Langage SQL

Sparks

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/

- berenguer.laure@laposte.net
- 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

Introduction bases de données

A la fin de cette session, vous serez capable de :

- Comprendre l'architecture d'une base de données relationnelle
- Reconnaître les tables
- Reconnaître les champs

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

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).

La table est une forme simple et parlante pour rassembler des données ou représenter des informations. La forme tabulaire nous étant familière, il est aisé d'interpréter sa structure au premier coup d'œil.

No de série	Fabricant	Modèle	Année	Immatriculation
YG100P9065QZ84	Ford	Taurus	2005	WWP 657
JK92876T6753W9	Nissan	Pathfinder XE	2004	KDF 324
PK8750927GH786	BMW	320 SI	2002	BGH 629

Eléments constitutifs

Requêtes mono-tables

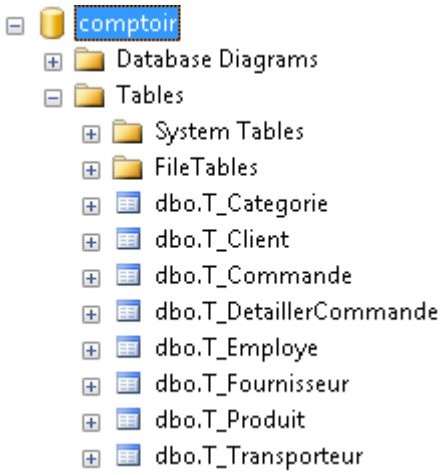
- Les tables
- Les champs (colonnes) et les types de données
- Les enregistrements (lignes)

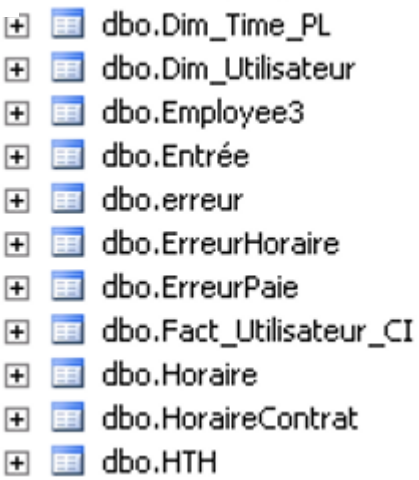
Requêtes multi-tables :

- Le modèle Physique de données (simplifié) **indispensable**
- Les clés
- Les relations

Base de données : Structure

Les tables

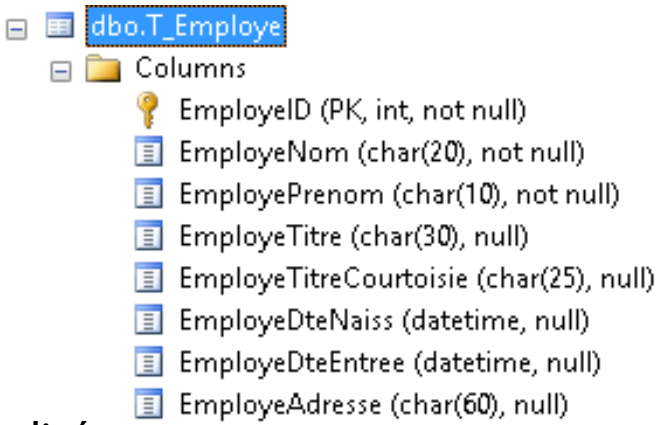




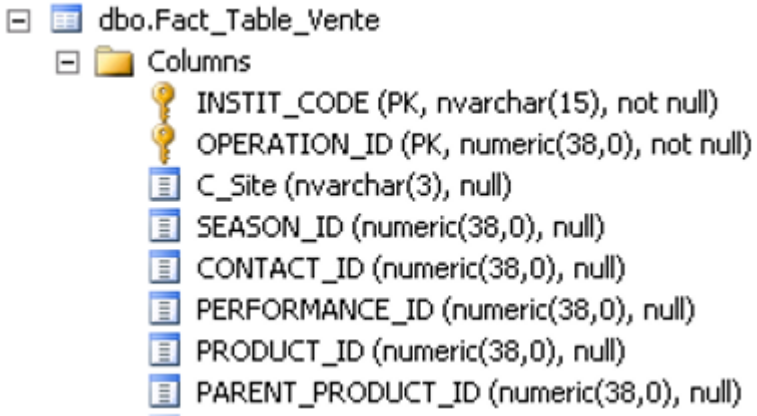
Table_Name
DWH_TNAC_CONTROL_HISTORY
DWH_TICKET_CHECK
DWH_SALE
DWH_V_SALE
DWH_PAYMENTS
ORDER_TABLE
DWH_CONTACT
ITEM

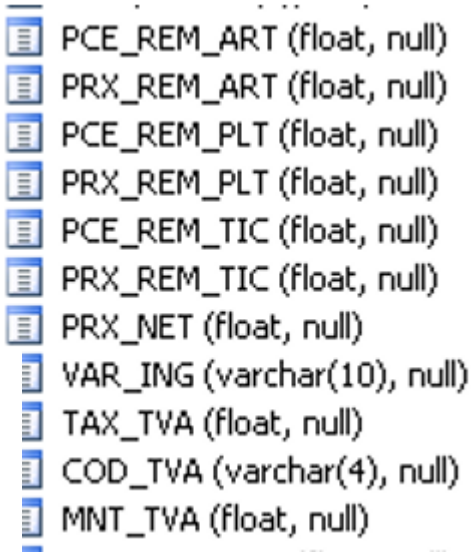
Base de données : Structure

Les champs



Champs normalisés





Champs non normalisés

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

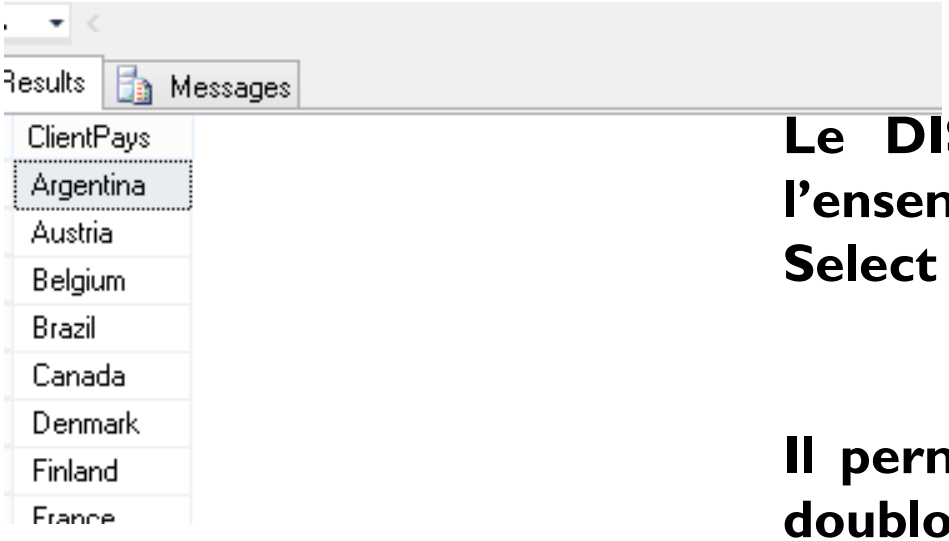
Retourne et affiche l'ensemble des champs (colonnes) et des lignes. (A éviter)

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
ORDER BY ClientPays
```



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

Il permet de supprimer les doublons sur les lignes.

La commande SELECT :

Exemples SQL server

```
SELECT top 10 *
from
DimCustomer
```

```
SELECT
FirstName,
LastName,
Education
from
DimCustomer
```

```
use ContosoRetailDW

select
firstname,
Lastname,
Education

from dimcustomer
where
education = 'Bachelors'
```

100 %

Results | Messages

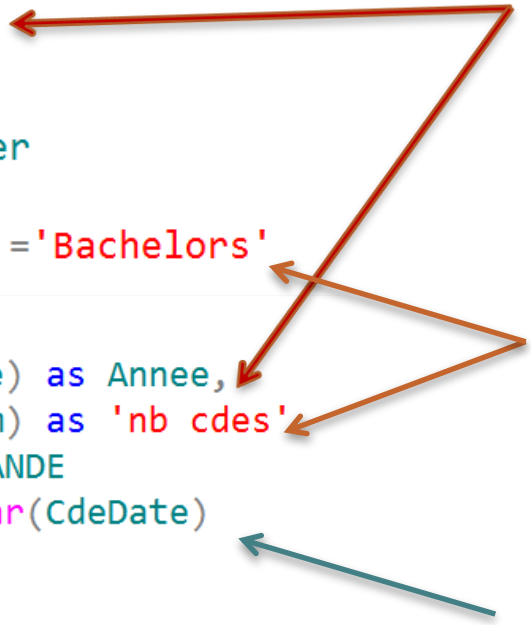
	firstname	Lastname	Education
1	Jon	Yang	Bachelors
2	Eugene	Huang	Bachelors
3	Ruben	Torres	Bachelors
4	Christy	Zhu	Bachelors
5	Elizabeth	Johnson	Bachelors

La commande SELECT :

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

```
SELECT
FirstName,
LastName,
Education
from
DimCustomer
where
education ='Bachelors'
```

```
SELECT
Year(CdeDate) as Annee,
COUNT(CdeNum) as 'nb cdes'
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 :

Exemples Oracle

```
SELECT home_id, home_type, bathrooms
FROM homes
WHERE home_id < 500
AND home_type = 'two-storey'
ORDER BY home_type ASC, bathrooms DESC;
```

```
SELECT *
FROM homes
WHERE bathrooms >= 2
ORDER BY home_type ASC;
```

```
SELECT *
FROM contacts
WHERE last_name = 'Smith'
AND contact_id >= 1000
AND contact_id <= 2000;
```

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 :

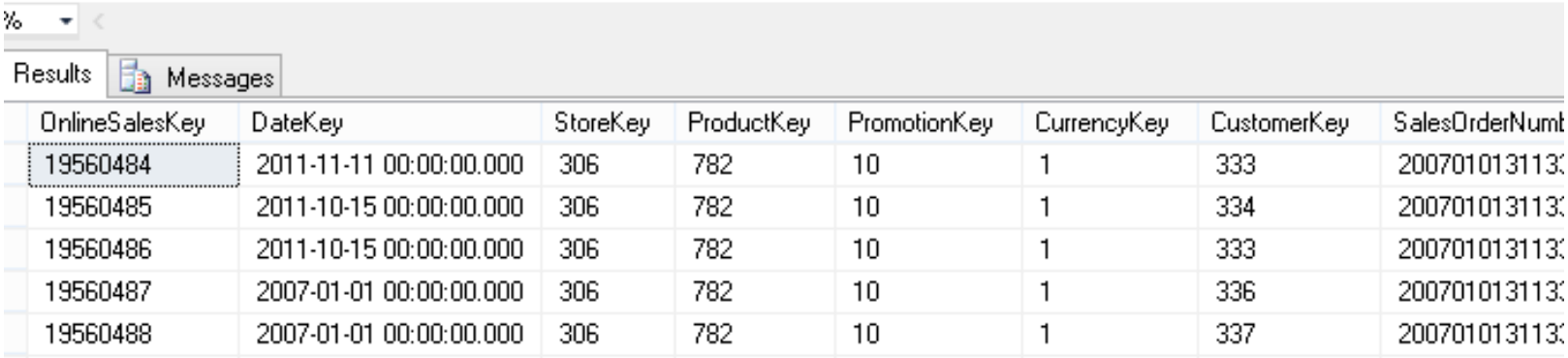
Règles d'écritures : (erreurs surlignées en rouge)
Où sont les erreurs ?

Ouvrir la requête : Ou_sont_erreurs_code.sql

La commande SELECT :

SQL Server et la clause TOP (voir diapo suivante pour Oracle et MySQL)

```
SELECT TOP 1000 *
  FROM [ContosoRetailDW].[dbo].[FactOnlineSales]
```



Results | Messages

OnlineSalesKey	DateKey	StoreKey	ProductKey	PromotionKey	CurrencyKey	CustomerKey	SalesOrderNumb
19560484	2011-11-11 00:00:00.000	306	782	10	1	333	200701013113
19560485	2011-10-15 00:00:00.000	306	782	10	1	334	200701013113
19560486	2011-10-15 00:00:00.000	306	782	10	1	333	200701013113
19560487	2007-01-01 00:00:00.000	306	782	10	1	336	200701013113
19560488	2007-01-01 00:00:00.000	306	782	10	1	337	200701013113

La commande SELECT :

Oracle et la clause RowNum
MySQL et la clause LIMIT

Oracle Syntax

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

MySQL Syntax

```
SELECT column_name(s)
FROM table_name
LIMIT number;
```

Example

```
SELECT *
FROM Persons
WHERE ROWNUM <=5;
```

Example

```
SELECT *
FROM Persons
LIMIT 5;
```

La commande SELECT :

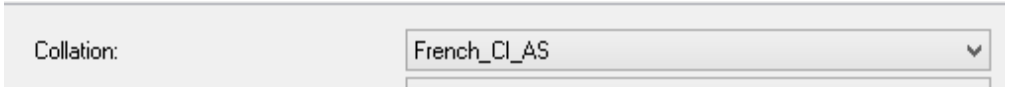
Sensibilité à la casse (majuscule / minuscule)

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.
Celle-ci : LA PLUS UTILISEE
*/
 SELECT
        CLIST AS 'Nom Client' ,
        CLITT AS 'Tel Client'
   FROM Client

/*
Ou encore celle là :
*/


SELECT 'Nom Client' = CLIST
      ,'Tel Client' = CLITT
   FROM Client
```

La commande SELECT :
La clause ORDER BY

```
select
ClientNom,
ClientRegion,
ClientPays

from T_Client
order by ClientPays, ClientVille desc
```

ordre **croissant**

Ordre **décroissant**

Attention, c'est que dans SQL SERVER que le ascendant est par défaut. 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')
```

Je me positionne à droite 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

```
use ContosoRetailDW
SELECT * from
DimProduct
WHERE ProductName LIKE 'BK%';

use comptoir_OLTP
SELECT
ClientNom,
ClientPays
FROM T_CLIENT
WHERE
ClientPays not like 'a%'
/*
qui contient a '%a%'
*/
```

Like = qui intègre les caractères

A% qui commence par a

%a qui termine par a

%a% qui contient a

% signifie qu'importe ce qui a avant et/ou qu'importe ce qui a après

La commande SELECT :

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

Cette commande avec AND à la place de between prendra trois fois plus de temps pour arriver au même résultat

```
USE AdventureWorks2012

SELECT SalesOrderID, OrderDate, CustomerID
FROM sales.SalesOrderHeader
WHERE OrderDate BETWEEN '20080211' AND '20080212'

SELECT SalesOrderID, OrderDate, CustomerID
FROM sales.SalesOrderHeader
WHERE OrderDate >= '20080211' AND OrderDate <'20080213'
```

Résultats | Messages

	SalesOrderID	OrderDate	CustomerID
1	63947	2008-02-11 00:00:00.000	21248
2	63948	2008-02-11 00:00:00.000	16705
3	63949	2008-02-11 00:00:00.000	27145
4	63950	2008-02-11 00:00:00.000	27592
5	63951	2008-02-11 00:00:00.000	26136
6	63952	2008-02-11 00:00:00.000	17076

	SalesOrderID	OrderDate	CustomerID
1	63947	2008-02-11 00:00:00.000	21248
2	63948	2008-02-11 00:00:00.000	16705
3	63949	2008-02-11 00:00:00.000	27145
4	63950	2008-02-11 00:00:00.000	27592

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 :

La clause CASE

```
select cdenum,
sum(qte_vente) as 'qté vente',

case
    when sum(qte_vente) < 50 then 'Faible'
    when sum(qte_vente) > 150 then 'Elevé'
    else 'Modéré'
    end as 'quantité vente'

from T_DetaillerCommande
group by CdeNum
```

200 %

Résultats | Messages

	cdenum	qté vente	quantité vente
1	10411	74	Modéré
2	10743	28	Faible
3	11075	42	Faible
4	10388	75	Modéré
5	10720	29	Faible

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…

La commande SELECT :

La clause CASE (exemples dates)

```
Use AdventureWorks2012

SELECT
  CASE
    WHEN StartDate IS NULL THEN 'Unknown'
    WHEN DATEDIFF(YY,StartDate, GETDATE()) BETWEEN 0 AND 10 THEN 'Recent'
    WHEN DATEDIFF(YY,StartDate, GETDATE()) BETWEEN 10 AND 20 THEN 'Old'
    ELSE 'Ancient'
  END AS Recency, count(*)
  FROM Production.WorkOrder

  group by
  CASE
    WHEN StartDate IS NULL THEN 'Unknown'
    WHEN DATEDIFF(YY,StartDate, GETDATE()) BETWEEN 0 AND 10 THEN 'Recent'
    WHEN DATEDIFF(YY,StartDate, GETDATE()) BETWEEN 10 AND 20 THEN 'Old'
    ELSE 'Ancient'
  END
```

La commande SELECT :
La clause CASE (Oracle)

```
SELECT supplier_id,
CASE
  WHEN supplier_name = 'IBM' and supplier_type = 'Hardware' THEN 'North office'
  WHEN supplier_name = 'IBM' and supplier_type = 'Software' THEN 'South office'
END
FROM suppliers;
```

Nous pouvons mettre autant de conditions, sur plusieurs colonnes différentes pour une seule retraduction.

Quand le supplier_name est égal à IBM ET que le type de supplier est égal à HardWare alors je crée la valeur North Office dans ma nouvelle colonne.

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

```
SELECT top 10
salesamount as 'vente',
discountamount as 'remise',
salesamount - discountamount as 'hors remise'
from
factonlinesales
```

Results | Messages

vente	remise	hors remise
12,5356	2,59	9,9456
12,5356	2,59	9,9456
12,5356	2,59	9,9456
12,5356	2,59	9,9456
12,5356	2,59	9,9456
4,8206	0,996	3,8246
4,8206	0,996	3,8246
4,8206	0,996	3,8246
4,8206	0,996	3,8246
4,8206	0,996	3,8246

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(*)
- SUM (<champs>)

```
/*nombre de commandes passées en 2004*/
select count(*)
from T_commande
where year(CdeDate)=2004
```

```
/*CA global (toutes années et toutes cdes confondues)*/
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)
from Dimproduct
where ProductKey between 2 and 1216
group by ColorName
```

Le critère where est **avant** le regroupement car sur un CHAMP

```
select
ColorName,
count(ProductKey)
from Dimproduct
where ProductKey between 2 and 1216
group by ColorName
having count(ProductKey) > 50
```

Le critère having est **après** le regroupement car sur une fonction d'agrégation

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 et heure du jour',
YEAR(cdedate) as 'Année',
MONTH(cdedate) as 'Mois',
DATEPART("weekday",cdedate) as 'jour de semaine',
DATEPART("quarter",cdedate) as 'trimestre',
DATEDIFF(day,cdedate, getdate()) as 'délai en jours',
DATEDIFF(Month,cdedate, getdate()) as 'délai en mois',
DATEDIFF(YEAR,cdedate, getdate()) as 'délai en année'
FROM T_Commande
```

getdate() = sélectionner date du jour
Attention si serveur en cloud avec une location et donc une date potentiellement différente

Résultats | Messages

	CdeNum	CdeDate	date et heure du jour	Année	Mois	jour de semaine	trimestre	délai en jours	délai en mois	délai en année
1	10249	2004-07-05 00:00:00.000	2019-12-16 17:22:22.510	2004	7	1	3	5642	185	15
2	10250	2004-07-08 00:00:00.000	2019-12-16 17:22:22.510	2004	7	4	3	5639	185	15
3	10251	2004-07-08 00:00:00.000	2019-12-16 17:22:22.510	2004	7	4	3	5639	185	15
4	10252	2004-07-09 00:00:00.000	2019-12-16 17:22:22.510	2004	7	5	3	5638	185	15
5	10253	2004-07-10 00:00:00.000	2019-12-16 17:22:22.510	2004	7	6	3	5637	185	15

Les calculs / fonctions intégrées

Les fonctions : Date (SQL Server) : Exemples

```
-- Extraire les numeros de bobine de l'heure
-- passée--

select top 100
Laminate_Id,
Start_Time

from
[dbo].[Coater_Speed]

where
DATEDIFF(minute, Start_Time,getdate())<'60'
```

Les calculs / fonctions intégrées

La clause WHERE : Les critères avec fonctions

datediff = différence entre 2 dates

Il faut préciser 3 éléments :

L'intervalle (année, mois, minutes, etc.)

La date de début

La date de fin

```
select CdeNum,
datediff(day,CdeDate,commandeshippeddate) as delai
from t_commande
where year (CdeDate)= 2004
-----------------------------------------------
select employenom
from T_Employe
where (datediff(year,employeDteEntree,getdate()))>15
```

Results | Messages

CdeNum	delai
10249	5
10250	4
10251	7
10252	2
10253	6
10254	12
10255	3
10256	2

employenom
Davolio
Fuller
Leverling

Les calculs / fonctions intégrées

Chaînes de caractères (SQL Server)

```
use Comptoir_OLTP
Select
LEFT(ClientPays,3) as '3ères lettres G',
SUBSTRING(ClientPays,3,2) as ' extrait du texte',
UPPER (ClientPays) as 'en majuscules',
LOWER(ClientPays) as 'en minuscules',
LEN(ClientPays) as 'Nb caractères',
CHARINDEX('R',ClientPays,1) as 'position du "R" dans la chaine',
REPLACE (ClientPays,'M','m') as 'remplace'

from
T_Client
```

Results | Messages

3ères lettres G	extrait du texte	en majuscules	en minuscules	Nb caractères	position du "R" dans la chaine	remplace
fra	an	FRANCE	france	6	2	france
Esp	pa	ESPAGNE	espagne	7	0	Espagne
Mex	xi	MEXICO	mexico	6	0	mexico
Mex	xi	MEXICO	mexico	6	0	mexico

Les calculs / fonctions intégrées

Convertir : La fonction CAST

```
select getdate() as 'date entiere',

    cast (getdate() as date) as 'date cast',
    convert (date, getdate()) as 'date convert',

    convert(varchar(20),getdate(),108) 'date format 108',
    convert(varchar(20),getdate(),103) 'date format 103',
    convert(varchar(20),getdate(),107) 'date format 107',
    convert(varchar(20),getdate(),113) 'date format 113',
    convert(varchar(20),getdate(),126) 'date format 126'
```

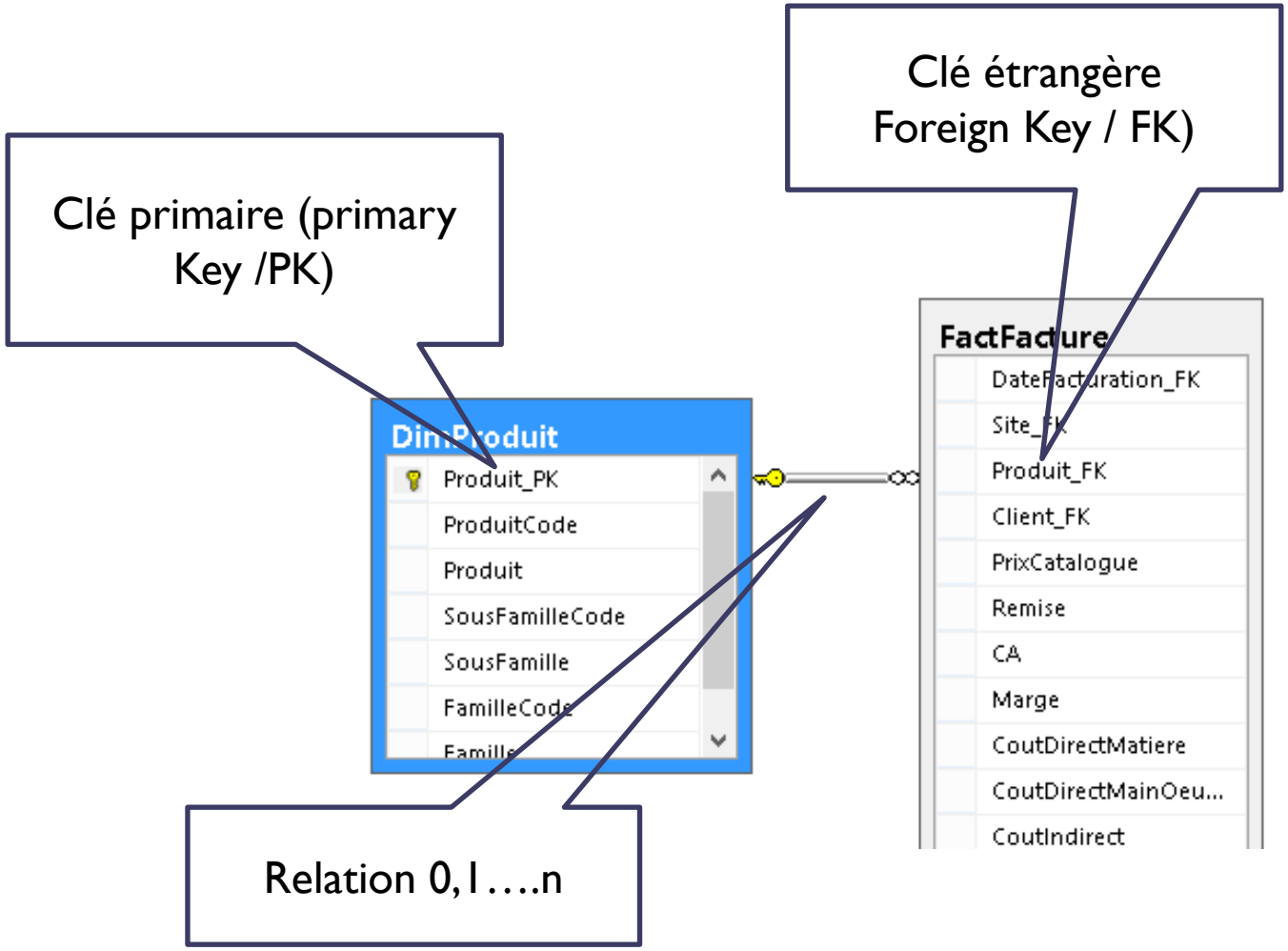
date entiere	date cast	date convert	date format 108	date format 103	date format 107	date format 113	date format 126
2020-01-20 16:29:21.447	2020-01-20	2020-01-20	16:29:21	20/01/2020	janv 20, 2020	20 janv 2020 16:29:2	2020-01-20T16:29:21.

Langage SQL

LES JOINTURES

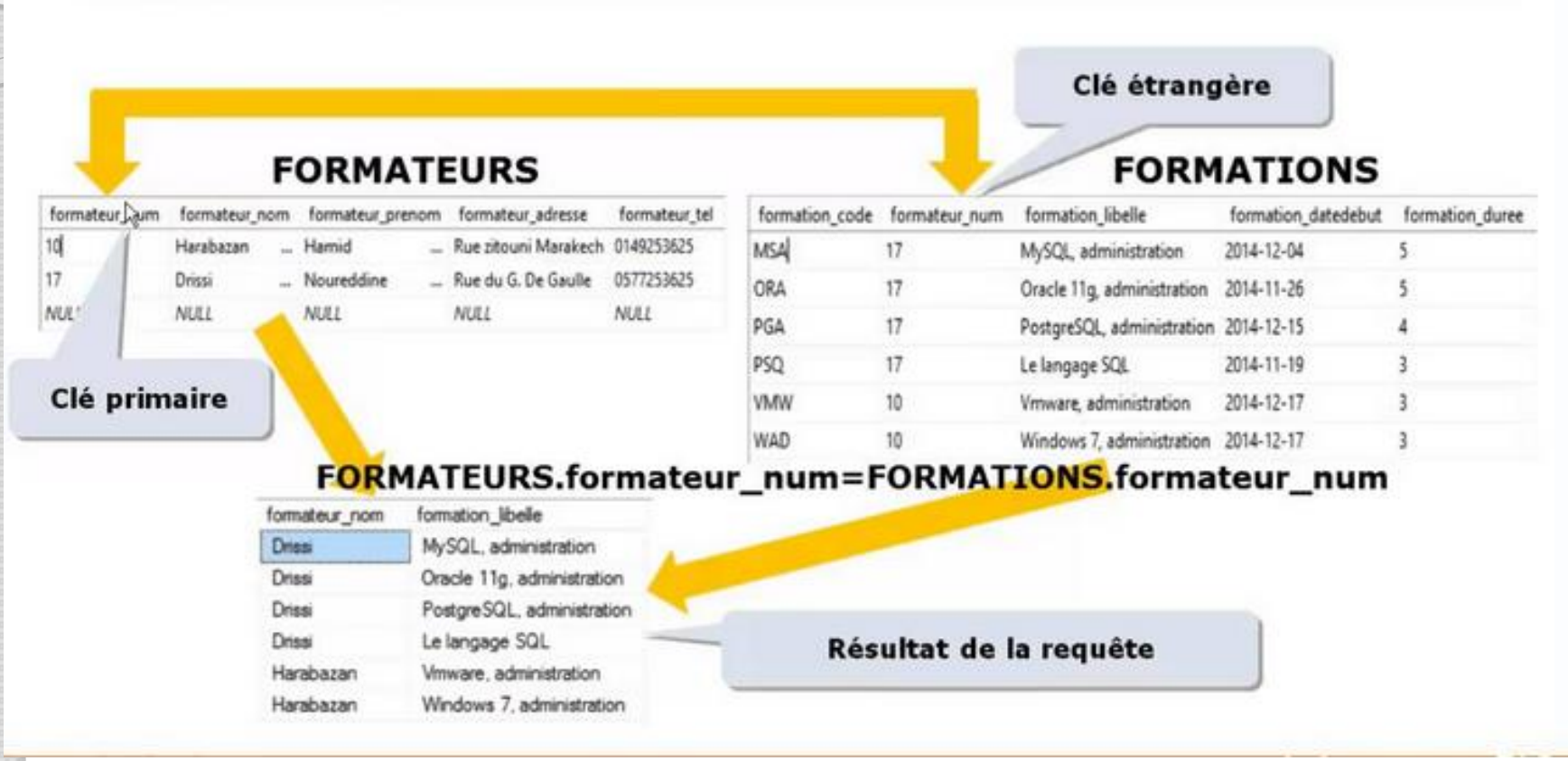
Les jointures

Structure d'une table : Les clés



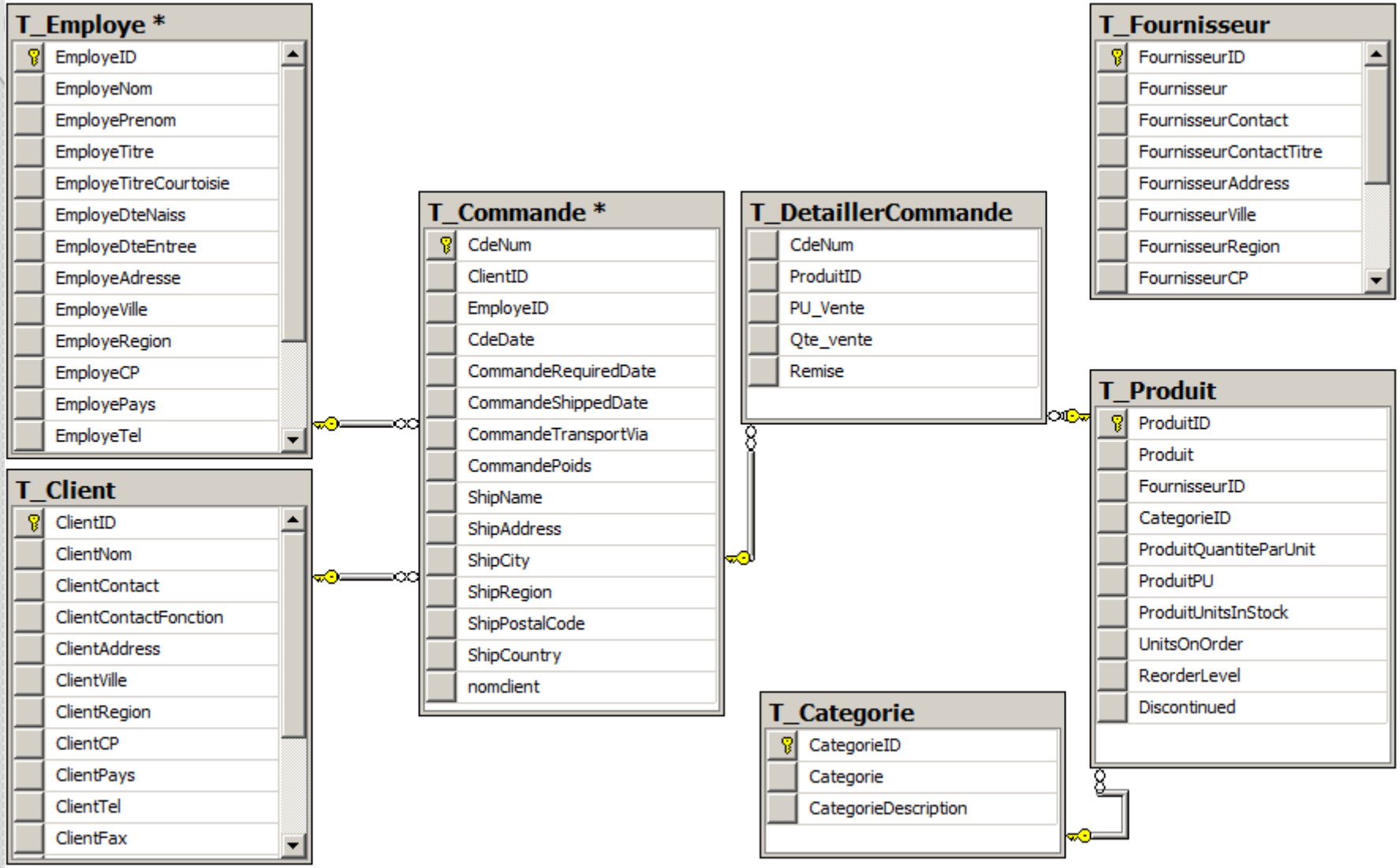
Les jointures

Rôle des clés



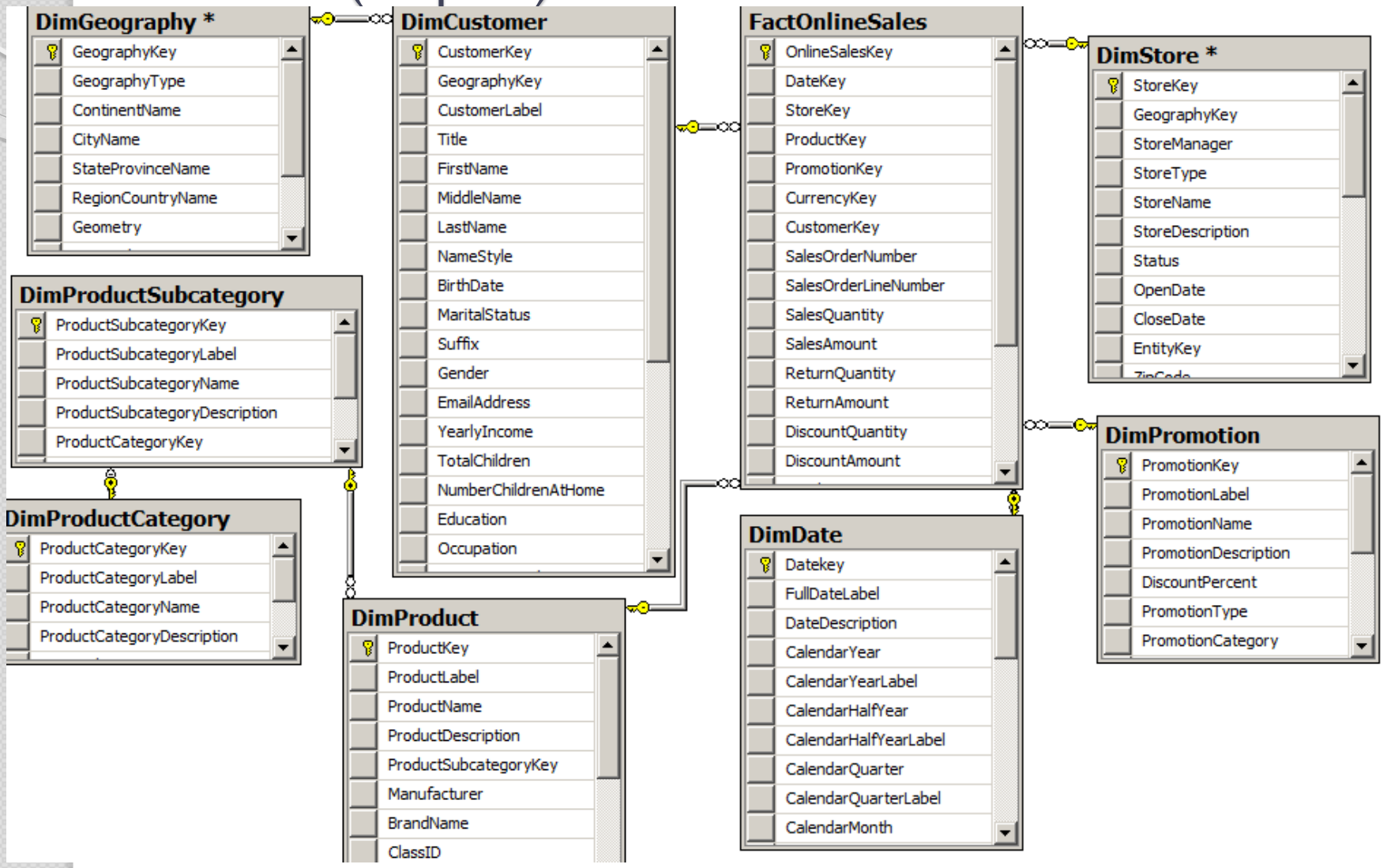
Les jointures

Lire un modèle relationnel (ou physique / MPD)



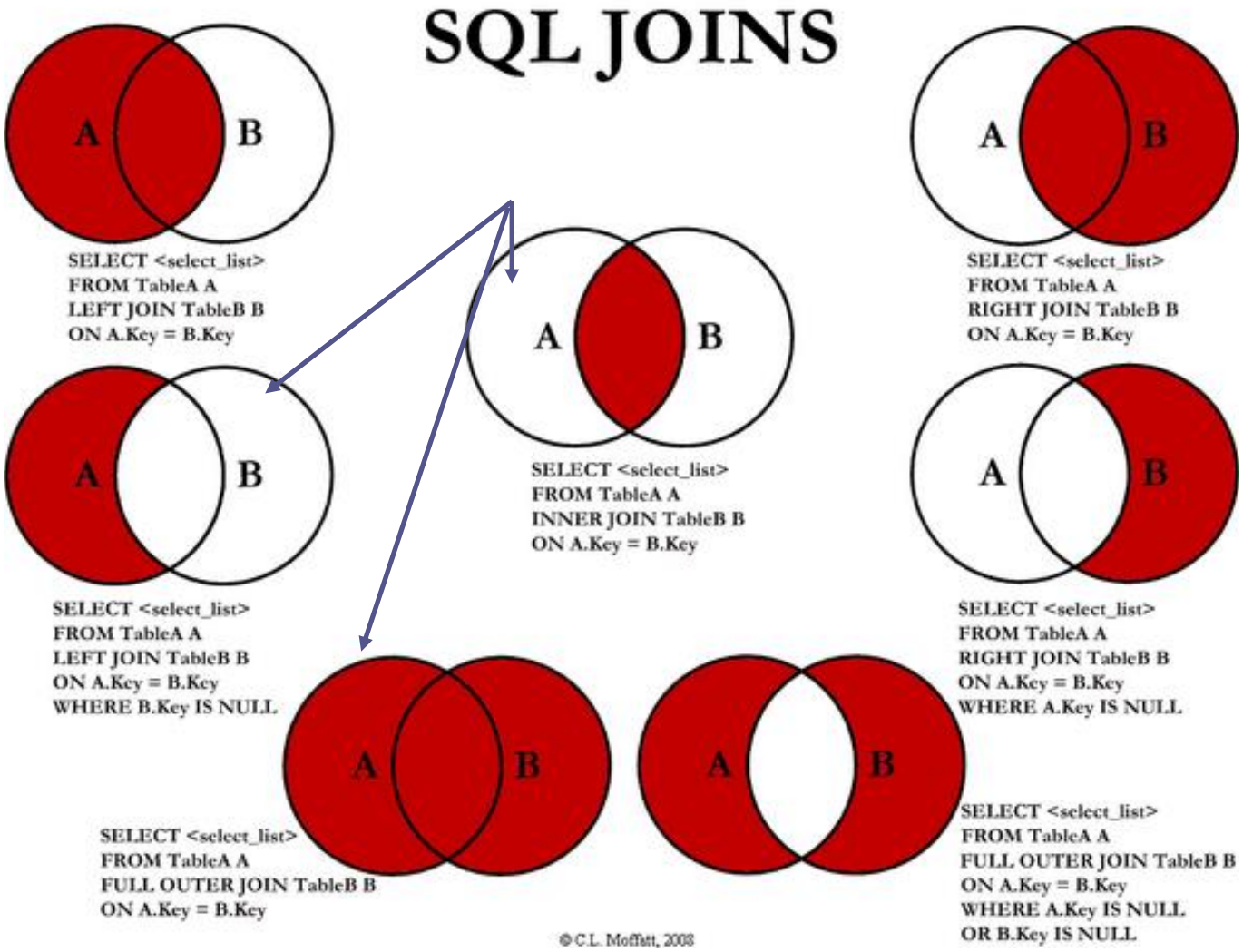
Les jointures

Le MPD (simplifié) de la base ContosoRetailDW



Les jointures

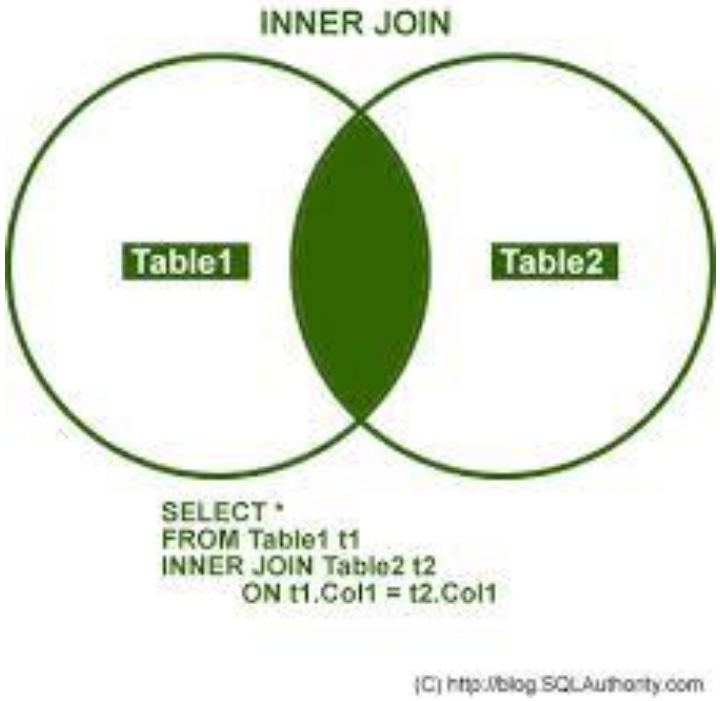
Les jointures Vue d'ensemble



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

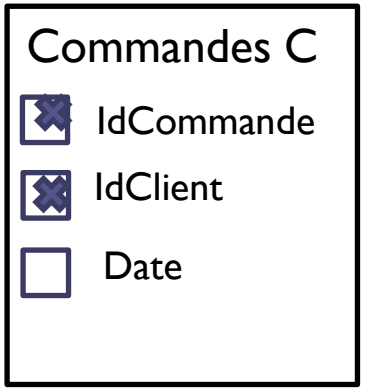
Les jointures

Renommer les tables (alias)

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

```
Select DAT.CalendarYear,
       STO.StoreName,
       FAC.SalesAmount
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'
```



Les jointures

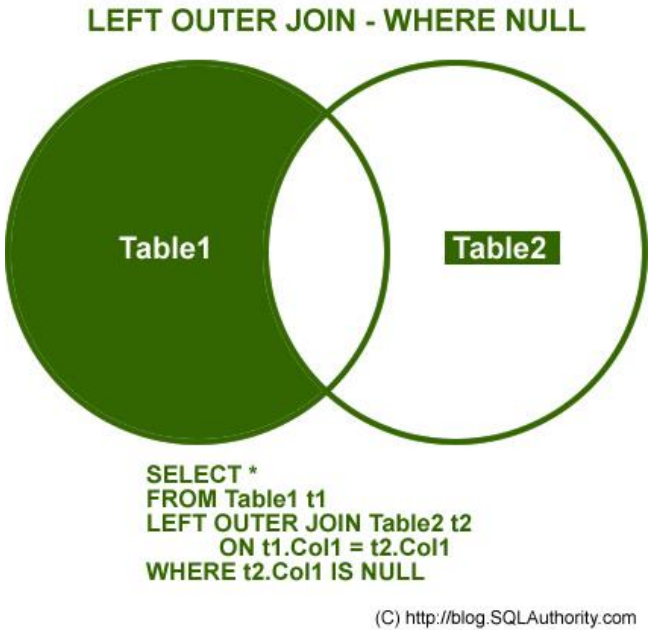
Renommer les tables (alias « métier »)

```
SELECT
a.NUM_0,
a.ACCDAT_0,
a.AMTNOT_0,
b.SDHNUM_0,
b.SOHNUM_0,
b.SDDLIN_0,
c.ITMDES1_0

from SINVOICE a
    inner join SINVOICED b
        on a.NUM_0 = b.NUM_0
    inner join SDELIVERYD c
        on b.SDHNUM_0 = c.SDHNUM_0
        AND b.SDDLIN_0 = c.SDDLIN_0

WHERE
a.NUM_0 = 'FAC1404-680097'
```

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_0 as 'num comm order',
a.ORDDAT_0,
a.SHIDAT_0,
a.DEMDLVDAT_0,
a.LASINVNUM_0,
b.SOHNUM_0 as 'num comm deliv',
b.SDHNUM_0,
b.ITMDES1_0

from SORDER a
    LEFT join SDELIVERYD b
    on a.SOHNUM_0 = b.SOHNUM_0

where
b.SOHNUM_0 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
where
A.ClientId = B.ClientId
And
B.EmployeId=C.EmployeID
And
A.ClientPays ='France'
```

Results | Messages

ClientNom	CdeDate	EmployeNom
Victuailles en stock	2004-07-08 00:00:00.000	Leverling
Blondesddsl père et fils	2004-07-25 00:00:00.000	Fuller
Vins et alcools Chevalier	2004-08-06 00:00:00.000	Suyama
Vins et alcools Chevalier	2004-09-02 00:00:00.000	Fuller
Blondesddsl père et fils	2004-09-04 00:00:00.000	Buchanan

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 distinct City FROM Customers

UNION

SELECT distinct City FROM Suppliers
ORDER BY City;
```

UNION → permet de récupérer les lignes des deux requêtes, tout en supprimant les doublons si des éléments (ici villes) 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 (villes) 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 (villes) qui sont dans la requête qui précède EXCEPT (celle du haut) mais qui ne sont pas dans la suivante (Suppliers).

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

```
USE comptoir_OLTP
SELECT ClientNom
FROM T_CLIENT
WHERE ClientID NOT IN (SELECT ClientId from T_COMMANDE)
--Cette requête peut se substituer à un LEFT JOIN


USE AdventureWorks2012;
GO
SELECT DISTINCT Name FROM Production.Product
WHERE ProductModelID IN
    (SELECT ProductModelID
     FROM Production.ProductModel
     WHERE Name LIKE 'Long-Sleeve Logo Jersey%');
```

Sous requêtes :

Dans la clause WHERE : ALL / ANY / SOME

```
SELECT CUSTOMERid
FROM sales.customer
where TerritoryID
in
-- IN (dans les deux tables)
-- NOT IN (qui n'est pas dans la seconde)

-- SOME = ANY (au moins une valeur : comparaison)
-- <> ALL (toutes les valeurs : comparaison)

(SELECT TerritoryID
 FROM sales.salesPerson)
```

- ALL : signifie supérieur à toutes les valeurs
- Exemple : > ALL(1,2,3) signifie supérieur à 3

- >ANY(1,2,3) = signifie supérieur à la valeur minimale donc supérieur à 1

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)

Sous requêtes Vs jointures

- Ces 2 requêtes retournent le **même** résultat. Les performances sont en faveur du **join**

(si grosse volumétrie)

```
USE AdventureWorks2012;
/* SELECT avec sous requête. */
SELECT Name
FROM AdventureWorks2012.Production.Product
WHERE ListPrice =
    (SELECT ListPrice
     FROM AdventureWorks2012.Production.Product
     WHERE Name = 'Chainring Bolts' );

/* SELECT avec inner join. */
SELECT Prd1. Name
FROM AdventureWorks2012.Production.Product AS Prd1
     INNER JOIN AdventureWorks2012.Production.Product AS Prd2
       ON (Prd1.ListPrice = Prd2.ListPrice)
WHERE Prd2. Name = 'Chainring Bolts';
```

Sous requêtes :

Dans la clause FROM

```
Select avg([Nombre de prod]) as 'Moyenne'
FROM
(
        SELECT count(ProductKey) as 'Nombre de prod',
        colorname
        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

```
use ContosoRetailDW

select
    DC.CustomerKey,
    DC.LastName,
    avg(salesamount) as 'moyenne cus',
    (select avg(salesamount) from FactOnlineSales)  as 'moy_gnrle_VL',
    (select avg(salesamount) from FactSales)  as 'moy_gnrle_VM'
from FactOnlineSales FOS join DimCustomer DC
            on FOS.CustomerKey=DC.CustomerKey
group by DC.CustomerKey, DC.LastName
HAVING avg(salesamount) > (select avg(salesamount) from FactOnlineSales)
```

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),
- Utiliser le mot clé « JOIN (inner n'est pas obligatoire),
- Choisir sous requêtes ou LEFT JOIN pour requêtes d'exclusion,
- 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
- DELETE
- TRUNCATE TABLE

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 T_titre (Tit_Titre, Tit_libellé)
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

Delete from T_Client WHERE condition

- La plus définitive
- On ne l'utilise qu'en développement,
- On préfère « flagger » un client (non actif, par exemple, associé à une date de modification) plutôt que le supprimer de la base de données

Requêtes « ACTION » :
Requête TRUNCATE TABLE

TRUNCATE TABLE T_CLIENT

Vide la table sans « journaliser »
Utilisé essentiellement par les développeurs

TRUNCATE TABLE `table`

DROP TABLE : supprime la table et non plus les lignes comme DELETE ou TRUNCATE.

ANNEXES

Sommaire / Plan détaillé du support de formation

Schémas des bases de données

Liens web utiles

Sommaire / Plan détaillé (1/2)

Sommaire / Plan détaillé (2/2)

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.

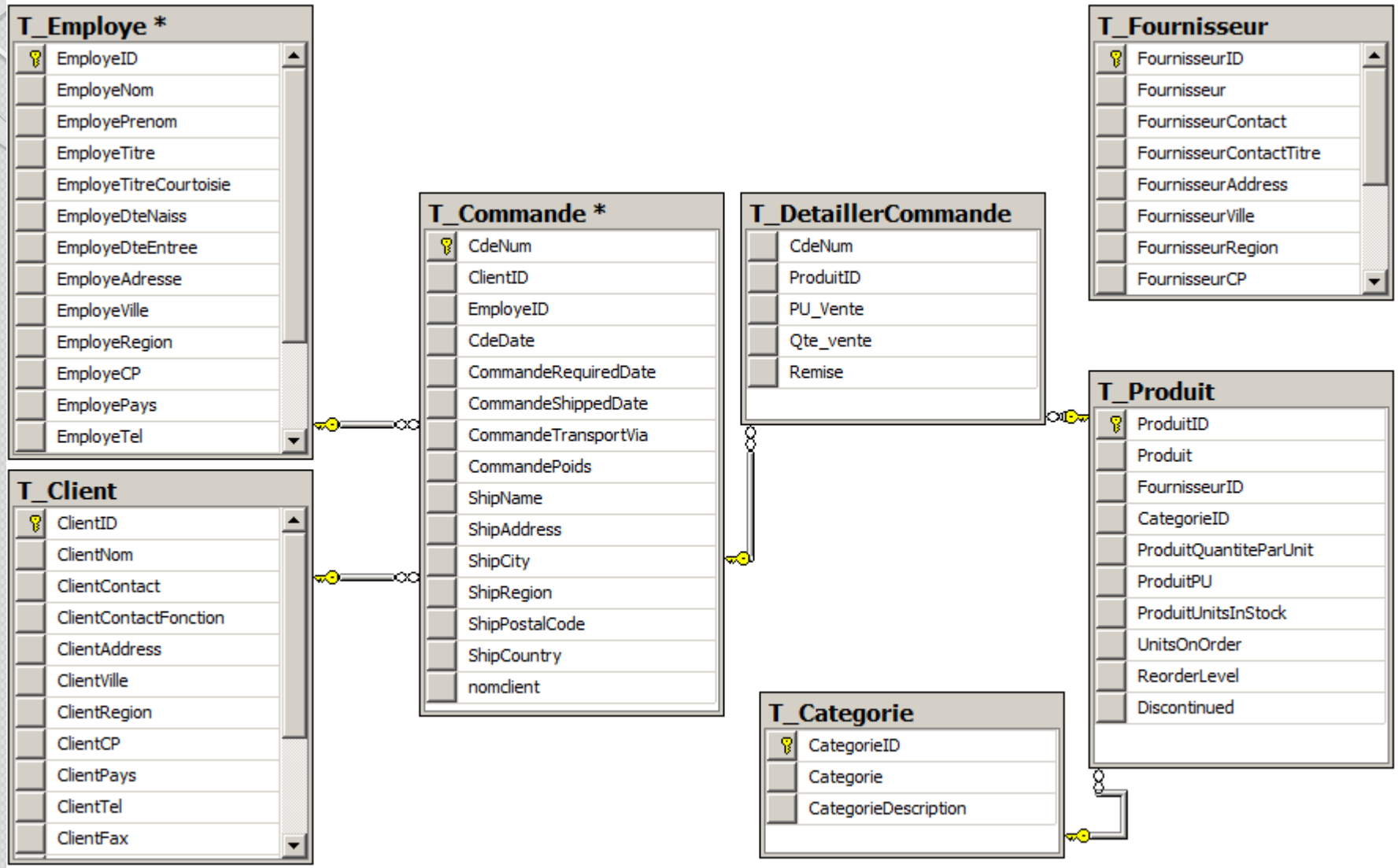
✓ Select ZERTIKL as 'Nom' → s'il n'y a pas d'espaces, les cotes ne sont pas obligatoires mais peuvent être utilisées.

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

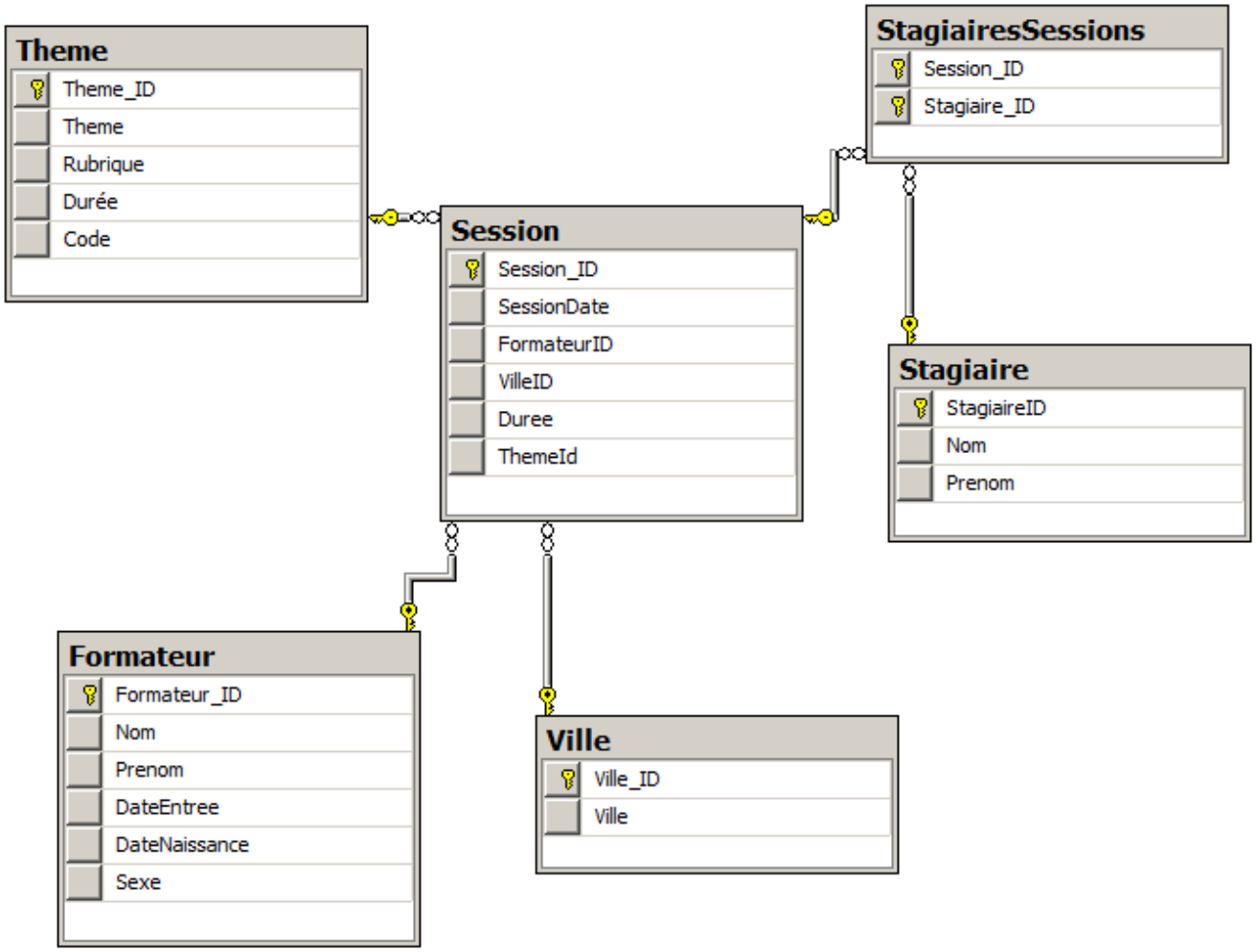
✓ Where ZERTIKL = 'DUPONT'

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

Schémas des bases de données - Comptoir

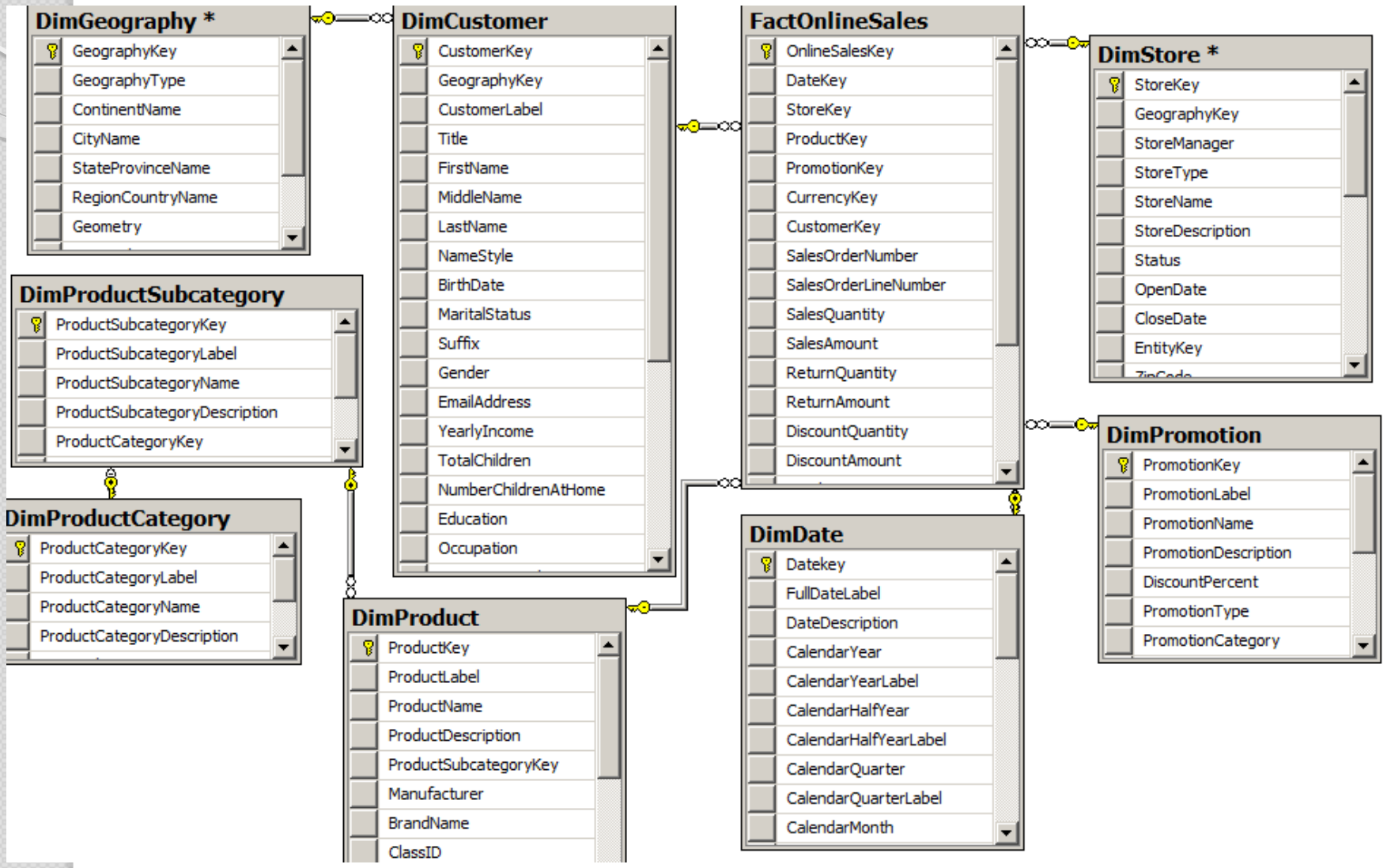


Schémas des bases de données _SQL_SESSION_SIF



Schémas des bases de données

ConsotoRetailDW (simplifié)



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