation ou tout autres traitements successifs) ?

Je vais devoir créer une requête imbriquée dans le FROM afin de créer une table temporaire / table dérivée

3. Est-ce que je souhaite récupérer une valeur dans mon select qui soit le résultat d'une autre requête ?

Je vais devoir créer une requête imbriquée dans le SELECT afin de créer un nouveau champ.

En général :

Règles à ne pas oublier !!!

- Toujours préciser les colonnes retourner par la recherche (le « select * » est très gourmand),
- Ne joindre que les tables nécessaires,
- Ne pas oublier le DISTINCT, souvent nécessaire avec les jointures,
- Pour augmenter la lisibilité des requêtes, renommer vos tables (alias) et colonnes,
- Documenter la requête avant de l'écrire :
  - But
  - Auteur
  - Date

Langage SQL

LES REQUÊTES ACTIONS

Requêtes « ACTION » :

- SELECT….. INTO
- INSERT INTO
- UPDATE
- CREATE
- DELETE

Requêtes « ACTION » :

INSERT : Ne renseigner que certains champs

```
use DistrisysDW

insert into DimSite
(SiteCode,Site)

VALUES
('D008','Marseille')
```

Requêtes « ACTION » :

INSERT : Renseigner tous les champs

INSERT T_Categorie VALUES (1,'Beverages','Soft drinks, coffees, teas, beers, and ales')

INSERT T_Client VALUES('ALFKI','Alfreds Futterkiste','Maria Anders','Sales Representative','Obere Str. 57','Berlin',NULL,'12209','Germany','030-0074321','030-0076545')

INSERT INTO T_Commande

VALUES (10248,N'VINET',5,'7/4/2004','8/1/2004','7/7/2004',3,32.38,

N'Vins et alcools Chevalier',N'59 rue de l''Abbaye',N'Reims', NULL,N'51100',N'France')

ET PLUSIEURS LIGNES

```
INSERT INTO T_TITRE (TIT_CODE, TIT_LIBELLE)
       VALUES  ('M.'    , 'Monsieur'),
               ('Mlle.' , 'Mademoiselle'),
               ('Mme.'  , 'Madame');
```

```
INSERT T_TITRE
       VALUES  ('M.'    , 'Monsieur',
                'Mlle.' , 'Mademoiselle'
                'Mme.'  , 'Madame')
```

Requêtes « ACTION » :

Requête INSERT INTO (sous requête)

```
CREATE TABLE T_into
(
    ClientNom varchar(255) NOT NULL,
    ClientPays nvarchar(50)
)


INSERT INTO T_into
    SELECT ClientNom,ClientPays
    FROM T_Client
    WHERE ClientPays = 'france';
```

Requêtes « ACTION » :

Requête UPDATE : Modification de valeurs

A noter, pour spécifier en une seule fois plusieurs modification, il faut séparer les attributions de valeur par des virgules. Ainsi la syntaxe deviendrait la suivante :

```
UPDATE table
SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2', colonne_3 = '
WHERE condition
```

Exemple

Imaginons une table « client » qui présente les coordonnées de clients.

Table « client » :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	18 Rue de l'Allier	Ponthion	51300	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Requêtes « ACTION » :

Requête UPDATE : Modifier une ligne

Imaginons une table « client » qui présente les coordonnées de clients.

Table « client » :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	18 Rue de l'Allier	Ponthion	51300	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Modifier une ligne

Pour modifier l'adresse du client Pierre, il est possible d'utiliser la requête SQL suivante :

```
UPDATE client
SET rue = '49 Rue Ameline',
  ville = 'Saint-Eustache-la-Forêt',
  code_postal = '76210'
WHERE id = 2
```

Cette requête sert à définir la colonne rue à « 49 Rue Ameline », la ville à « Saint-Eustache-la-Forêt » et le code postal à « 76210 » uniquement pour ligne où l'identifiant est égal à 2.

Résultats :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Requêtes « ACTION » :

Requête UPDATE : Modifier toutes les lignes

Modifier toutes les lignes

Il est possible d'effectuer une modification sur toutes les lignes en omettant d'utiliser une clause conditionnelle. Il est par exemple possible de mettre la valeur « FRANCE » dans la colonne « pays » pour toutes les lignes de la table, grâce à la requête SQL ci-dessous.

```
UPDATE client
SET pays = 'FRANCE'
```

Résultats :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	FRANCE
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	FRANCE
3	Romain	3 Chemin du Chiron	Trévérien	35190	FRANCE

Requêtes « ACTION » :

Requête UPDATE

```
UPDATE T_Produit
SET Produit_PU = Produit_PU * 1.1

Where
PRODUIT_PU < (select avg (Produit_PU) from
T_Produit
```

Requêtes « ACTION » :

Requête CREATE TABLE

```
CREATE TABLE utilisateur (
        id INT PRIMARY KEY NOT NULL,
        nom VARCHAR(100),
        prenom VARCHAR(100),
        email VARCHAR(255),
        date_naissance DATE,
        pays VARCHAR(255),
        ville VARCHAR(255),
        code_postal VARCHAR(5),
        nombre_achat INT
                                )
```

Je souhaite créer un table qui s'intitule UTILISATEUR et qui contient :
Un id de type intéger/entier dont on n'autorise pas les null,
Un Nom de type Varchar (charactère variable) de taille maximum = 100

Un date de naissance de type DATE car on ne souhaite pas connaître l'heure, les minutes, les secondes, etc.

Requêtes « ACTION » :

Requête CREATE TABLE

```
CREATE TABLE artists
        (  idartiste    INTEGER PRIMARY KEY NOT NULL,
           name  TEXT)

CREATE TABLE tracks
        (  traid     INTEGER PRIMARY KEY NOT NULL,
           title   TEXT,
           ida INTEGER,
          FOREIGN KEY(ida) REFERENCES artists(idartiste)  )
```

Je crée une première table « Artists » avec deux colonnes, un ID PK et un nom.

Je crée ensuite une seconde table « Tracks » avec un ID PK, un title, un Ida de type entier qui fait référence à une clé étrangère relative au champs PK Idartiste de la table artiste.

Requêtes « ACTION » :

Requête DELETE / Drop

Delete from T_Client

- La plus définitive
- On supprime les lignes d'une table
- On préfère « flagger » un client (non actif, par exemple, associé à une date de modification) plutôt que le supprimer de la base de données

- DROP : on supprime la table

ANNEXES

Sommaire / Plan détaillé du support de formation

Schémas des bases de données

Liens web utiles

Sommaire / Plan détaillé (1/3)

Sommaire / Plan détaillé (2/3)

Sommaire / Plan détaillé (3/3)

La commande SELECT :

L'ordre de sa syntaxe (1/2)

1.SELECT sum(ventes) as 'Montant des ventes' → AFFICHAGE / RESULTAT
, year(cdedate) as 'Année, service (Virgules)

2.FROM table1 JOIN table2 → PROVENANCE / DANS QUELLE TABLE ?
(JOIN/ON et/ou virgules)

On table1.PK=table2.FK
JOIN table3
On table2.Pk=Table3.FK

3.WHERE pays = 'France' and Age= 20 → CONDITION ? Où ? FILTRE SUR CHAMP DANS LA BASE DE DONNEES
(AND et/ou OR)

La commande SELECT :

L'ordre de sa syntaxe (2/2)

4.GROUP BY year(cdedate), service → REGROUPER PAR (tout ce qui est ici est à copier-coller dans le Select) (Virgules)

5.HAVING sum(ventes) > 50000 → CONDITION ? Où ? FILTRE SUR une fonction d'agrégation dont le résultat est visible après avoir fait le GROUP BY (AND et/ou OR)

6.ORDER By CHAMP1 asc/desc → TRIER PAR (Virgules)

La commande SELECT :

Quand faut-il mettre des parenthèses ?

Les parenthèses sont **obligatoires dans 3 cas** :

- Lors de l'utilisation des **fonctions intégrées** (en rose dans SQL Server) : Year(champ), count(champ), convert(champ), etc.

- Lors de l'utilisation du **IN / NOT IN** → signifie que l'on souhaite mettre une condition correspondant au listing indiqué à l'intérieur :

Where année_naissance IN (1985, 1988) = *année est égale à 1985 ou 1988*
Where ville NOT IN ('Lyon','Lille') = ville n'est pas Lyon ou Lille

- Lors de l'utilisation de **OR et AND cumulatifs** dans une requête :

Where age = 50 and **(**ville like '%p' or ville like '%f'**)** → afin qu'il ne se mélange pas les pinceaux entre AND et OR

- Lors de l'utilisation **d'une sous requête** : une requête imbriquée dans une autre requête sera toujours entourée de parenthèses.

La commande SELECT :

Quand faut-il mettre des cotes ?

Les cotes sont **obligatoires dans 2 cas** :

- **Lors de la création d'un alias de colonne,** si ce dernier contient plusieurs mots :

✓ Select ZERTIKL as 'Nom du client' → si j'ai des espaces, alors je dois mettre des cotes.

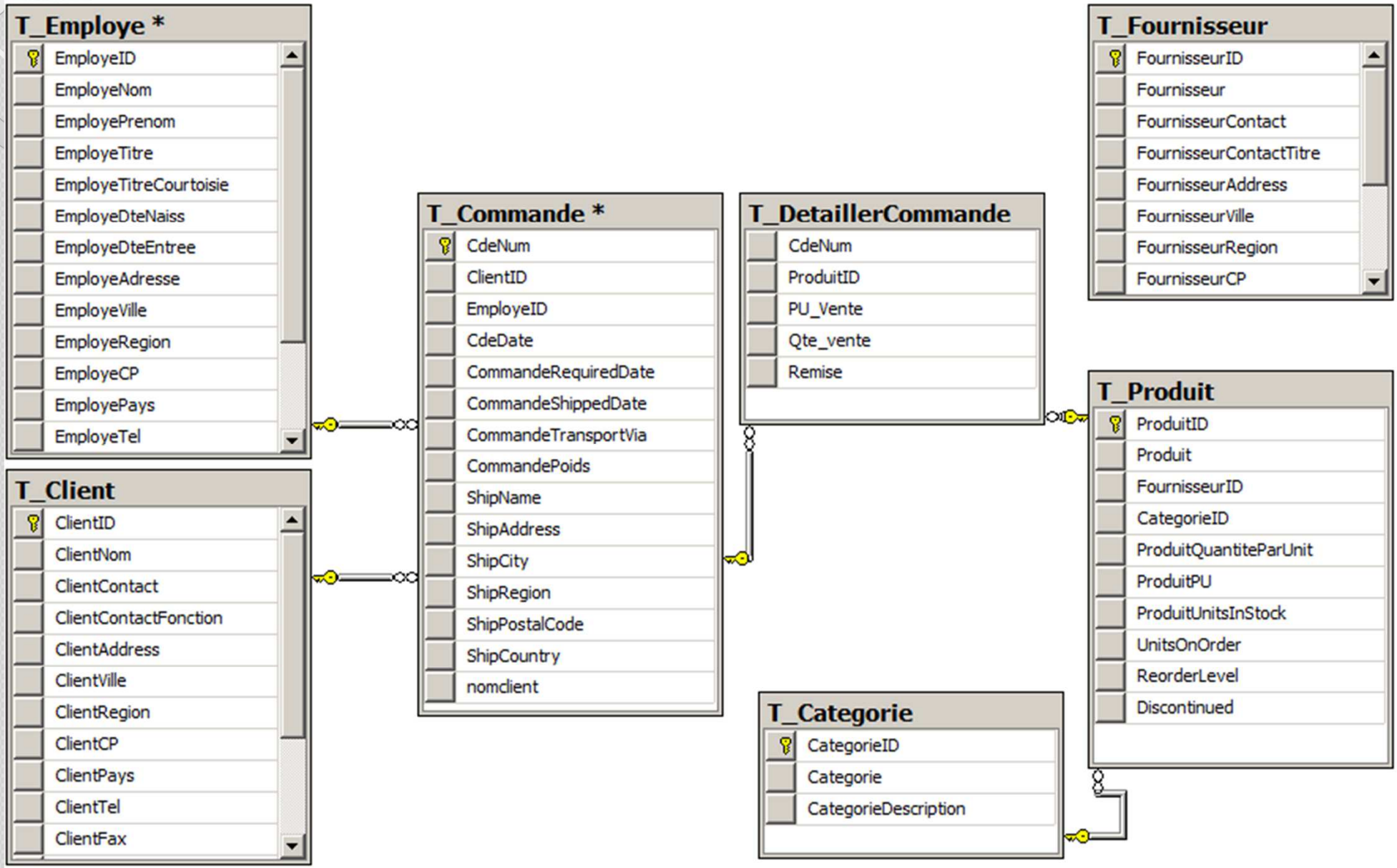
✓ Select ZERTIKL as 'Nom' → s'il n'y a pas d'espaces, les cotes ne sont pas obligatoires mais peuvent être utilisées.

- **Lors de la création d'une condition sur une valeur,** si le type de donnée de cette dernière n'est pas du numérique :

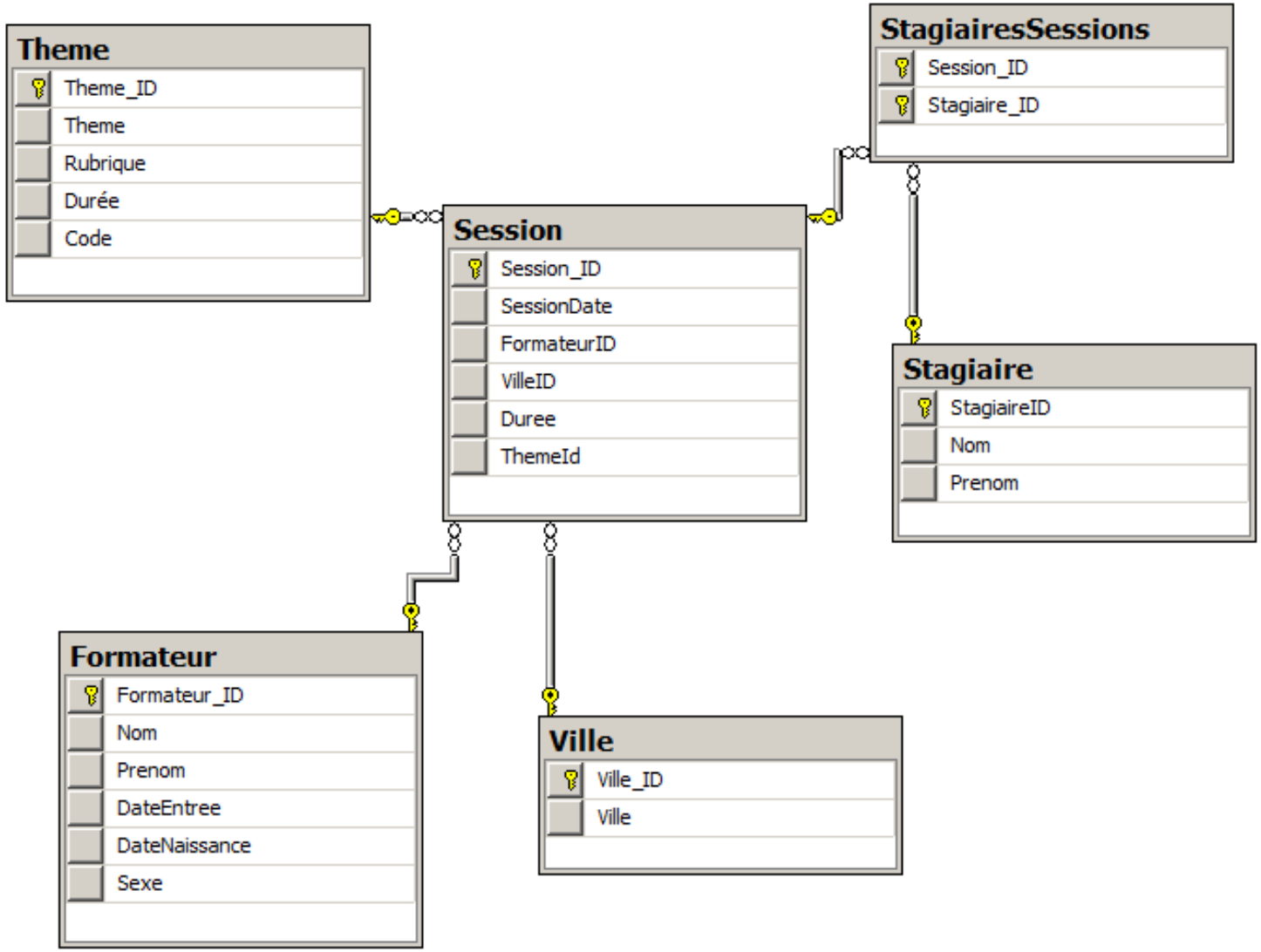
✓ Where ZERTIKL = 'DUPONT'

✓ Where Age = '50' → la valeur est du numérique, les cotes ne sont pas obligatoires mais peuvent être utilisées.

Schémas des bases de données - Comptoir

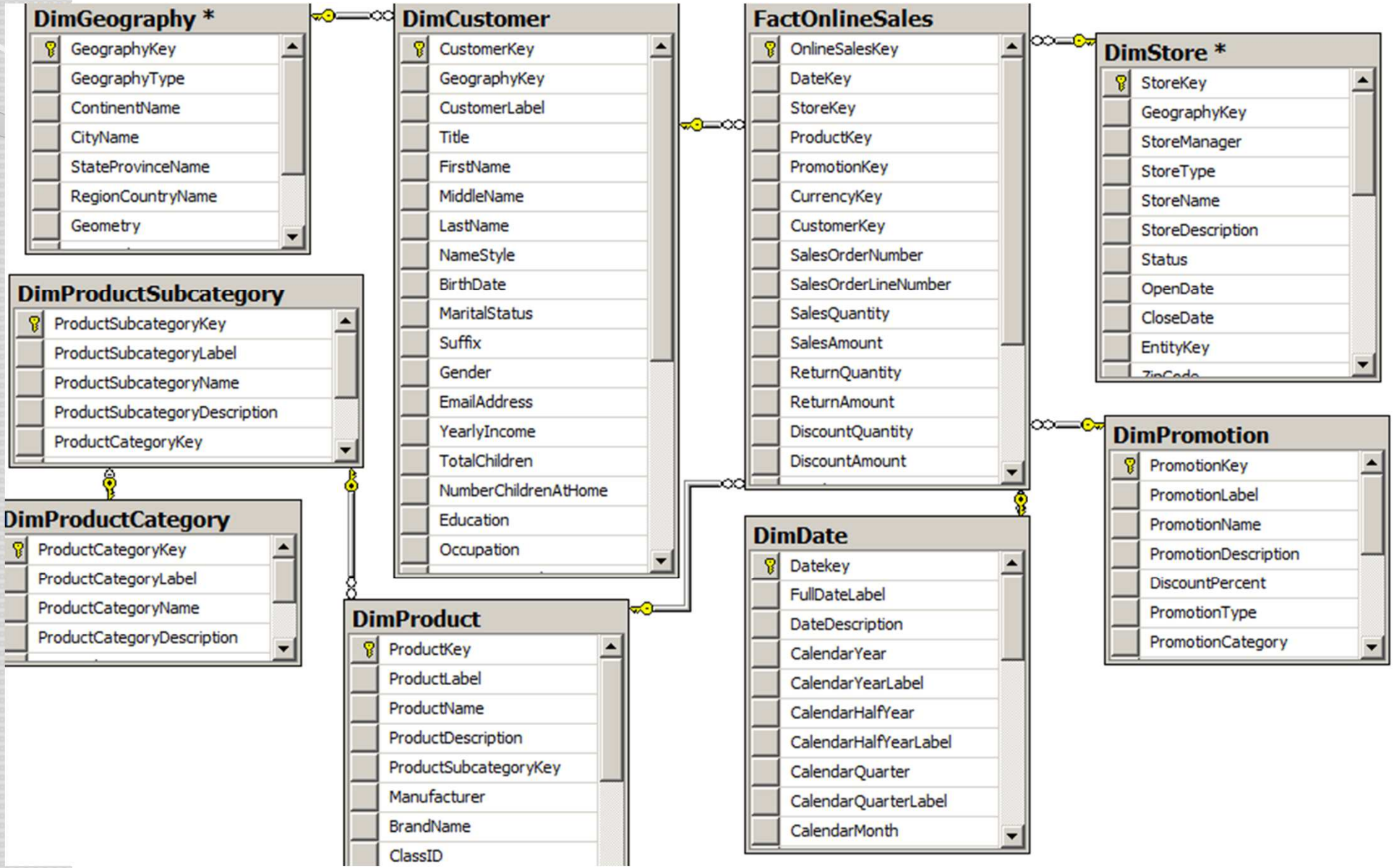


Schémas des bases de données _SQL_SESSION_SIF



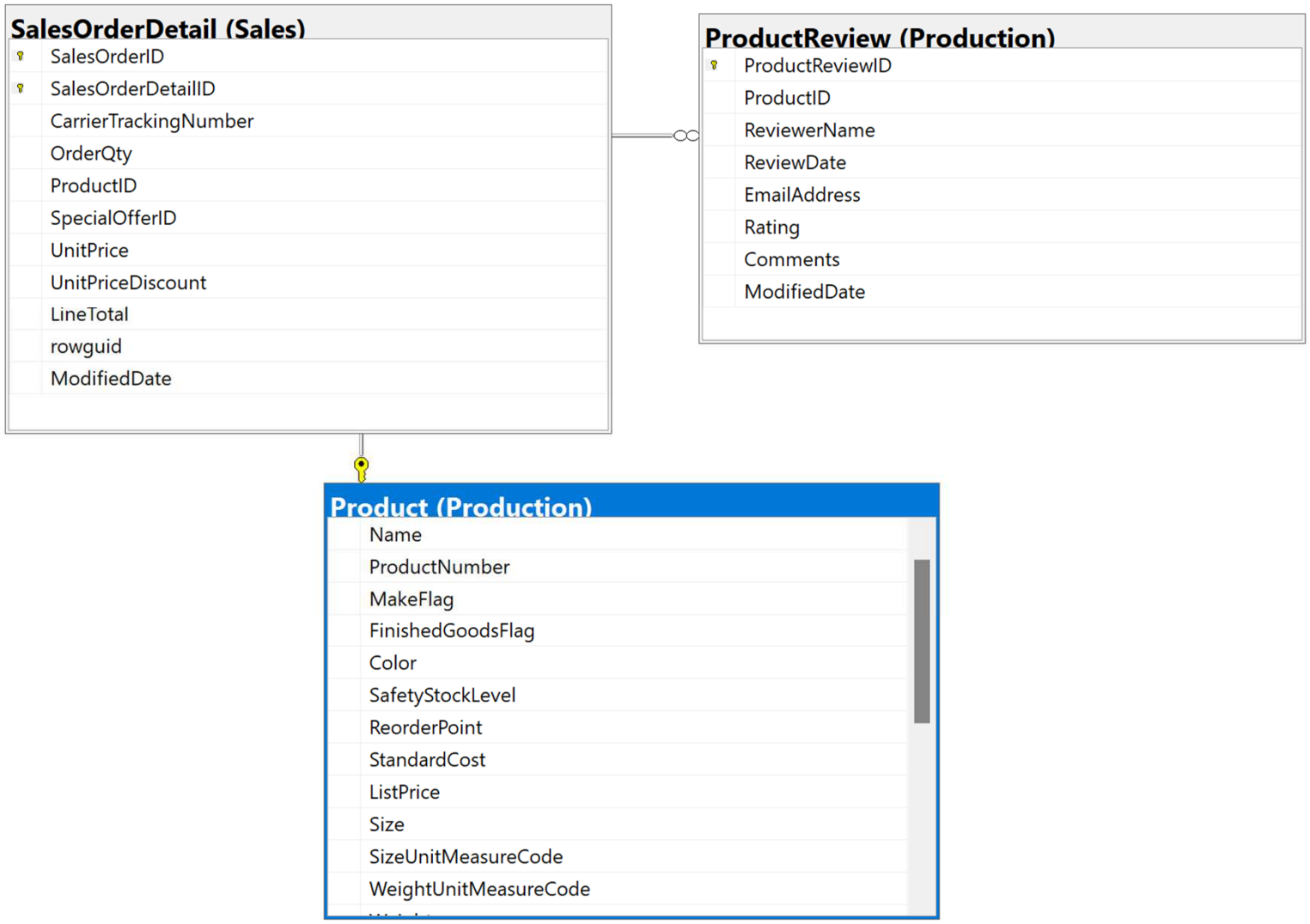
Schémas des bases de données

ConsotoRetailDW (simplifié)



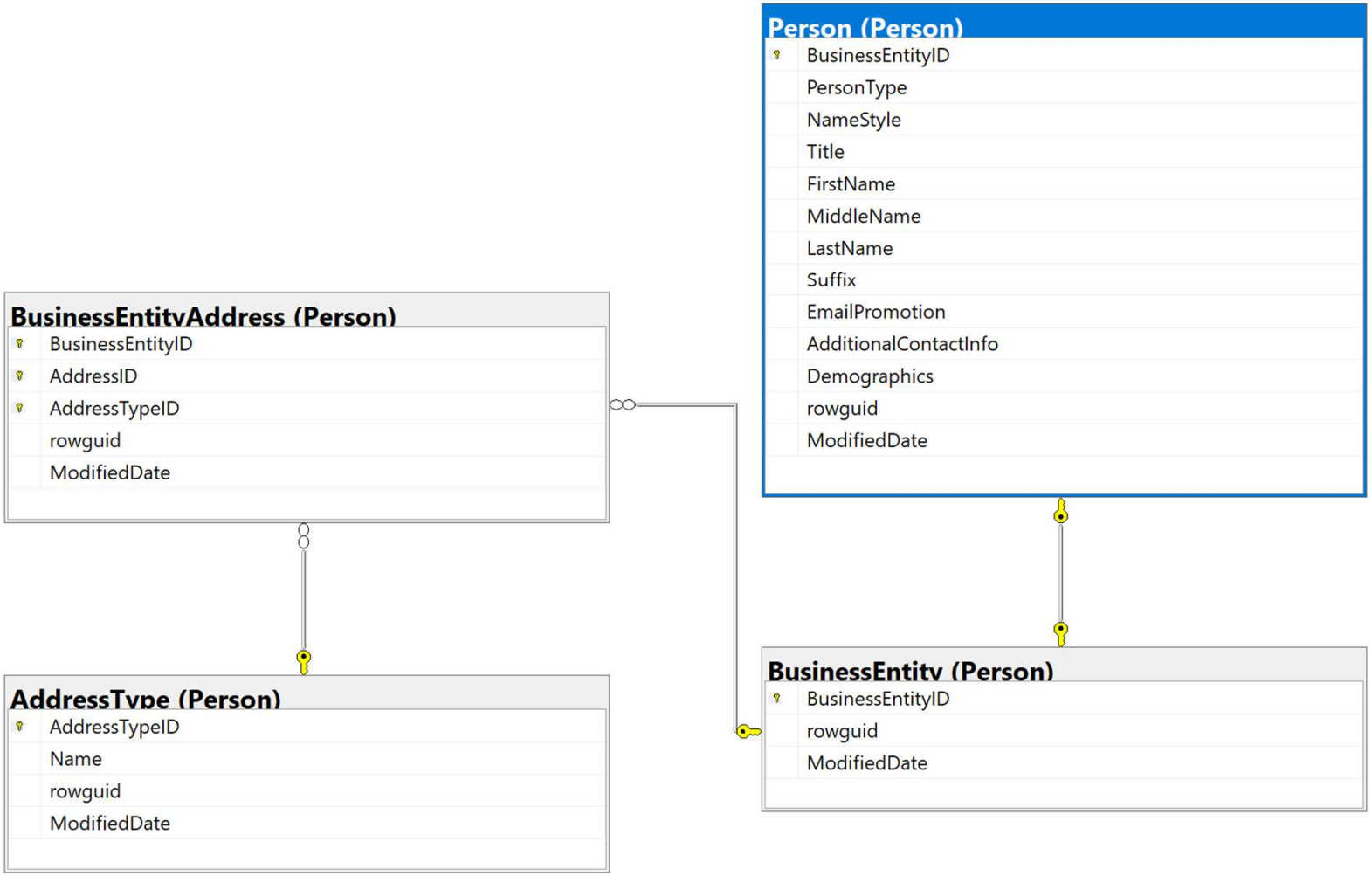
Schémas des bases de données

AdventureWorks2012 (Exo 5 et exos 7 à 10 du doc 8)



Schémas des bases de données

AdventureWorks2012 (Exo 6 du doc 8)



Liens web utiles

Le Web regorge de sites traitant (parfois de façon excellente) du SQL.

Comme il n'est pas possible de les citer tous, en voici une liste tout à fait arbitraire mais représentative de ce qu'on trouve sur le web :

- http://sql.sh/
- https://openclassrooms.com/courses/apprenez-a-programmer-en-vb-net/introduction-au-langage-sql
- http://sql.dev
- https://stackoverflow.com/
- https://technet.microsoft.com/fr-fr/library/ms190750(v=sql.105).aspxeloppez.com/#apprendre-sql

MERCI POUR VOTRE ATTENTION

J'espère que vous repartez confiant et serein autour du langage SQL

Bonne continuation