


Langage SQL

Sparks



The slide features a decorative vertical bar on the left with overlapping circles. The main content area contains the title 'Langage SQL' and subtitle 'Sparks'. Below these are four logos: Microsoft SQL Server 2016 (a red and white globe icon), Oracle (a red rectangle with white text), MySQL (a blue fish icon), and Teradata (orange text with 'Raising Intelligence' below it).

1

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...).

2

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/>

3

3

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

4

4

<h1>Langage SQL</h1> <h2>Le Sommaire</h2>	
• Introduction aux bases de données	Page 6
• Extraire les données d'une table	Page 14
• Calculs et fonctions intégrées	Page 38
• Interroger plusieurs tables : les Jointures	Page 50
• Utiliser des sous-requêtes	Page 66
• Requêtes Actions	Page 76
• Transactions	Page 89
• Annexes	Page III

5

5



6

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

7

7

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	WWF 657
JK92876T6753W9	Nissan	Pathfinder XE	2004	KDF 324
PK8750927GH786	BMW	320 SI	2002	BGH 629

8

8

Éléments constitutifs

Requêtes mono-tables

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

Requêtes multi-tables :

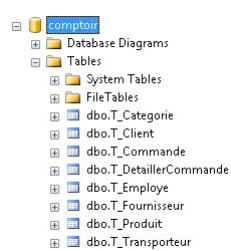
- Le modèle Physique de données (simplifié) **indispensable**
 - ▀ Les clés
 - ▀ Les relations

9

9

Base de données : Structure

Les tables



+ dbo.Dim_Time_PL
 + dbo.Dim_Utilisateur
 + dbo.Employee3
 + dbo.Entrée
 + dbo.erreur
 + dbo.ErreurHoraire
 + dbo.ErreurPaie
 + dbo.Fact_Utilisateur_CI
 + dbo.Horaire
 + dbo.HoraireContrat
 + dbo.HTH

Table_Name
DWH_TNAC_CONTROL_HISTORY
DWH_TICKET_CHECK
DWH_SALE
DWH_V_SALE
DWH_PAYMENTS
ORDER_TABLE
DWH_CONTACT
ITEM

10

10

Base de données : Structure

Les champs

Champs normalisés

- EmployeeID (PK, int, not null)
- EmployeeNom (char(20), not null)
- EmployeePrenom (char(10), not null)
- EmployeeTitre (char(30), null)
- EmployeeTitreCourtoisie (char(25), null)
- EmployeeDteNaiss (datetime, null)
- EmployeeDteEntree (datetime, null)
- EmployeeAdresse (char(60), null)

Champs non normalisés

- PCE_REM_ART (float, null)
- PRX_REM_ART (float, null)
- PCE_REM_PLT (float, null)
- PRX_REM_PLT (float, null)
- PCE_REM_TIC (float, null)
- PRX_REM_TIC (float, null)
- PRX_NET (float, null)
- VAR_ING (varchar(10), null)
- TAX_TVA (float, null)
- COD_TVA (varchar(4), null)
- MNT_TVA (float, null)

11

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 : entier (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)

12

12

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 !

13

13

Langage SQL

EXTRAIRE LES DONNEES D'UNE TABLE

14

14

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 !!

15

15

SELECT et la syntaxe SQL

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

3 mots clés :

SELECT

Champ1,
Champ2

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

FROM

Table

provenant (FROM) du tableau appelé « nom_du_tableau ».

WHERE

Champ3 ='test'

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

16

16

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.

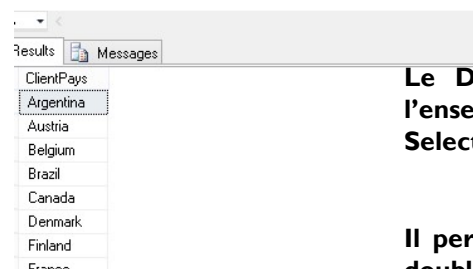
17

17

Filtrage des données :

La clause DISTINCT

```
use comptoir
SELECT DISTINCT ClientPays from T_CLIENT
ORDER BY ClientPays
```



ClientPays
Argentina
Austria
Belgium
Brazil
Canada
Denmark
Finland
France

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

Il permet de supprimer les doublons sur les lignes.

18

18

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'

```

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

19

19

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

20

20

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;
```

21

21

Les commentaires en SQL

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

```
/*
Afficher les 100 premières 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
```

22

22

La commande SELECT :

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

Ouvrir la requête : Ou_sont_erreurs_code.sql

23

23

La commande SELECT :

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

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

OnlineSalesKey	DateKey	StoreKey	ProductKey	PromotionKey	CurrencyKey	CustomerKey	SalesOrderNumt
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:

24

24

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;
```

Example

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

MySQL Syntax

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

Example

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

25

25

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'
```

Collation: French_CLAS

26

26

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

```

27

27

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**

28

28

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 1^{er} caractère.

S'il correspond à P ou à F alors je le sélectionne.

Texte :

Numérique :

= 'France'

= '25'

> '50'

<> 'Spain'

< '50'

<> '25'

29

29

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.

30

30

La commande SELECT :

La clause WHERE : (not)LIKE

```

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

```

Like = qui intègre les caractères
A% qui commence par a
%a qui termine par a

```

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

```

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

31

31

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'

```

	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
c	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

32

32

La commande SELECT :

La clause WHERE : Synthèse

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

33

33

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
```

	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...

34

34

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
```

35

35

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.

36

36

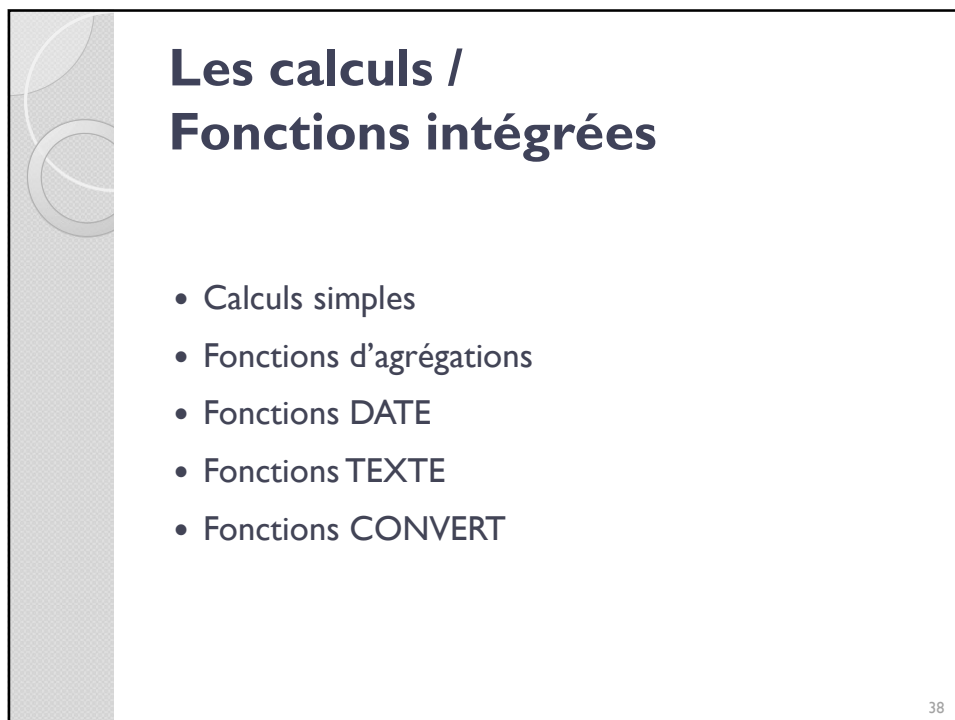


Langage SQL

**LES CALCULS ET
FONCTIONS
INTÉGRÉES**

37

37



**Les calculs /
Fonctions intégrées**

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

38

38

Les calculs / fonctions intégrées

Champs calculé simple

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

vente	remise	hors remise
12,5396	2,59	9,9456
12,5396	2,59	9,9456
12,5396	2,59	9,9456
12,5396	2,59	9,9456
12,5396	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

39

39

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
```

40

40

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**

41

41

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

```
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);
```

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

42

42

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%'

```

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

43

43

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

	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

44

44

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'
```

Laminate_Id	Start_Time
1054179476	2016-04-04 15:05:00.000

45

45

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
```

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

employenom
Davolio
Fuller
Levelling
.....

46

46

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

```

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

47

47

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.

48

48

Langage SQL

LES JOINTURES

49

49

Les jointures

Structure d'une table : Les clés

Clé primaire (primary Key /PK)

Clé étrangère Foreign Key / FK

Relation 0, 1...n

DimProduit	
Produit_PK	Produit
ProduitCode	
SousFamilleCode	
SousFamille	
FamilleCode	
Famille	

FactFacture	
DateFabrication_FK	
Site_FK	
Produit_FK	
Client_FK	
PrixCatalogue	
Remise	
CA	
Marge	
CoutDirectMatiere	
CoutDirectMainOeu...	
CoutIndirect	

50

50

Les jointures

Rôle des clés

formateur_num	formateur_nom	formateur_prenom	formateur_adresse	formateur_tel	formation_code	formateur_num	formation_libelle	formation_datedebut	formation_duree
17	Harabazan	Hamid	Rue zitouni Marrakech	0149253625	MSA	17	MySQL, administration	2014-12-04	5
17	Drissi	Noureddine	Rue du G. De Gaulle	0577253625	ORA	17	Oracle 11g, administration	2014-11-26	5
NULL	NULL	NULL	NULL	NULL	PGA	17	PostgreSQL, administration	2014-12-15	4
					PSQ	17	Le langage SQL	2014-11-19	3
					VMW	10	Vmware, administration	2014-12-17	3
					WAD	10	Windows 7, administration	2014-12-17	3

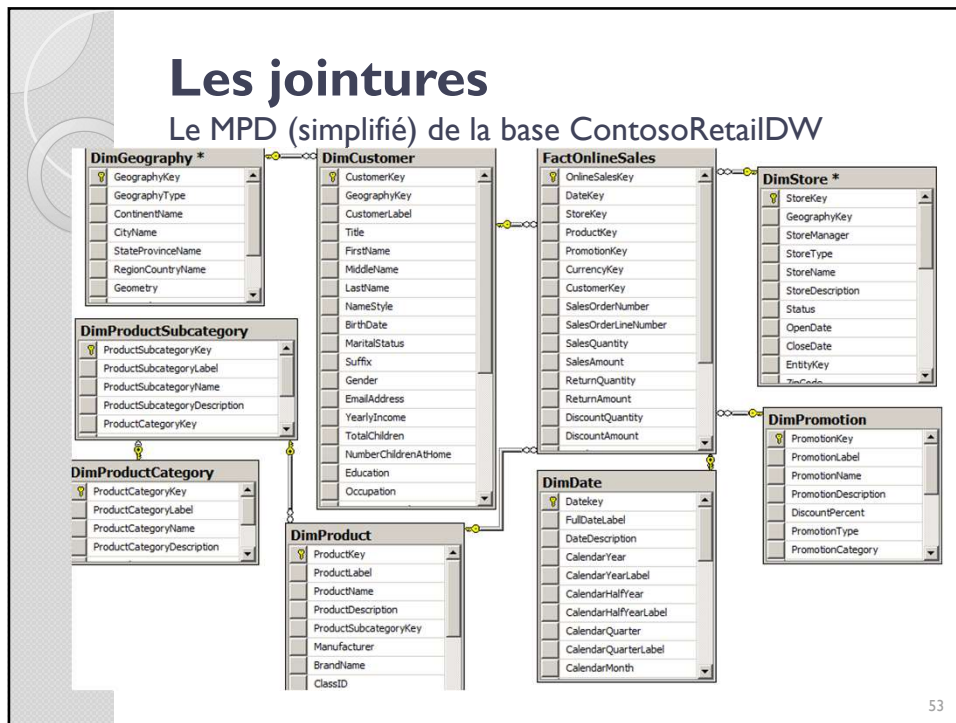
formateur_nom	formation_libelle
Drissi	MySQL, administration
Drissi	Oracle 11g, administration
Drissi	PostgreSQL, administration
Drissi	Le langage SQL
Harabazan	Vmware, administration
Harabazan	Windows 7, administration

51

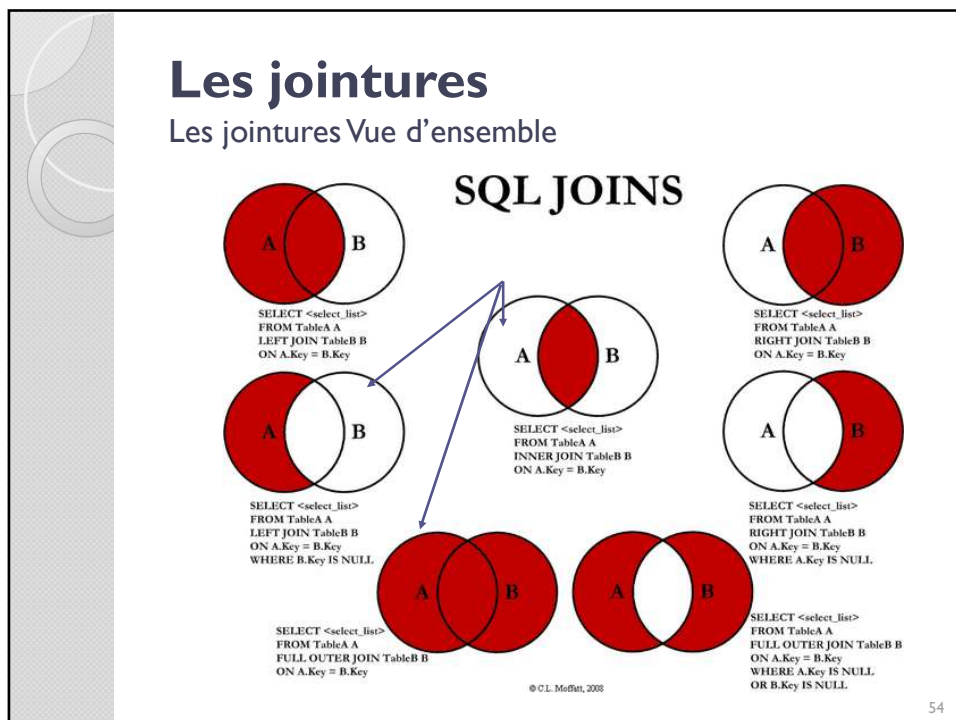
Les jointures

Lire un modèle relationnel (ou physique / MPD)

52



53

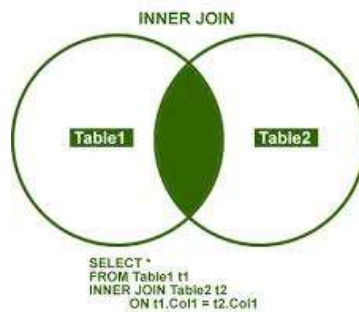


54

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



(C) <http://blog.sqlauthority.com>

55

55

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	DEPTNO	DEPTNAME
AD3100	ADMIN SERVICES	D01	A00	SPIFY COMPUTER SERVICE DIV.
IF1000	QUERY SERVICES	C01	B01	PLANNING
IF2000	USER EDUCATION	E01	C01	INFORMATION CENTER
MA2100	WELD LINE AUTOMATION	D01	D01	DEVELOPMENT CENTER
PL2100	WELD LINE PLANNING	B01		

↓
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

56

56

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'
```

Commandes C

- IdCommande
 IdClient
 Date

57

57

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'
```

58

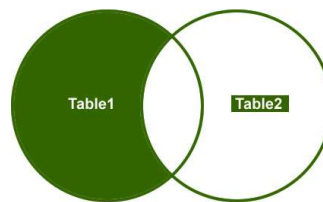
58

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.

LEFT OUTER JOIN - WHERE NULL



```
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.Col1 = t2.Col1
WHERE t2.Col1 IS NULL
```

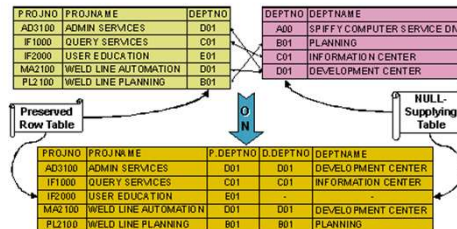
(C) <http://blog.SQLAuthority.com>

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
```



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 !

61

61

Les jointures

« Old School »

```

select
A.ClientNom,
B.CdeDate,
C.EmloyeNom
from
T_Client A, T_Commande B, T_Employe C
where
A.ClientId = B.ClientId
And
B.EmloyeId=C.EmloyeID
And
A.ClientPays = 'France'

```

ClientNom	CdeDate	EmloyeNom
Victualles en stock	2004-07-08 00:00:00.000	Leverling
Blondesdls 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
Blondesdls père et fils	2004-09-04 00:00:00.000	Buchanan

62

62

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.

63

63

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.

64

64

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).

65

65

Langage SQL

LES SOUS REQUÊTES

66

66

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

67

67

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%');
```

68

68

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

69

69

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);
```

3 différences avec les requêtes simples :

- Dans la condition where et le prédicat Exists : aucun champ car on va lier les requêtes par les tables entre la 1^{ère} 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 (corrélations des requêtes)

70

70

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';
```

71

71

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

72

72

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)

```

73

73

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.

74

74

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

75

75

Langage SQL

LES REQUÊTES ACTIONS

76

76

Requêtes « ACTION » :

- SELECT..... INTO
- INSERT INTO
- UPDATE
- DELETE
- TRUNCATE TABLE

77

77

Requêtes « ACTION » :

INSERT : Ne renseigner que certains champs

```
use DistrisysDW
```

```
insert into DimSite  
(SiteCode,Site)
```

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

78

78

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 (Titre, Tit libellé) insert T_titre
values ('M.', 'Monsieur'),          values ('M.', 'Monsieur'),
      ('Mlle.', 'Mademoiselle'),    ('Mlle.', 'Mademoiselle'),
      ('Mme.', 'Madame')            ('Mme.', 'Madame')
```

79

79

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';
```

80

80

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 = 'valeur 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

81

81

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 Amélie',
    ville = 'Saint-Eustache-la-Forêt',
    code_postal = '76210'
WHERE id = 2
```

Cette requête sert à définir la colonne rue à « 49 Rue Amélie », 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 Amélie	Saint-Eustache-la-Forêt	76210	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

82

82

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

83

83

Requêtes « ACTION » :

Requête UPDATE

```
UPDATE T_Produit
SET Produit_PU = Produit_PU * 1.1
```

Where

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

84

84

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 entier/entier dont on n'autorise pas les null,

Un Nom de type Varchar (caractère 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.

85

85

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 titre, un Ida de type entier qui fait référence à une clé étrangère relative au champs PK Idartiste de la table artiste.

86

86

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

87

87

Requêtes « ACTION » :

Requête TRUNCATE TABLE

TRUNCATE TABLE T_CLIENT

Vide la table sans « journaliser »

Utilisé essentiellement par les développeurs

TRUNCATE TABLE `table`

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

88

88

Langage SQL

LES TRANSACTIONS

89

89

Implémentation SQL

Transactions : Définition

- Une transaction est un **traitement cohérent et fiable**. Dans le cas des bases de données, c'est une action qui transforme la base de données d'un état stable cohérent en un autre état stable cohérent (après Insertion, suppression ou mise à jour de données)
- En cas d'interruption pour une raison quelconque, la transaction garantit de laisser la base de données dans l'état dans lequel elle l'avait trouvée (**Rollback**). **On dira ici que l'unité logique de traitement (transaction) est complètement abandonnée.**
- En cas de réussite, la base contient les données modifiées (ou nouvelles) (**Commit**). **On dira ici que l'unité logique de traitement (transaction) est complètement exécutée.**
- Une transaction a quatre types d'opérations : un début de transaction, la lecture, l'écriture, et enfin la terminaison.

90

Implémentation SQL

Transactions : Définition

- **COMMIT**
- La commande COMMIT termine une transaction avec succès ; toutes les mises à jour de la transaction sont validées
- On dit que la transaction est validée
- Tous ses effets sont alors connus des autres transactions s'exécutant concurremment.

- **ROLLBACK**
- La commande ROLLBACK termine une transaction avec échec ; toutes les mises à jour de la transaction sont annulées (tout se passe comme si la transaction n'avait jamais existé).
- On dit que la transaction est annulée.
- Aucune des opérations effectuées par cette transaction n'est connue des autres transactions

91

Implémentation SQL

Vues

- Une vue est une table virtuelle, c'est-à-dire dont les données ne sont pas stockées dans une table de la base de données, et dans laquelle il est possible de rassembler des informations provenant de plusieurs tables.

- On parle de "vue" car il s'agit simplement d'une représentation des données dans le but d'une exploitation visuelle.
- Les données présentes dans une vue sont définies grâce à une clause SELECT

92

Implémentation SQL

CTE

- Une expression de table commune (CTE, Common Table Expression) peut être considérée comme un **jeu de résultats temporaire** défini dans l'étendue d'exécution d'une seule instruction SELECT, INSERT, UPDATE, DELETE ou CREATE VIEW.
- Elles permettent de remplacer les sous requêtes dans le From (table dérivée) qui rendent un **code illisible** (et souvent sans commentaire) dans une requête standard.

93

Implémentation SQL

CTE : exemple (1/2)

-- version en sous-requête : table dérivée

```
select count(*) as nb
from
    (select
      CUS.lastname,
      GEO.[RegionCountryName],
      sum(FAC.salesamount) as total
    from
      DimCustomer CUS
    inner join DimGeography GEO
    on CUS.GeographyKey=GEO.GeographyKey
    inner join FactOnlineSales FAC
    on FAC.CustomerKey=CUS.CustomerKey
    group by
      CUS.lastname,
      GEO.[RegionCountryName]) as fun
```

94

Implémentation SQL

CTE : exemple (2/2)

with fun as

```
(select
  CUS.lastname,
  GEO.[RegionCountryName],
  sum(FAC.salesamount) as total
from
  DimCustomer CUS
  inner join DimGeography GEO
  on CUS.GeographyKey=GEO.GeographyKey
  inner join FactOnlineSales FAC
  on FAC.CustomerKey=CUS.CustomerKey
group by
  CUS.lastname,
  GEO.[RegionCountryName])
```

```
select count(*) as nb
from fun
```

95

Implémentation SQL

Procédures stockées

- Une procédure stockée est un groupe d'instructions Transact-SQL qui est compilé une fois pour toutes et qui peut être exécuté plusieurs fois.
- Lorsqu'elle est exécutée, les performances sont améliorées car les instructions Transact-SQL ne doivent pas être recompilées.

96

Implémentation SQL

Procédures stockées (1/2)

Je crée ma procédure stockée :

- Je vais la nommer et créer la variable,
- Je met ma requête qui utilise ma variable dans la procédure stockée (SP: stocked procedure)
- J'exécute le tout → enregistrement de la SP.

```
create proc sp_nbclientpays @strPays varchar(50)
as
select
clientpays,
count(clientpays) as 'Nombre de clients'
from T_Client
Where clientpays = @strPays
Group by ClientPays
```

Puis donner la valeur pour variable : exec sp_nbclientpays 'SPAIN'

97

97

Implémentation SQL

Procédures stockées (2/2)

Je crée ma procédure stockée avec deux variables :

```
create proc sp_nbclientpays2 @strPays varchar(50), @strclient varchar(5)
as
select
clientpays,
clientnom,
count(clientpays)
from T_Client
Where clientpays= @strPays
and left(ClientNom,1) = @strclient
Group by ClientPays, ClientNom
```

Puis les valeurs : exec sp_nbclientpays2 'FRANCE', 'B'

98

98

Implémentation SQL

Déclaration BEGIN... END

Begin

```
-- Declaring a variable
Declare @v_Result Int;
-- Declaring a variable with a value of 50
Declare @v_a Int = 50;
-- Declaring a variable with a value of 100
Declare @v_b Int = 100;

-- Print out Console (For developer).
-- Using Cast to convert Int to String
-- Using + operator to concatenate 2 string
Print 'v_a= ' + Cast(@v_a as varchar(15));

-- Print out Console
Print 'v_b= ' + Cast(@v_b as varchar(15));

-- Sum
Set @v_Result = @v_a + @v_b;

-- Print out Console
Print 'v_Result= ' + Cast(@v_Result as varchar(15));

End;
```

Résultat :

```
v_a= 50
v_b= 100
v_Result= 150
```

99

99

Implémentation SQL

Gestion des erreurs : 1^{er} exemple

Je veux attraper mon erreur :

```
set nocount on
```

```
begin try -- fonction
select 1/0 -- je fais un essai : requête avec une
possibilité d'erreur
end try
```

```
begin catch -- j'attrape -- interception erreur car il
n'aime pas si on divise par 0 :
print 'division par 0';
throw; -- je jette car je n'aime pas

end catch
```

100

100

Implémentation SQL

Gestion des erreurs : 2 exemple

-- un ajout ou une suppression ou une mise à jour commence toujours par un begin transaction

```
BEGIN TRANSACTION;

BEGIN TRY -- code dans de l'applicatif
    -- Generate a constraint violation error.
DELETE FROM Production.Product
WHERE ProductID = 980; -- si on lance la requête il y a
une contrainte d'intégrité référentielle
-- il interdit de supprimer un produit si déjà commandé :
déjà clé étrangère dans une table.
END TRY
```

101

101

Implémentation SQL

Gestion des erreurs : suite

```
BEGIN CATCH -- si pas de problème, il ne viendra pas ici
    SELECT -- si problème, il vient exécuter toutes les
lignes qu'il y a jusqu'au End Catch
        ERROR_NUMBER() AS ErrorNumber
-- il fait un select pour nous afficher le numéro de
l'erreur
        ,ERROR_SEVERITY() AS ErrorSeverity
-- sévérité systeme (message d'erreurs systèmes)
        ,ERROR_MESSAGE() AS ErrorMessage;
-- le message d'erreur

    IF @@TRANCOUNT > 0
    -- variable système : j'exécute ou pas
        ROLLBACK TRANSACTION; -- à ce moment il annule
la transaction (delete) il revient dans l'état initial de
la table)
END CATCH;
```

102

102

Implémentation SQL

Déclaration IF ELSEIF ELSE

```

IF <condition 1> THEN
    Job 1;
[ELSIF <condition 2> THEN
    Job 2;
]
[ELSE
    Job n + 1;
]
END IF;

```

- si la condition 1 est respectée alors JOB 1
- sinon, si condition 2 est validée alors JOB 2 (action à faire),
- pour tous les autres Job N+1
- Fin du IF

103

103

Implémentation SQL

Déclaration IF ELSEIF ELSE : exemple

```

BEGIN
-- Declare a variable
DECLARE @v_Option integer;
DECLARE @v_Action varchar(30);

SET @v_Option = 2;

IF @v_Option = 1
    SET @v_Action = 'Run';
ELSE IF @v_Option = 2
    BEGIN
        PRINT 'In block else if @v_Option = 2';
        SET @v_Action = 'Backup';
    END;
ELSE IF @v_Option = 3
    SET @v_Action = 'Stop';
ELSE
    SET @v_Action = 'Invalid';

- Logging
PRINT '@v_Action= ' + @v_Action;

END; /*

```

Que permet de
faire cette requête
?

Résultat :

In block else if
@v_Option = 2
@v_Action= Backup

104

104

Implémentation SQL WHILE : La boucle

```

BEGIN
-- Declaring 2 variables x and y.
DECLARE @x integer = 0;
DECLARE @y integer = 10;

-- Step
DECLARE @step integer = 0;

-- While @x < @y
WHILE (@x < @y)
BEGIN
    SET @step = @step + 1;

    -- Every time loop execute, x increases by 1.
    SET @x = @x + 1;
    -- Every time loop execute, x decreases by 2.
    SET @y = @y - 2;

    PRINT 'Step =' + CAST(@step AS varchar(10));
    PRINT '@x =' + CAST(@x AS varchar(10)) + ' / @y = ' + CAST(@y AS varchar(10));

END;

-- Write log
PRINT 'x,y = ' + CAST(@x AS varchar(10)) + ', ' + CAST(@y AS varchar(10));
END;

```

Je crée une boucle :
Quand x est plus petit que y.

Je monte d'une étape à chaque fois.

Mes données x passent à la supérieure.
Mes données y passent à l'inférieur.

105

105

Implémentation SQL WHILE : La boucle et le BREAK

```

BEGIN
-- Declaring 2 variables x and y
DECLARE @x integer = 0;
DECLARE @y integer = 10;

-- Step
DECLARE @step integer = 0;

-- While @x < @y
WHILE (@x < @y)
BEGIN
    SET @step = @step + 1;

    -- Every time the loop execute, x increases by 1
    SET @x = @x + 1;
    -- Every time the loop execute, y decreases by 1
    SET @y = @y - 2;

    PRINT 'Step =' + CAST(@step AS varchar(10));
    PRINT '@x =' + CAST(@x AS varchar(10)) + ' / @y = ' + CAST(@y AS varchar(10));

    -- If @x > 2 then exit the loop
    -- (Although conditions in the WHILE is still true).
    IF @x > 2
        BREAK;

END;

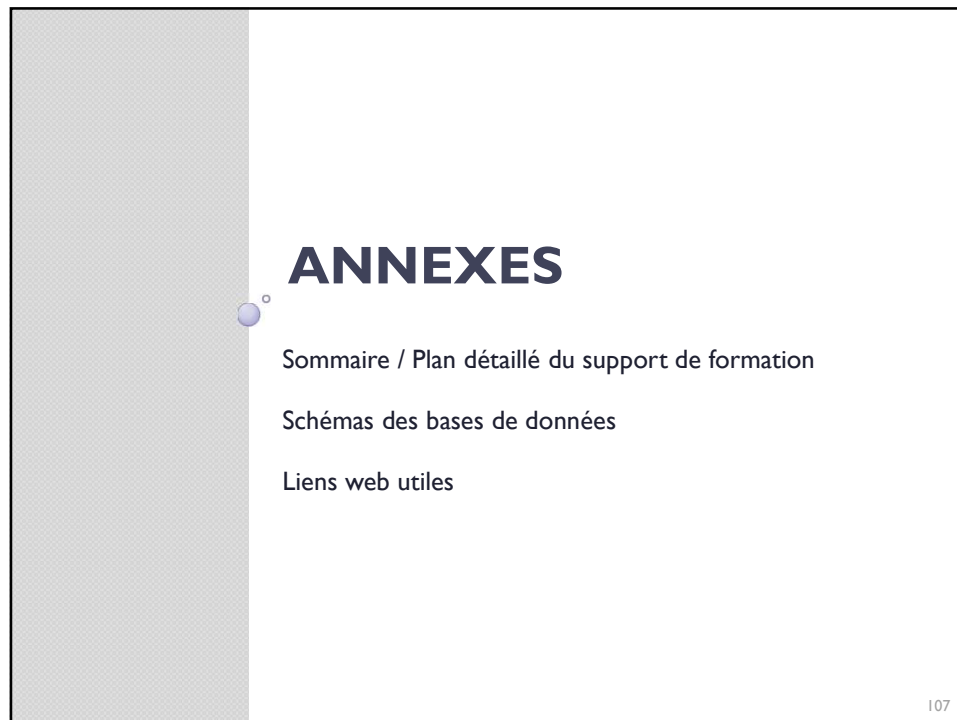
-- Write log
PRINT 'x,y = ' + CAST(@x AS varchar(10)) + ', ' + CAST(@y AS varchar(10));
END;

```

Je stoppe ma boucle si les données suivent une autre condition.

106

106



107

Sommaire / Plan détaillé (1/3)	
Objectifs de la formation	Page 4
Sommaire synthétique	Page 5
Introduction aux bases de données	
Base données et présentation	Page 8 / 9
Les Tables	Page 10
Les champs	Page 11
Les types de données	Page 12
Extraire les données d'une table	
Commande Select et syntaxe	Page 14
La Clause DISTINCT	Page 16 / 17
Select et règles d'écritures : erreurs type	Page 18
Les commentaires en SQL	Page 20
Clause Top	Page 22
Sensibilité à la casse	Page 24 / 25
	Page 26
Renommer les colonnes	Page 27
La Clause ORDER BY	Page 28
La clause WHERE : critères simples	Page 29
La clause WHERE : (not) IN	Page 30
La clause WHERE : (not) LIKE	Page 31
La clause WHERE : BETWEEN AND	Page 32
La clause WHERE : Synthèse	Page 33
La clause CASE	Page 34 – 36
Calculs et fonctions intégrées	
Champs calculés simples	Page 37
Les fonctions d'agrégation	Page 39
Group by : Clause where et Clause having	Page 40
	Page 41

108

108

Sommaire / Plan détaillé (2/3)

Fonctions Date	Page 44-46
Fonctions textes (chaînes de caractères)	Page 47
La fonction CAST et la conversion	Page 48
Les jointures	Page 49
Structure d'une table : les clés	Page 50
Rôle des clés	Page 51
Modèles relationnels (MPD)	Page 52-53
Les jointures : vue d'ensemble	Page 54
Les jointures : INNER JOIN	Pages 55-56
Renommer les tables	Pages 57-58
Les jointures : LEFT JOIN	Pages 59-61
Jointures Old School	Page 62
Les jointures : Synthèse et Méthodologie	Page 63-64
Clauses UNION, INTERSECT, EXCEPT	Page 65
Les sous-requêtes	Page 66
Dans la clause WHERE: IN ou NOT IN (requêtes simples)	Pages 68
Dans la clause WHERE : ALL / ANY / SOME	Page 69
Dans la clause WHERE: EXISTS (requêtes corrélées)	Page 70
Sous-requêtes VS Jointures	Page 71
Sous-requêtes dans la clause FROM	Page 72
Sous-requêtes dans la clause SELECT	Page 73
Les sous-requêtes : Synthèse et Méthodologie	Page 74
Règles à ne jamais oublier	Page 75

109

109

Sommaire / Plan détaillé (3/3)

Les requêtes actions	Page 76
INSERT	Page 78
INSERT à partir d'une requête	Page 80
UPDATE	Page 81
CREATE TABLE	Page 85
DELETE	Page 87
TRUNCATE ET DROP	Page 98
Les transactions	Page 89
Définitions	Page 90
COMMIT et ROLLBACK	Page 91
Vues	Page 92
CTE	Page 93
Procédures stockées	Page 96
BEGIN END	Page 99
Gestion des erreurs	Page 100
IF ELSEIF ELSE	Page 103
WHILE	Page 105
Annexes	Page 107
Sommaire / Plan détaillé du support de formation	Page 108
Schémas des bases de données	Page 115
Liens Utiles	Page 118

110

110

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 table3` (JOIN/ON et/ou virgules)
`On table1.PK=table2.FK`
`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)

111

111

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)

112

112

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

113

113

La commande SELECT :

Quand faut-il mettre des cotes ?

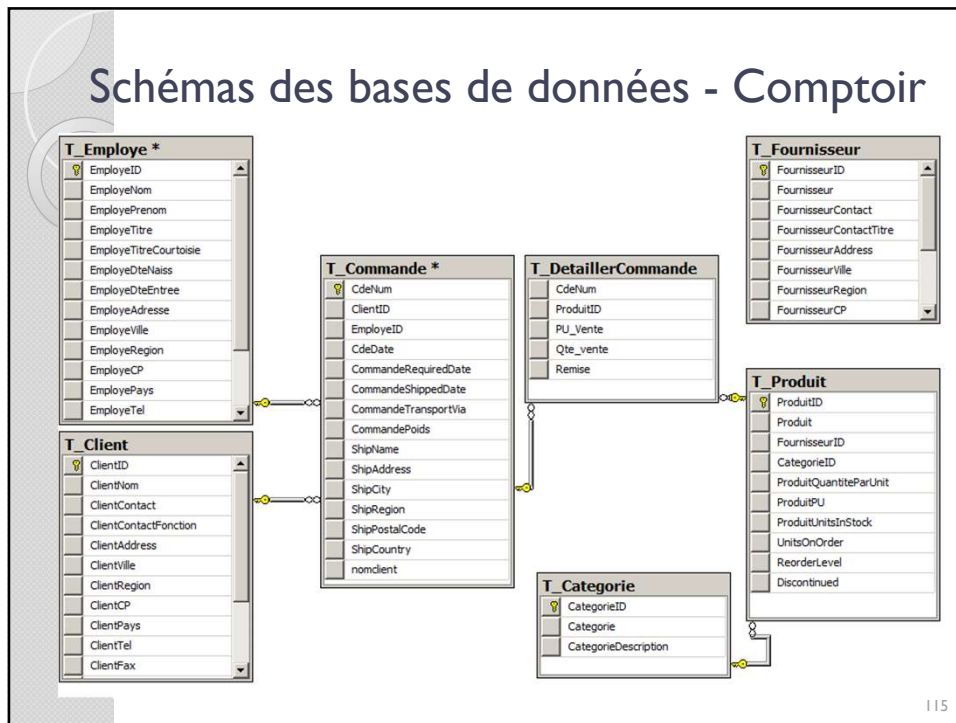
Les cotes sont **obligatoires dans 2 cas** :

- **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.

114

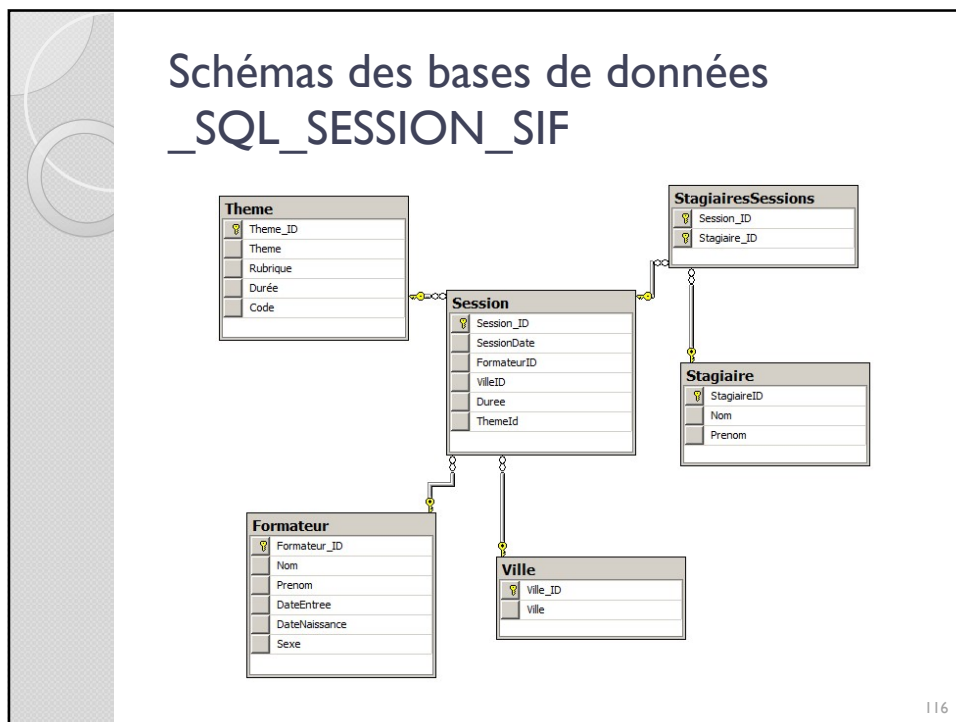
114

Schémas des bases de données - Comptoir



115

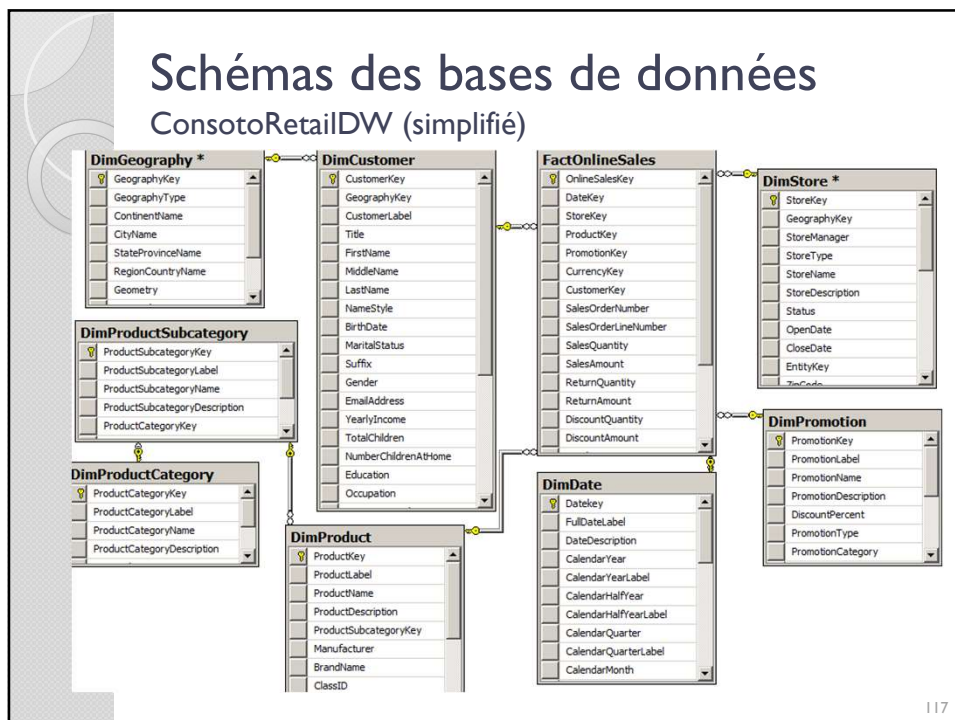
Schémas des bases de données _SQL_SESSION_SIF



116

Schémas des bases de données

ConsotoRetailDW (simplifié)



117

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\(v=sql.105\).aspx](https://technet.microsoft.com/fr-fr/library/ms190750(v=sql.105).aspx) developpez.com/#apprendre-sql

118

MERCI POUR VOTRE ATTENTION

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

Bonne continuation



119

119