






# Transact SQL

## Débutant et perfectionnement



1

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

2

# Langage SQL

## La Formatrice

- Laure BERENGUER, consultante et formatrice indépendante
  
- [berenguer.laure@laposte.net](mailto: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

<b>Langage SQL</b>	
<b>Le Sommaire</b>	
• Introduction aux bases de données	Page 6
• Extraire les données d'une table	Page 14
• Calculs et fonctions intégrées	Page 39
• Interroger plusieurs tables : les Jointures	Page 52
• Utiliser des sous-requêtes	Page 68
• Requêtes d'actions	Page 79
• Les transactions	Page 92
• Annexes	Page 118

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 :

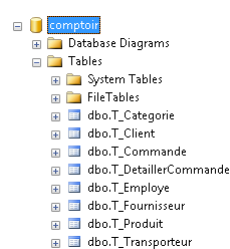
- Le modèle Physique de données (simplifié)  
**indispensable**
- Les clés
- Les relations

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

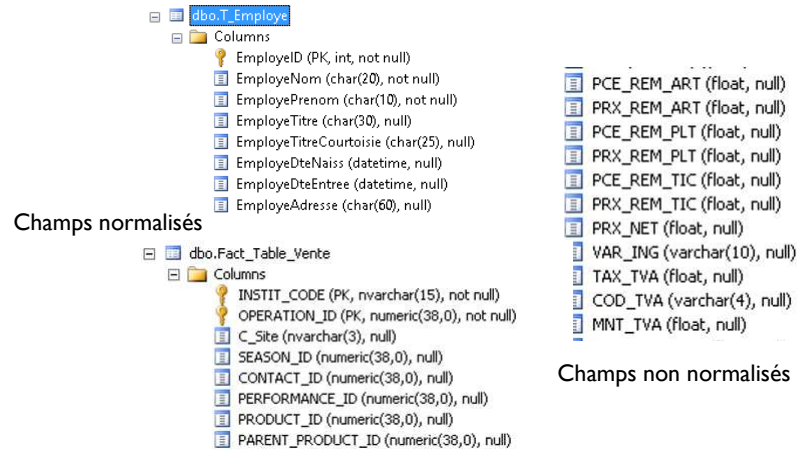
Table_Name
DWH_TNAC_CONTROL_HISTORY
DWH_TICKET_CHECK
DWH_SALE
DWH_V_SALE
DWH_PAYMENTS
ORDER_TABLE
DWH_CONTACT
ITEM

10

10

# Base de données : Structure

## Les champs



11

# Base de données : Structure

## Les types de données courants

### Texte

- Char(n), VarChar(n), nChar(n), nVarchar(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, Smallint, bigint selon la valeur maximale acceptée
- 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, SmallDateTime (« juste » la date, ou date jusqu'au centièmes de seconde)

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

## Les instructions SQL (DDL – DCL – DML)

### **DDL**

Data Definition Language  
Déclarations (DDL) sont utilisés pour définir la structure de base de données ou un schéma.

### **DCL**

Data Control Language  
Contrôle des droits d'accès

### **DML (C'est LE langage (99%) des utilisateurs SQL)**

Data Manipulation Language  
Insert, Update, Delete, Select

### **TCL**

Transaction contrôle des déclarations (TCL) sont utilisés pour gérer les modifications apportées par les instructions DML  
Commit, RollBack..

16

16



## 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

17

17

## 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.

**SELECT DISTINCT**

**ClientPays**

Retourne la liste des pays des clients  
sans doublons (on affiche distinctement)

18

18

## 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**

19

19

## La commande SELECT : Exemples SQL server

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

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

```
use ContosoRetailDW
select
firstame,
Lastname,
Education
from dimcustomer
where
education = 'Bachelors'
```

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

20

20

## 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

21

21

## 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;
```

22

22

## 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
```

23

23

## La commande SELECT :

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

Ouvrir la requête : Ou\_sont\_erreurs\_code.sql

24

24

## 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	SalesOrderNum
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

25

25

## 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;
```

26

26

## La commande SELECT :

Sensibilité à la casse (majuscule / minuscule)

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

27

27

## 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
```

28

28

## La commande SELECT :

La clause ORDER BY

```
select |
ClientNom,
ClientRegion,
ClientPays

from T_Client
order by ClientPays, ClientVille desc
```

ordre **croissant**

Ordre **décroissant**

29

29

## 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')
```

**Texte :**

= 'France'

<> 'Spain'

**Dates :**

Between '12/01/1950' and '25/06/1970'

**Numérique :**

= '25'

> '50'

< '50'

<> '25'

30

30

## La commande SELECT :

La clause WHERE : (not) IN

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

Select *

from Orders

where
Year in ('2008', 2009, 2012)
and
Month in ('april', 'october')
```

31

31

## 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

32

32



## La commande SELECT :

La clause WHERE : Between Vs > & <

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
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

33

33

## La commande SELECT :

La clause WHERE : Synthèse

= <>	exactement égal à une seule valeur where ClientPays <> 'France'
(not) in	exactement égal à un ensemble de valeurs (remplace un or) where ClientPays not in ('France', 'Espagne') where (ClientPays <> 'France' or ClientPays <> 'Espagne')
(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 numbers between 5 and 7

34

34

## 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

35

35

## 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

```

36

36

## La commande SELECT :

### La clause CASE (Oracle)

```
SELECT cust_last_name,
       CASE credit_limit WHEN 100 THEN 'Low'
       WHEN 5000 THEN 'High'
       ELSE 'Medium' END
FROM customers;
```

CUST_LAST_NAME	CASECR
...	
Bogart	Medium
Nolte	Medium
Loren	Medium
Gueney	Medium

```
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;
```

37

37

## La commande SELECT :

### CASE imbriqués

```
select
Article = CASE
  WHEN pr.ArticleId = 4 THEN
    CASE
      WHEN pr.LevelCode = '1A' THEN '1d. Art7A'
      WHEN pr.LevelCode = '2+B' THEN '1c. Art7B'
      WHEN pr.LevelCode = '2C' THEN '1b. Art7C'
      WHEN pr.LevelCode in ('3D','4') THEN '1a. Art7D'
      ELSE '1e. Art7Inconnu'
    END
  WHEN pr.ArticleId = 5 THEN '2. Art8'
  WHEN pr.ArticleId = 7 THEN
    CASE
      WHEN pr.KnowledgeLevelCode = 'E' THEN '3a. Art92E'
      WHEN pr.KnowledgeLevelCode = 'V' THEN '3a. Art92V'
      ELSE '3b. Art92Inconnu'
    END
  WHEN pr.ArticleId = 8 THEN '4. Art10'
  WHEN pr.ArticleId = 9 THEN '5. Art11'
  WHEN pr.ArticleId = 10 THEN '6. Art12'
  WHEN pr.ArticleId = 11 THEN '7. Art13'
  WHEN pr.ArticleId = 12 THEN
    CASE
      WHEN pr.KnowledgeLevelCode = 'E' THEN '8a. Art14E'
      WHEN pr.KnowledgeLevelCode = 'V' THEN '8a. Art14V'
      ELSE '8b. Art14Inconnu'
    END
  ELSE '9. ArtInconnu'
END,
```

38

38

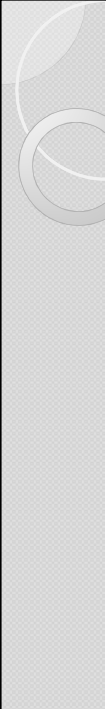


# Langage SQL

## LES CALCULS ET FONCTIONS INTÉGRÉES

39

39



## Les calculs / Fonctions intégrées

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

40

40

## 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
12.5356	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

41

41

## 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>)

```
/*CA global (toutes années et toutes cdes confondues)*/
select sum(Qte_vente*PU_vente) as CA
from
T_DetaillerCommande
/*nombre de commandes passées en 2004*/
select count(*)
from T_commande
where year(CdeDate)=2004
```

42

42

## Les calculs / fonctions intégrées

Les fonctions d'agrégations

GROUP BY : Regroupement et agrégation de données

```
SELECT
Year(CdeDate) as Annee,
COUNT(CdeNum) as 'nb cdes'
From T_COMMANDE
GROUP BY Year(CdeDate)
```

```
SELECT
SalesOrderID,
SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
Having SUM(LineTotal) >1000
```

- Utilisation des fonctions d'agrégation
- Utilisation de la clause Group By
- Filtrage de groupes avec la clause Having

43

43

## 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**

44

44

# 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	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	774	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

45

45

# 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

46

46

# Les calculs / fonctions intégrées

## Les fonctions : Date (SQL Server)

```
select
CdeNum,
CdeDate,
GETDATE() as 'date jour',
year(cdedate) as 'year',
month(cdedate) as 'mois',
Datepart("weekday", cdedate) as 'jour semaine',
Datepart("YYYY", cdedate) as 'année',
Datepart("QUARTER", cdedate) as 'trimestre',
DATEDIFF(day, cdedate, GETDATE()) as 'delai en jours',
DATEDIFF(MM, cdedate, GETDATE()) as 'delai en mois',
DATEDIFF(YEAR, cdedate, GETDATE()) as 'delai 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 jour	year	mois	jour semaine	année	trimestre	delai en jours	delai en mois	delai en mois
1	10249	2004-07-05 00:00:00.000	2017-04-04 13:59:22.210	2004	7	2	2004	3	4656	153	13
2	10250	2004-07-08 00:00:00.000	2017-04-04 13:59:22.210	2004	7	5	2004	3	4653	153	13
3	10251	2004-07-08 00:00:00.000	2017-04-04 13:59:22.210	2004	7	5	2004	3	4653	153	13
4	10252	2004-07-09 00:00:00.000	2017-04-04 13:59:22.210	2004	7	6	2004	3	4652	153	13
5	10253	2004-07-10 00:00:00.000	2017-04-04 13:59:22.210	2004	7	7	2004	3	4651	153	13
6	10254	2004-07-11 00:00:00.000	2017-04-04 13:59:22.210	2004	7	1	2004	3	4650	153	13
7	10255	2004-07-12 00:00:00.000	2017-04-04 13:59:22.210	2004	7	2	2004	3	4649	153	13
8	10256	2004-07-15 00:00:00.000	2017-04-04 13:59:22.210	2004	7	5	2004	3	4646	153	13
9	10257	2004-07-16 00:00:00.000	2017-04-04 13:59:22.210	2004	7	6	2004	3	4645	153	13

47

47

# 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
1	1054179476	2016-04-04 15:05:00.000

48

48



## Les calculs / fonctions intégrées

### La clause WHERE : Les critères avec fonctions

datediff = différence  
entre 2 dates

```
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
Levellings

49

49

## 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 chaîne',
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 chaîne	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

50

50

## Les calculs / fonctions intégrées

### Convertir : La fonction CAST

```

select
  cast(25.5 as int) + cast(15.6 as int) + cast(33.6 as int) as result
go -- 25,5 deviendra 25

select GETDATE() as 'date entière',
       cast(getdate() as date) as 'date simple',
       convert(date, getdate()) as 'date simple convert',
       convert(varchar(20), getdate(),108) 'date format 108',
       convert(varchar(20), getdate(),107) 'date format 107',
       convert(varchar(20), getdate(),113) 'date format 113'

```

Results		Messages				
result						
1	73					
	date entière	date simple	date simple convert	date format 108	date format 107	date format 113
1	2018-10-09 09:07:17.990	2018-10-09	2018-10-09	09:07:17	Oct 09, 2018	09 Oct 2018 09:07:17

51

51

## Langage SQL

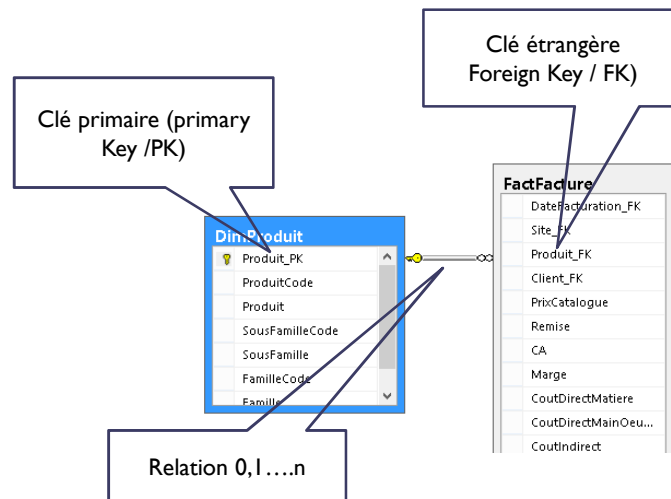
## LES JOINTURES

52

52

## Les jointures

Structure d'une table : Les clés



53

53

## Les jointures

Rôle des clés

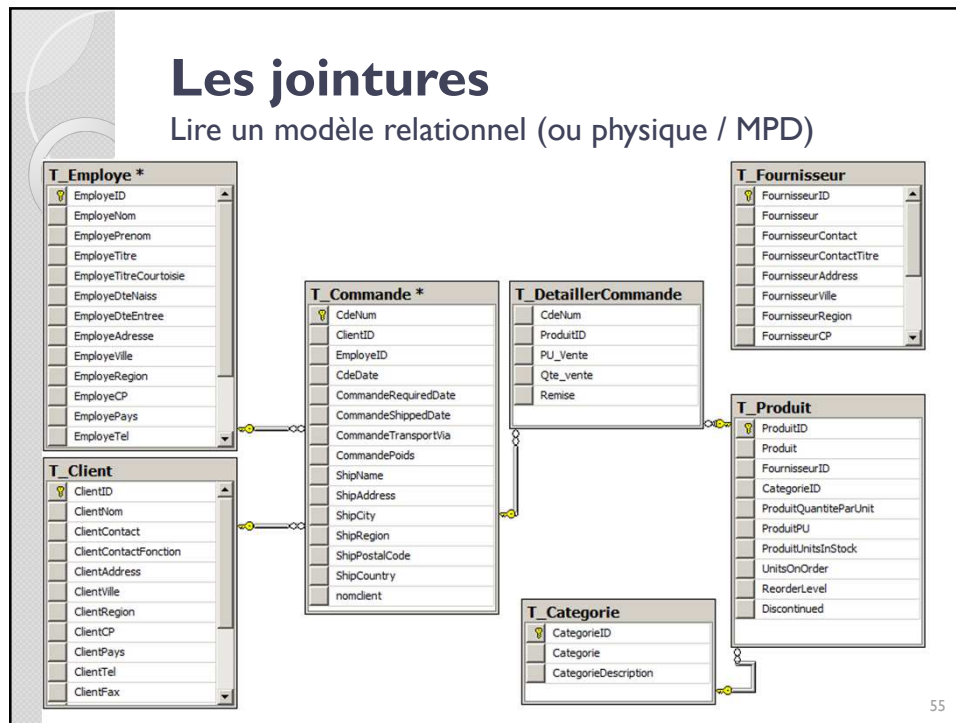


54

54

# Les jointures

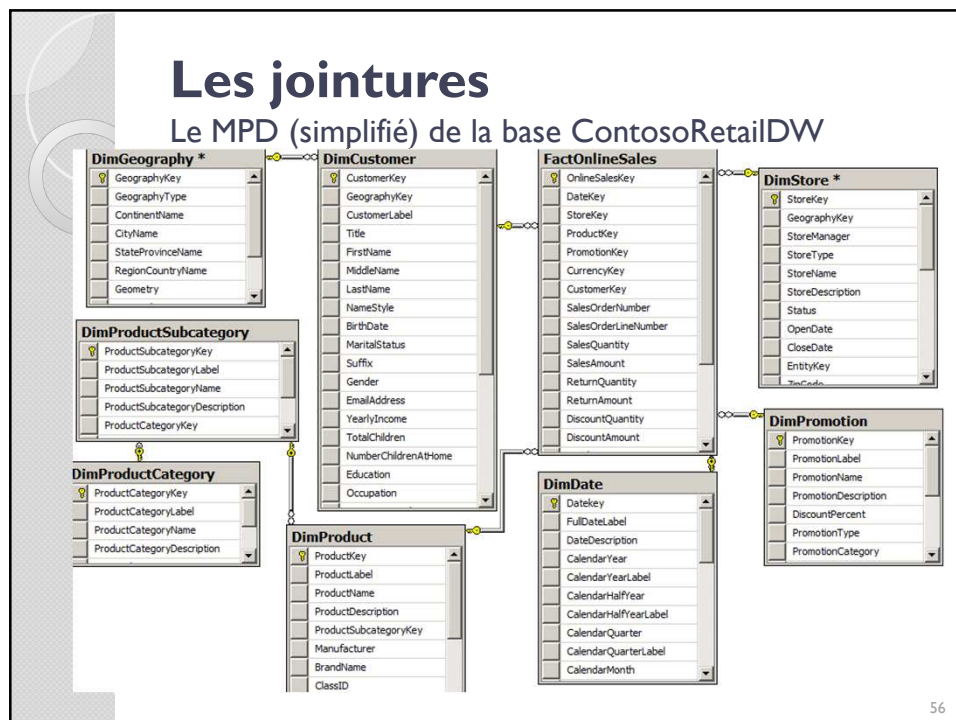
Lire un modèle relationnel (ou physique / MPD)



55

# Les jointures

Le MPD (simplifié) de la base ContosoRetailDW

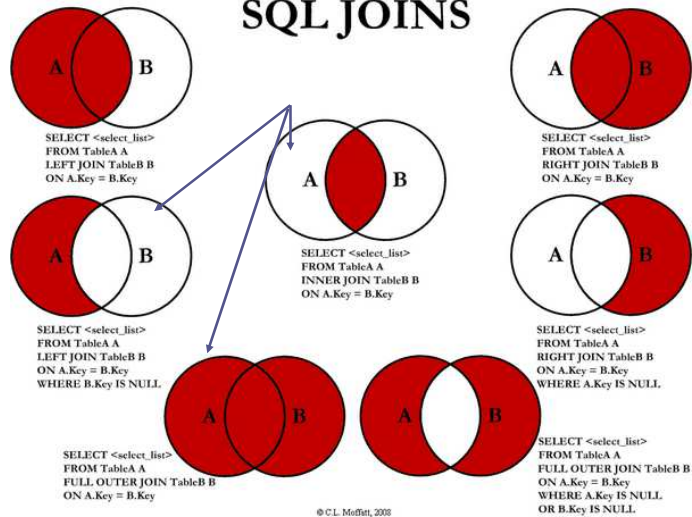


56

# Les jointures

Les jointures Vue d'ensemble

## SQL JOINS



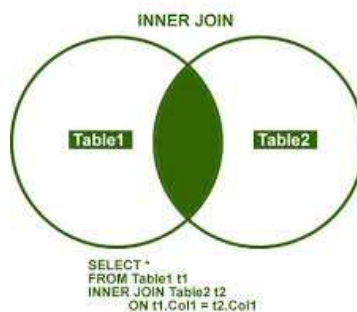
57

57

# 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>

58

58

# 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	D01	DEVELOPMENT CENTER
IF1000	QUERY SERVICES	C01	C01	INFORMATION CENTER
IF2000	USER EDUCATION	E01	E01	DEVELOPMENT CENTER
MA2100	WELD LINE AUTOMATION	D01	D01	DEVELOPMENT CENTER
PL2100	WELD LINE PLANNING	B01	B01	PLANNING

59

59

# Les jointures

## Renommer les tables (alias)

```
--DAT : DimDate
--FAC : FactonlineSales
--PROM : DimPromotion
--STO : DimStore
select
DAT.CalendarYear,
sto.StoreName,
FAC.SalesAmount
from FactSales FAC
inner join DimProduct PRO
on FAC.ProductKey = PRO.ProductKey
inner join DimPromotion PROM
on FAC.PromotionKey = PROM.PromotionKey
inner join DimDate DAT
on FAC.DateKey = DAT.Datekey
inner join DimStore STO
on FAC.StoreKey = STO.StoreKey
where
DAT.CalendarYear in (2008,2009,2007)
```

### Exercice :

- Dessiner le schéma avec les tables utilisées,
- Cocher clés primaires et étrangères,
- Lister champs utilisés par tables
- Dites-moi s'il y a une erreur dans le code

Client C	
<input checked="" type="checkbox"/>	IdClient (PK)
<input checked="" type="checkbox"/>	ProduitID (FK)
<input type="checkbox"/>	Nom

60

60

## 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'
```

61

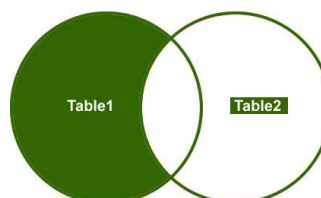
61

## 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>

62

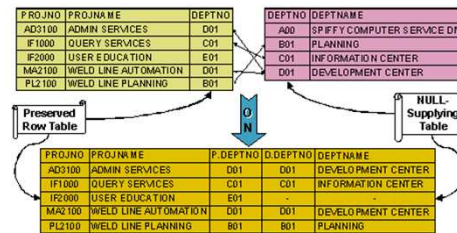
62

# 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
```



63

63

# 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
```

64

64



## Les jointures

« Old School » à proscrire dans la plupart des cas  
(même si les versions récentes des SGBD créent le même plan d'exécution et offrent les mêmes performances)

```
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'
```

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

65

65

## Les jointures

### Synthèse et Méthodologie

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

(A afficher et pour mes conditions) ?

FROM : Je fais mes jointures et je crée mes alias pour relier mes tables

(je peux dessiner un schéma lors de cette étape si besoin)

#### 2. 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

#### 3. 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.

#### 4. Comment je peux vérifier mon résultat ?

SELECT : mettre mes champs du where afin de vérifier que les critères fonctionnent

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

INNER JOIN ou LEFT JOIN ou RIGHT JOIN

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

66

66

## La commande SELECT :

### SELECT UNION / ALL / INTERSECT / EXCEPT

```
SELECT distinct City FROM Customers  
  
UNION  
  
SELECT distinct City FROM Suppliers  
ORDER BY City;  
  
USE AdventureWorks2012  
go  
  
SELECT CurrencyCode  
FROM [Sales].[CountryRegionCurrency]  
  
UNION  
  
SELECT CurrencyCode  
FROM [Sales].[Currency]  
ORDER BY CurrencyCode
```

67

67

## Langage SQL

### LES SOUS REQUÊTES

68

68

## 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

69

69

## Sous requêtes :

Dans la clause WHERE : In ou NOT IN (requête simples)  
L'inclusion et l'exclusion

```
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%');
```

70

70

## 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

71

71

## 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<sup>ère</sup> 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)

72

72

## 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';
```

73

73

## Sous requêtes :

Dans la clause FROM

```
-- Nbre de clients factures par année (SOUS REQUETE)
select
année,
count(*)
from
    (
        select distinct      -- DISTINCT s'applique aux 2 champs
        year(datekey) as année,
        customerkey
        from
        factonlinesales
    )
as t                          -- Obligatoire si sous-requete
group by année

--autre solution (PLUS EFFICACE)
select
year(datekey) as 'année',
count(distinct CustomerKey)
from FactOnlineSales
group by year(datekey)
```

74

74

## 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)

```

75

75

## Les sous-requêtes

### Synthèse et Méthodologie (I/2)

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

(A afficher et pour mes conditions) ?

FROM : Je fais mes jointures et je crée mes alias pour relier mes tables

(je peux dessiner un schéma lors de cette étape si besoin)

#### 2. 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

#### 3. 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.

#### 4. Comment je peux vérifier mon résultat ?

SELECT : mettre mes champs du where afin de vérifier que les critères fonctionnent

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

INNER JOIN ou LEFT JOIN ou RIGHT JOIN Ou SOUS-REQUETE

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

76

76

## Les sous-requêtes

### Synthèse et Méthodologie (2/2)

**6. Est-ce que j'ai besoin de faire un calcul (fonction d'agrégation : sum, count, max, min, average) sur une autre fonction d'agrégation ou sur un autre résultat ?**

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

**7. Est-ce que je souhaite avoir d'autres calculs (fonction d'agrégation) dans mon select qui soit un résultat général afin de comparer avec le résultat par enregistrement (client, commande, etc.) ?**

**Est-ce que je souhaite ramener un résultat d'une autre table ou d'un autre résultat dans ma requête afin d'avoir l'information sur mon extraction?**

Je vais devoir créer une requête imbriquée dans le SELECT afin de créer un nouveau champ qui aura pour chaque ligne la même valeur.

77

77

## Sous requêtes :

Règles à ne pas oublier !!!

- Toujours préciser les colonnes retourner par la recherche (le « select \* » est très gourmand)
- TOUJOURS utiliser le mot clé « JOIN (inner pour les select d'inclusion Multi-tables
- Choisir sous requêtes ou LEFT JOIN pour requêtes d'exclusion
- Ne joindre que les tables nécessaires
- Pour augmenter la lisibilité des requêtes, renommer vos tables (alias)
- Documenter la requête avant de l'écrire :
  - But
  - Auteur
  - Date

**ET ne pas oublier le TOP !!**

78

78

# Langage SQL

## LES REQUÊTES ACTIONS

79

79

### Requêtes « ACTION » :

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

80

80



## Requêtes « ACTION » :

INSERT : Ne renseigner que certains champs

use DISTRISYSDW

insert into DimSite  
(SiteCode,Site)

VALUES  
( 'D008','Marseille' )

81

81

## 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) insert T_titre
values ('M.', 'Monsieur'),          values ('M.', 'Monsieur'),
      ('Mlle.', 'Mademoiselle'),    ('Mlle.', 'Mademoiselle'),
      ('Mme.', 'Madame')             ('Mme.', 'Madame')
```

82

82

## 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';
```

83

83

## Requêtes « ACTION » :

Requête UPDATE : Modification de valeurs

A noter, pour spécifier en une seule fois plusieurs modifications, 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

84

84

## 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'Aller	Portblon	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

85

85

## 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

86

86

## Requêtes « ACTION » :

Requête UPDATE

```
UPDATE T_Produit
```

```
SET Produit_PU = Produit_PU * 1.1
```

Where

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

87

87

## 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/lentier 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.*

88

88

## 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.*

89

89

## 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

90

90

## 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.

91

91

## Langage SQL



## LES TRANSACTIONS

92

92

## 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.

93

## 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

94

## Implémentation SQL

### Transactions : Les propriétés ACID

- **Atomicité** : Cette propriété garantit qu'une série d'opérations (une "transaction") est exécutée soit en totalité soit pas du tout.
- **Cohérence** : Cette propriété garantit que toute modification fait évoluer la base d'un état cohérent à un autre état cohérent, en respectant l'intégralité des contraintes d'intégrité, y compris les contraintes référentielles.
- **Isolation** : Cette propriété garantie qu'aucune transaction en cours n'est affectée par une quelconque transaction non encore validée, même si certaines des opérations qui la composent sont déjà validées séparément.
- **Durabilité (rémanence)** : Cette propriété, qui ne peut jamais être absolue du fait des risques de destruction matérielle des supports d'information, garantit que les résultats d'une transaction une fois validée sont permanents, y compris en cas de destruction de parties du matériel non affectées au stockage des données de la base.

95

## Implémentation SQL

### Transactions : Verrouillage

L'outil le plus répandu pour sérialiser les transactions est basé sur l'utilisation de verrous (*lock*).

On impose que l'accès aux données se fassent de manière mutuellement exclusive.

Un verrou est posé sur chaque donnée accédée par une transaction afin d'empêcher les autres transactions d'y accéder.

96



## Implémentation SQL

### Transactions : Niveau d'isolation

```
USE DB_ISO_LEVEL
GO

SET TRANSACTION ISOLATION READ UNCOMMITTED
BEGIN TRANSACTION TRAN1

DECLARE @TOTAL INT

SELECT @TOTAL = SUM(COL)
FROM T_ISO

WAITFOR DELAY '00:00:20'

SELECT @TOTAL = @TOTAL - SUM(COL)
FROM T_ISO

SELECT @TOTAL AS TOTAL

COMMIT TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Read uncommitted :  
les données  
physiquement  
corrompues ne sont  
pas lues mais lecture  
non validée.

Il est donc nécessaire  
d'avoir **Read  
Committed** : niveau  
par défaut du moteur  
de base de données :  
Mise en place de  
verrous sur les  
lectures empêchant  
que la modification des  
lignes ne soient  
commises pendant la  
lecture.

97

## 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

98

## 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.

99

## 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
```

100

## 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
```

101

## 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.

102

## 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'

103

103

## 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'

104

104

## 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
```

105

105

## Implémentation SQL

### Gestion des erreurs : 1<sup>er</sup> 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
```

106

106

## 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
```

107

107

## 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 système (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;
```

108

108

## 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

109

109

## 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

110

110

## 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.

|||

111

## 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.

|||

112



## Implémentation SQL IF dans une procédure stockée

```
USE AdventureWorks2012;
GO

DECLARE @AvgWeight decimal(8,2), @BikeCount int -- ,@AvgWeight2 decimal(8,2)
IF
(SELECT COUNT(*) FROM Production.Product WHERE Name LIKE 'Touring-3000%' ) > 5
BEGIN
    SET @BikeCount =
        (SELECT COUNT(*)
         FROM Production.Product
         WHERE Name LIKE 'Touring-3000%'); -- je compte le nombre de produits
    SET @AvgWeight =
        (select avg(weight) -- je fais la moyenne sur mon top 5
         from
             (SELECT top 5 * -- j'affiche mes 5 produits par ordre de poids, du plus grand au plus petit
              FROM Production.Product
              WHERE Name LIKE 'Touring-3000%'
              order by Weight desc) as top_5);
    /*SET @AvgWeight2 =
        (select avg(weight)
         FROM Production.Product
         WHERE Name LIKE 'Touring-3000%') -- je fais la moyenne de tous les produits */

    PRINT 'There are ' + CAST(@BikeCount AS varchar(3)) + ' Touring-3000 bikes.'
    PRINT 'The average weight of the top 5 Touring-3000 bikes is ' + CAST(@AvgWeight AS varchar(8)) + '.';
    -- PRINT 'The average weight of all the Touring-3000 bikes is ' + CAST(@AvgWeight2 AS varchar(8)) + '.'
END
ELSE -- si moins de 5 produits Touring-3000%, alors je fais juste moyenne générale
BEGIN
    SET @AvgWeight =
        (SELECT AVG(Weight)
         FROM Production.Product
         WHERE Name LIKE 'Touring-3000%');
    PRINT 'Average weight of the Touring-3000 bikes is ' + CAST(@AvgWeight AS varchar(8)) + '.';
END ;
```

113

113

## Implémentation SQL déclencheurs (triggers)

- Un déclencheur est un type spécifique de procédure stockée qui n'est pas appelé directement par un utilisateur. Lorsque le déclencheur est créé, il est défini de façon à se déclencher lorsqu'un certain type de modification de données est effectué dans une table ou une colonne spécifique (After INSERT, Instead Of DELETE)
- On peut dire qu'il s'agit d'une procédure stockée événementielle.
- Il existe des déclencheurs de tables et de bases de données (DDL)

114

## Implémentation SQL

### déclencheurs (trigger DDL)

- Dans les [bases de données](#), lors de la mise à jour ou de la suppression d'une donnée, si un déclencheur existe, il peut lancer automatiquement une [procédure stockée](#), qui agit en parallèle sur la même donnée dans une table afférente.
- Il peut également remplir automatiquement une table historique, ou contrôler la validité de la données
- Cela permet d'automatiser certains traitements assurant la cohérence et l'intégrité de la base de données.
- **Aujourd'hui c'est souvent l'applicatif qui gère ces problématiques.**

115

## Implémentation SQL

### Sécurité (droits d'accès et rôles)

- Les risques propres à une source de données sont les suivants :
  - vol de données (perte de confidentialité)
  - altération de données (perte d'intégrité)
  - destruction de données (remise en cause de la continuité d'activité)
  - augmentation du niveau de privilèges d'un utilisateur d'une application (sécurité, espionnage)
  - ressources systèmes abusives (déni de service)

116

## Implémentation SQL

### Sécurité (droits d'accès et rôles)

- Contrôle des privilèges
- La principale question qui se pose lors du développement d'une application, c'est quelle stratégie adopter vis à vis des utilisateurs : contrôle de leurs droits d'accès par l'application ou par le SGBD ?
- Le créateur d'une table est propriétaire de cette table et obtient les droits d'accès de cette table.
- Le propriétaire de la table peut passer ses privilèges sélectivement à d'autres utilisateurs ou à tout le monde.

117

## ANNEXES

Sommaire / Plan détaillé du support de formation

Schémas des bases de données

Liens web utiles

118

118

## Sommaire / Plan détaillé (1/3)

<b>Objectifs de la formation</b>	<b>Page 4</b>
<b>Sommaire synthétique</b>	<b>Page 5</b>
<b>Introduction aux bases de données</b>	<b>Page 6</b>
Base données et présentation	Pages 8 / 9
Les Tables	Page 10
Les champs	Page 11
Les types de données	Page 12
<b>Extraire les données d'une table</b>	<b>Page 14</b>
Commande Select et syntaxe	Page 17 / 18
La Clause DISTINCT	Page 19
Select et règles d'écritures : erreurs type	Page 21
Les commentaires en SQL	Page 23
Clause Top	Page 25 / 26
Sensibilité à la casse	Page 27
Renommer les colonnes	Page 28
La Clause ORDER BY	Page 29
La clause WHERE : critères simples	Page 30
La clause WHERE : (not) IN	Page 31
La clause WHERE : (not) LIKE	Page 32
La clause WHERE : BETWEEN AND	Page 33
La clause WHERE : Synthèse	Page 34
La clause CASE	Page 35 - 38
<b>Calculs et fonctions intégrées</b>	<b>Page 39</b>
Champs calculés simples	Page 41
Les fonctions d'agrégation	Page 42
Group by : Regroupement et Agrégation	Page 43

119

119

## Sommaire / Plan détaillé (2/3)

Group by : Clause <u>where</u> et Clause <u>having</u>	Page 44
Group by ou La clause OVER	Page 45
Les clauses <u>Row NUMBER</u> et Rank	Page 46
Fonctions Date	Page 47 - 48
Clause <u>Where</u> intégrant une fonction	Page 49
Fonctions textes (chaînes de caractères)	Page 50
La fonction CAST et la conversion	Page 51
<b>Les jointures</b>	<b>Page 52</b>
Structure d'une table : les clés	Page 53
Rôle des clés	Page 54
Modèles relationnels (MPD)	Page 55 / 56
Les jointures : vue d'ensemble	Page 57
Les jointures : INNER JOIN	Pages 58 / 59
Renommer les tables	Page 60
Les jointures : LEFT JOIN	Pages 62 - 64
Jointures Old <u>School</u>	Page 65
Les jointures : Synthèse et Méthodologie	Page 66
Clauses UNION, INTERSECT, EXCEPT	Page 67
<b>Les sous-requêtes</b>	<b>Page 68</b>
Dans la clause <u>WHERE</u> ; IN ou NOT IN (requêtes simples)	Pages 70
Dans la clause WHERE : ALL / ANY / SOME	Page 71
Dans la clause <u>WHERE</u> ; EXISTS (requêtes corrélées)	Page 72
Sous-requêtes VS Jointures	Page 73
Sous-requêtes dans la clause FROM	Page 74
Sous-requêtes dans la clause SELECT	Page 75
Les sous-requêtes : Synthèse et Méthodologie	Page 76 / 77
Règles à ne jamais oublier	Page 78

120

120

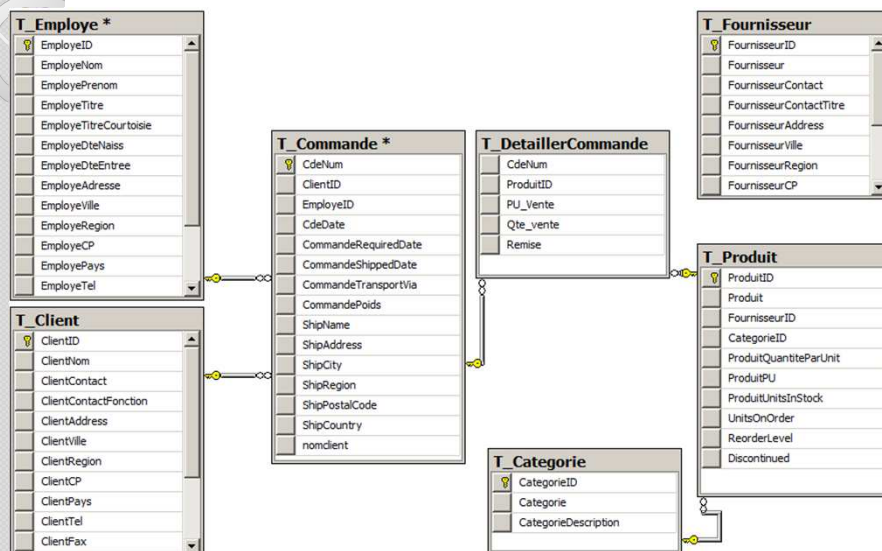
## Sommaire / Plan détaillé (3/3)

<b>Les requêtes actions</b>	<b>Page 79</b>
INSERT	Page 81
INSERT à partir d'une requête	Page 83
UPDATE	Page 84
CREATE TABLE	Page 88
DELETE	Page 90
TRUNCATE ET DROP	Page 91
<b>Les transactions</b>	<b>Page 92</b>
Définitions	Page 93
COMMIT et ROLLBACK	Page 94
Propriétés ACID	Page 95
Verrouillage	Page 96
Niveau d'isolation	Page 97
Vues	Page 98
CTE	Page 99
Procédures stockées	Page 102
BEGIN END	Page 105
Gestion des erreurs	Page 106
IF ELSEIF ELSE	Page 109
WHILE	Page 111
Déclencheurs	Page 114
Sécurité	Page 116
<b>Annexes</b>	<b>Page 118</b>
Sommaire / Plan détaillé du support de formation	Page 119
Schémas des bases de données	Page 122
Liens Utiles	Page 125

121

121

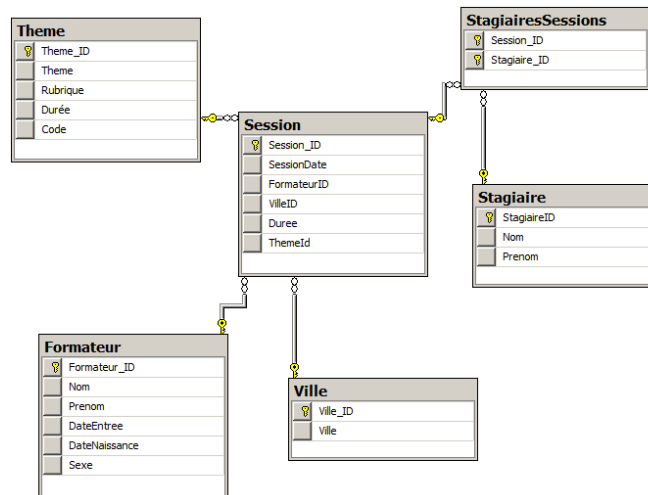
## Schémas des bases de données - Comptoir



122

122

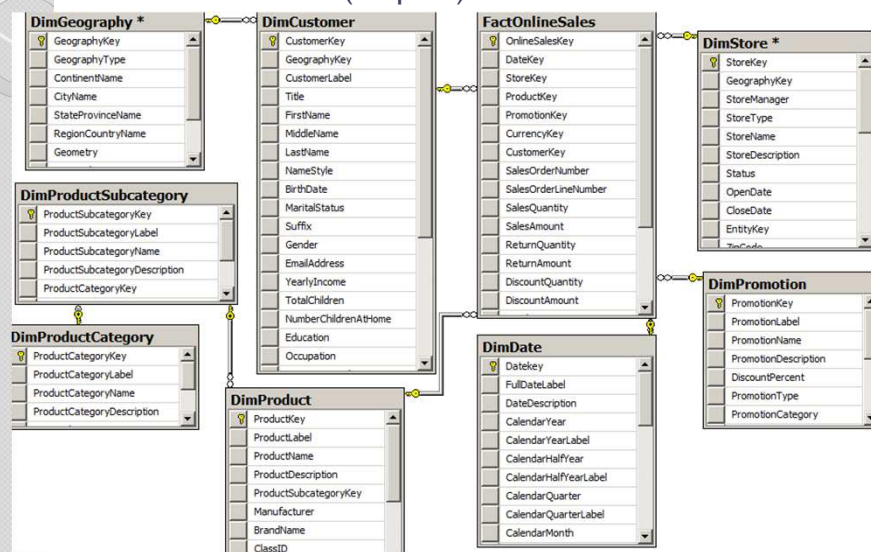
## Schémas des bases de données \_SQL\_SESSION\_SIF



123

123

## Schémas des bases de données ConsotoRetailDW (simplifié)



124

124

## 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](http://developpez.com/#apprendre-sql)

125

125

## MERCI POUR VOTRE ATTENTION

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

Bonne continuation



126

126