

## Chapitre 01: Rappel "SQL"

SQL 1. LDD = Langage de définition de données (creation, modification, suppression)

2. LMD = " manipulation "

Create

Alter

Delete

Drop

char(80) statique → T T O U V V

varc har (80) dynamique → T O T O

EXP:

BDD { clients (nom, adresse, val-compte)  
 CC (num\_cc, nom, marchandise, qte, date\_com)  
 Fournisseurs (nom, adresse, marchandise, prix)

\* ajouter le tel au Table de clients: On Oracle

Alter Table clients ADD (tel char(10));

\* modifier la taille du tel :

Alter Table client MODIFY (tel char(13));

- Pour avoir la forme de Alter

SQL > Help Alter table;

2

a) - Requête simple ⇒ Select + From

b) - Requête avec conditions ⇒ Select + From + Where

Conditions avec sous-requête b)

Conditions de comparaison

Opérateur: +

=, >, <, >=

<=, In, Not In,

IS, ISNULL,

Between,

Like, NOT like

Opérateur logique:

And,

OR

Conditions avec jointeur

Simple ... ④

Multiple (+ de 2 tables de jointure)

Auto-jointeur

externe

- Select Nom ADR From Fournisseur

where ADR is NOT NULL

and (Nom Like "D%" OR Nom like "?A%"),

%

premier

lettre est n'importe quoi

g) - Requête avec Groupement

a) - Nom, ADR client / client commandé des pommes

Distinct

Select client.Nom, ADR

From client, cc

where client.Nom = cc.Nom

AND Marchandise = "Pomme",

b) - Nom, ADR de client / client commandé Marchandise (Bouziad)

Select client.Nom, client.ADR

From client, cc, Fournisseur

where client.Nom = cc.Nom and cc.Marchandise = Fournisseur.Marchandise

and cc.Marchandise IN (Select Marchandise From Fournisseur

where Nom = 'Bouziad');

c) - Requête Ensemble : R = (Req<sub>1</sub>) UNION  
INTERSECT (Req<sub>2</sub>)  
MINUS

(Select Nom  
From Client) U (Select Nom  
From Fournisseur)

(H) Select Marchandise, Round(prix)  
From Fournisseur;

(D) Select AVG(prix)  
From Fournisseur;

d) - Requête avec opérateur Exists / NOT Exists

e) - Requête avec TRI : Select + From + [where] + order by

Select Nom, ADR

From Client

ORDER BY ADR, Nom ASC;

(H) - de bases : numérique, conversion, chaîne de car, date  
dans Select / dans where → s'applique sur une valeur d'une colonne

f) - Requête avec Fonction

(D) - Agrégat → s'applique sur un ens de vals et de retour  
func. COUNT / MIN / MAX du Groupby

## Chapitre 2: Contraintes d'intégrité

### Introduction:

les données stockées dans BDD doivent être conforme à la réalité, on parlera de cohérence de BDD ou d'intégrité de la BDD. Dans le contexte de BDD, la cohérence est traduit souvent des règles appelées contrainte d'intégrité.

L'intégrité de BDD couvre 2 aspects:

- \* **l'intégrité symétrique** (ex: chaque personne doit avoir un age)
- \* **l'intégrité externe** (ex: concerne la reprise après une panne de BDD)

→ les actions qui peuvent déclencher le contrôle d'intégrité sont, **insert**, **update**, **delete**

Syntaxe: Create Table <nom\_Table>

<c-1> <type1> [cI-c1],  
<c-2> <type2> [cI-c2],  
|  
<c-n> <type n> [cI-cn],

Niv-col

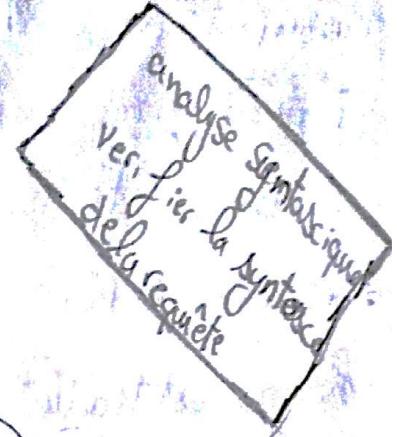
Niv-Table { [cI-Table] } ;

[cI-ci]: [constraint <nom\_cI>]

- NotNull / null
- unique
- primary key (unique + not null)
- check(dition)
- references <table> (colonne)

# SQL - Procedural

Schema BDD: → Pilote (brevet, nom, comp, nbtrivol)  
 ↳ Compagnie (comp, nfcie, rue, ville, nomComp)



## Partie 1 : Base

① SQL Procedural = SQL + Σ instructions

② Généralités:

2.1. Bloc d'instructions  $\Leftrightarrow \{ \text{ et } \}$

float  
Number (4,2)

Begin et end

2.2. Portée des objets: les objets utilisent seulement dans les blocs non dans un autre bloc

2.3. Casse: non différence entre majuscule et minuscule

Visibilité: utiliser des normes de prog pour un propre programme

2.4. Identificateurs: nom d'un var, Tab, fonction, ...

2.5. Commentaires: -- une ligne de commentaire

/\* = \*/ → plusieurs lignes

③ Variables = la Syntaxe:

Scalarie

Externes

@**DECLARE** nomvar1, nomvar2, ..., nomvarn type;

EXP: **DECLARE** v-datenissance **DEFAULT** Date;  
 variable (norme)

DEFAULT "20/10/2018"

DECLARE v-datenissance **DEFAULT** Sysdate();  
 Date date d'aujourd'hui

(b) Affectation de variables: en C: v int; v = 5;

SET Variable := expression

Select 5; → 5

Exp: SET v := 2;

Select 2 INTO v;

### c) Resolution noms:

- Declare v-nom Varchar(20) default "Bougaré"; } fault, il faut toujours distinguer entre le nom de la colonne et le nom de la variable (changer le nom de l'une des deux)
- Delete From Pilote Where nom = v-nom;

### d) Les opérateurs: m que SQL

e) Variables de Session: SET @varSession = expression;

```
> SET @varSession1 = 1;
> Select @varSession1;
> 1
```

PP.

- decimal = Réel
- number = entier

on peut affecter directement sans déclarer la var

Exp: Page 4: SET @vs-num = 'PL-4'; declare v-nom Varchar(20);  
declare v-nbtval Decimal(7,2);

Select nom, nbtval } procedure

From Pilote

where brevet = 'PL-4';

↓  
@vs-num

INTO v-nom, v-nbtval

Insert INTO Pilote values  
( 'Boutef', 'AH',

### f) Conventions/Normes de développement:

Variables → V-nomVar

Constantes → C-nomConstante

Var Session → VS-nomVarSession

### g) Expl de bloc:

#### H. Structure de contrôles : les if - Case

H-1). Conditionnelle if / if . else

corps

H-2). Itération

H-2.a). while: while condition Do ..... End while ;

Exp: La somme de 100 entiers

H-2.b). Repeat: Repeat ..... until <cond> End repeat;

Corps

H-2-c) Loop: loop; ... END Loop; (boucle infinie)

Etg (obligatoire) corps

boucle 1: Loop

obligatoire Set v\_Somme := v\_Somme + v\_entier;

Set v\_entier := v\_entier + 1; **Iterate** (recommencer)

If v\_entier > 100 Then Leave boucle 1;

END If; v\_entier <= 100

END Loop;

pour sortir de boucle infinie

Ex2

Leave boucle 1;

Page 8

## I)- Interaction avec la Base

I-1) Extraire des données : - Select .... INTO var .... From .... Where ....

I-2). Manipulation des données

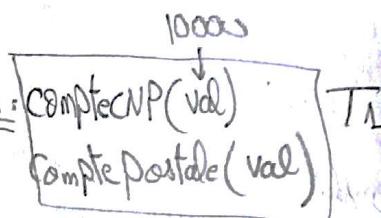
I-2-1) Insertion  $\rightarrow$  Insert into <Table> values (10, 'K', 'AAB');

I-2-2) Update

I-2-3) Delete

atomicité  
non divisibilité

J) Transaction (page 10).



(var1, var2, AAB);

Transfert(500€)  $\Rightarrow$  Début

update compteCNP(-500)  
update comptePostale(+500)

Fin

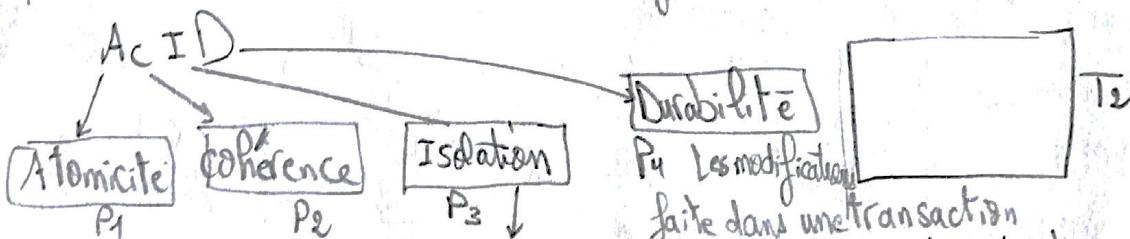
$\Rightarrow$  CompteCNP(9500)  
comptePostale(0)

Transaction

coupure de courant

résultat  
incohérente

- Pour assurer des bonnes résultats il faut que l'transaction a une propriété



Durabilité

Pu les modifications

faite dans une transaction

mettre la Selection (select) iff faut rester (stocker)

avant / après la transaction et non dans la transaction (etats initial et final etats intermédiaires)

(6)

Coursure : mémoiser les résultats d'une requête dans une zone comme un tableau accessible (liste) au bien

qui a un nom

open Cours1;  $\Rightarrow$  Cours1

ID	nbr	comp
1	100	'AF'
2	50	'Bc'
:	:	
10	25	'DE'

Fetch Cours1  $\Rightarrow$  afficher (chercher) les lignes avec 3 colonnes

INTO v-Id, v-nbr, v-comp;

a chaque fois il faut faire un Fetch pour afficher la Totalité de coursure (Cours1)

Declare v-Id--;

;

Declare v-nbr---;

Declare v-comp---;

1<sup>er</sup> Fetch: 1 100 AF

2<sup>eme</sup> Fetch: 2 50 Bc

! 10<sup>ieme</sup> Fetch: 10 25 DE

11<sup>ieme</sup> Fetch: Erreur

les fcts sur les Coursure:

- \* ouverture : exécuter Requête + remplissage
- \* Fermeture : libérer la zone mémoire
- \* Déclaration : Declare nom type Pour requête
- \* accès : afficher le Coursure FOR contenu de

Exp: ~~la boucle~~ Declare v-cpt  
Declare v-Som decimal (7, 2) Default 0;  
Declare v-Fin boolean Default 0;

Declare v-Id---;

int Default 0;

Fetch  $\rightarrow$  while (NOT v-Fin) do

Fetch Curs1 INTO v-Id, v-nbr, v-comp;

SSET v-Som := v-Som + v-nbr;

(@) SSET v-cpt := v-cpt + 1;

Declare Curs1 Coursur For  
Select ID, nbr, comp From Pilote  
where comp = 'AF';

Exit while;

$\Rightarrow$  dans 11<sup>ieme</sup> fetch : il y a erreur donc l'exception déclanche (NotFound) alors

solution : mettre (@) avant le Fetch

Select v-Som  
v-cpt;

Declare Continue

Handler For NotFound

Set v-Fin := 1;

4 - Triggers : déclencheur, on associe à l'evenement . l'événement . le prog

Syntaxe: PLT { Before | After }  
Create Trigger <nom déclencheur> { Delete / Insert / Update } <evenement>  
ON <nomTable>  
FOR EACH Row { Begin  
|  
End; / Instruction } ;

s'applique sur les règles de gestion

qualification (brevet, typeA, DateExp)

Exemple: Insert into qualification values ("PL1", "B720", "01/01/99")

Select count(\*) From qualification  
where brevet = "PL1";

Create trigger Ins.PL.RG1

Before Insert On qualification

For each Row

Begin

Declare v\_nbr int Default 0;  
Select count(\*) from qualification

into v\_nbr;

where brevet = New.brevet

if (v\_nbr = 3) Then /\* bloquer / annuler insertion  
Select 1/0; \*/ Select "Erreur";  
Insert into qualification values (New);  
End;

End;

Ex08

Ex07: RG1-T

Create Trigger T-Del-QL

AFTER Delete On qualification  
<sup>Insert</sup>

For each Row

Begin

update filiale

Set nb\_Qlt = nb\_Qlt + 1

where brevet = Del.brevet

New

End.

Exemple:

Before insert on qualification

For each Row

Begin

Declare v\_nbql int default 0;

Select count(\*) into v\_nbql

where brevet = New.brevet

if (v\_nbql >= 3) then Select "Erreur: RG F8 pas respecté";  
insert into Table values (Null);

End;

End;

Exo: update brevet table Qual

Create Trigger up ql.RG Exo

After update on qualification

For each Row

Begin

update Pilote

Set nbql = nbql - 1

where brevet = old.brevet

update Pilote

set nbql = nbql + 1  
where brevet = New.brevet

End;

Exo:

Create Trigger insert-Grade

Before, Insert on Pilote

For each Row

Begin

### Ex 10:

Create Trigger TrigInsGrade before Insert update

On Table

For each row

Begin

if (new.nbhval  $\geq$  4000 or new.nbhval  $\leq$  100) and new.grade = 'Comd'

Set new.grade := old.grade

end

if (new.nbhval  $\leq$  100 or new.nbhval  $\geq$  1000) and new.grade = 'Cop'

Set new.grade := old.grade

end

if (new.nbhval  $<$  3000) and new.grade = 'Ins'

Set new.grade := old.grade

end

end;