

# Le langage SQL

## Histoire

### □ SQL: Structured Query Language

- Version commerciale du langage SEQUEL issue du prototype System R développé par IBM San José en 1975
- Adopté par la plupart des SGBD relationnels
- Normalisé
  - SQL2: relationnel, 1992
  - SQL3: objet-relationnel, 1998

## Le langage SQL

---

### Définition des données

*Conventions méta syntaxiques*

- $A ::= B \equiv A$  est redéfini par B
  - $A | B \equiv A$  ou B
  - $[A] \equiv A$  est optionnel
  - $A^+ \equiv A[, \dots, A]$  liste de un ou plusieurs A
- ◆ Les symboles  $::=$   $|$   $[]$   $()^+$  ne figurent jamais dans les expressions SQL, ils permettent uniquement de spécifier sa syntaxe
- ◆ Les parenthèses font partie de la syntaxe si elles figurent en noir

### Création de domaine et de table

## CREATE DOMAIN

**domaine ::=** CREATE DOMAIN *nom type* [*valeur\_par\_défaut*];

**type ::=**  
| CHAR [(longueur)]  
| VARCHAR [(longueur)]  
| INTEGER  
| FLOAT  
| DATE  
| ...

**valeur\_par\_défaut ::=**  
| DEFAULT *constante*  
| USER  
| NULL  
| CURRENT\_DATE  
| CURRENT\_TIME  
| ...

## CREATE DOMAIN

### □ Exemples

**CREATE DOMAIN d\_numero\_client INTEGER;**

**CREATE DOMAIN d\_nom CHAR (30);**

**CREATE DOMAIN d\_prenom CHAR (40);**

**CREATE DOMAIN d\_adresse VARCHAR (100);**

**CREATE DOMAIN d\_ville CHAR (30) DEFAULT 'PARIS';**

**CREATE DOMAIN d\_code\_postal CHAR(5) DEFAULT '75000';**

La créations de domaines n'est pas obligatoire

## CREATE TABLE

En SQL une relation est une table

**table** ::= CREATE TABLE **nom** (**définition\_colonne**<sup>+</sup> [**contrainte\_table**<sup>+</sup>]);

**définition\_colonne** ::= **nom** type  
| **domaine**  
| [**contrainte\_colonne**]  
| [**valeur\_par\_défaut**]

En SQL un attribut est une colonne

**contrainte\_colonne** ::=  
PRIMARY KEY  
| NOT NULL  
| UNIQUE  
| CHECK (**condition**)  
| REFERENCES **nom\_table** (**nom\_colonne**)

## CREATE TABLE (fin)

**contrainte\_table** ::=  
[CONSTRAINT **nom**]  
| **type\_contrainte\_table**  
| **mode\_contrainte**

**type\_contrainte\_table** ::=  
PRIMARY KEY (**nom\_colonne**<sup>+</sup>)  
| NOT NULL (**nom\_colonne**<sup>+</sup>)  
| UNIQUE (**nom\_colonne**<sup>+</sup>)  
| CHECK (**condition**)  
| FOREIGN KEY (**nom\_colonne**<sup>+</sup>)  
REFERENCES **nom\_table** (**nom\_colonne**<sup>+</sup>)

**mode\_contrainte** ::= [NOT] DEFERRABLE

## CREATE TABLE (exemples)

### ❑ Exemple 1

```
CREATE TABLE t_client (  
    n_client d_numero_client,  
    nom d_nom NOT NULL,  
    prenom d_prenom,  
    adresse d_adresse,  
    ville d_ville,  
    code_postal d_code_postal  
);
```

## CREATE TABLE (exemple)

### ❑ Exemple 2

#### ■ Définition de la clé primaire par une contrainte de colonne

```
CREATE TABLE t_client (  
    n_client d_numero_client NOT NULL PRIMARY KEY,  
    nom d_nom NOT NULL,  
    prenom d_prenom,  
    adresse d_adresse,  
    ville d_ville,  
    code_postal d_code_postal  
);
```



**Possible uniquement si la clé primaire est constituée d'un seul attribut**

## CREATE TABLE (exemple)

### ❑ Exemple 3

#### ■ Définition de la clé primaire par une contrainte de table

```
CREATE TABLE parcours_prevu (  
    numero INTEGER NOT NULL,  
    periodicité CHAR (7),  
    numero_vol INTEGER NOT NULL,  
    PRIMARY KEY (numero, numero_vol)  
);
```



**Obligatoire** lorsque la clé primaire est constituée de **plusieurs attributs**

Possible aussi lorsque la clé primaire est constituée d'un seul attribut

## CREATE TABLE (exemple)

### ❑ Exemple 4

#### ■ Définition d'une clé étrangère par une contrainte de table

##### ❑ Nommage de la contrainte de clé (non obligatoire)

```
CREATE TABLE parcours_prevu (  
    numero INTEGER NOT NULL,  
    periodicité CHAR (7),  
    numero_vol INTEGER NOT NULL,  
    CONSTRAINT t_parcours_prevu_pk PRIMARY KEY (numero,  
                                                numero_vol)  
);  
PRIMARY KEY (numero, numero_vol)  
peut suffire
```

## CREATE TABLE (exemple)

### ❑ Exemple 5

- Définition d'une clé étrangère par une contrainte de table
  - ❑ Nommage de la contrainte de clé (non obligatoire)

```
CREATE TABLE parcours_effectif (  
    numero INTEGER NOT NULL,  
    date DATE NOT NULL,  
    numero_vol INTEGER NOT NULL,  
    PRIMARY KEY (numero, date, numero_vol),  
    CONSTRAINT t_parcours_effectif_fk FOREIGN KEY (numero,  
        numero_vol) references parcours_prevu (numero, numero_vol)  
);  
FOREIGN KEY (numero, numero_vol) references parcours_prevu (numero,  
    numero_vol)  
peut suffire
```

Modification, suppression  
de table et domaine

## ALTER / DROP

---

```
ALTER DOMAIN nom_domaine type ;
```

```
ALTER TABLE nom_table modification_table ;
```

```
modification_table ::=
```

```
    ADD COLUMN définition_colonne
```

```
    | ADD CONSTRAINT contrainte_table
```

```
    | ALTER définition_colonne
```

```
    | DROP COLUMN nom_colonne
```

```
    | DROP CONSTRAINT nom_contrainte
```

```
DROP DOMAIN nom_domaine ;
```

```
DROP TABLE nom_table ;
```

## ALTER/DROP (exemple)

---

```
ALTER TABLE parcours_effectif ADD COLUMN meteo CHAR (20);
```

```
ALTER TABLE parcours_prevu ALTER periodicite CHAR (30);
```

```
ALTER TABLE t_client DROP COLUMN ville;
```

```
ALTER TABLE parcours_effectif DROP CONSTRAINT t_parcours_effectif_fk;
```

```
DROP TABLE t_client;
```



# Le langage SQL

## Manipulation des données

## INTERROGATION: « SFW »

*requête ::=*

**SELECT** [DISTINCT] *projection*

[FROM (nom\_table [[ AS ] nom ])\* | (*requête* AS nom)]

[WHERE *condition*]

[GROUP BY nom\_colonne\*]

[HAVING *condition*]

[ORDER BY nom\_colonne\* | rang\_colonne\* [ASC||DESC] ;

| *requête* UNION | INTERSECT | EXCEPT *requête*

|(*requête*)

## INTERROGATION (suite)

**projection** ::= \* | nom\_table.\* | (**terme\_projection** [AS **nom**])<sup>+</sup>  
**terme\_projection** ::= **expression** | **agrégation**  
**expression** ::= **valeur** | nom\_colonne | **expression\_arithmétique** ...  
**valeur** ::= nombre | chaîne\_caractères  
**nom\_colonne** ::= [nom\_table.]nom  
**agrégation**  
::= COUNT(\*) | **opérateur\_agrégation**([DISTINCT] **expression**)  
**opérateur\_agrégation** ::= COUNT | SUM | AVG | MAX | MIN

## INTERROGATION (suite)

**condition** ::= **condition\_élémentaire**  
| NOT **condition**  
| **condition** AND | OR **condition**  
| (**condition**)  
**condition\_élémentaire** ::=  
  **condition\_like**  
  **condition\_null**  
  **comparaison**  
  **condition\_intervalle**  
  **condition\_in**  
  **condition\_exists**  
**condition\_like** ::= **expression** [NOT] LIKE **modèle\_de\_chaîne**  
**condition\_null** ::= nom\_colonne IS [NOT] NULL

## INTERROGATION (fin)

**comparaison ::=**

**terme\_comparaison comparateur terme\_comparaison**

**| expression comparateur ALL | SOME sélection\_mono\_colonne**

**terme\_de\_comparaison ::= expression | requête**

**comparateur ::= = | <> | > | < | <= | >=**

**condition\_intervalle ::= expression BETWEEN expression AND expression**

**condition\_in ::= expression [NOT] IN (requête)**

**condition\_exists ::= [NOT] EXISTS (requête)**

## INTERROGATION

### ❑ Equivalence algébrique

**SELECT DISTINCT  $A_1, A_2, \dots, A_n$   
FROM  $R_1, R_2, \dots, R_p$   
WHERE P**

## INTERROGATION

### □ Equivalence algébrique

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P (R_1 \times R_2 \times \dots \times R_p))$$


**SELECT DISTINCT  $A_1, A_2, \dots, A_n$**   
**FROM  $R_1, R_2, \dots, R_p$**   
**WHERE P**

**Pas d'élimination des doubles sans la clause  
DISTINCT !!**

## INTERROGATION (exemples)

**SELECT nom, n\_client**  
**FROM t\_client**  
**WHERE ville = 'PARIS'**

## INTERROGATION (exemples)

---

```
SELECT nom, n_client  
FROM t_client  
WHERE ville = 'PARIS'
```



$\Pi_{\text{nom, n\_client}} (\sigma_{\text{ville} = \text{'PARIS'}} (\text{t\_client}))$

## INTERROGATION (exemples)

---

```
SELECT x.nom, y.solde  
FROM t_client x, t_compte_crt y  
WHERE x.n_client=y.n_client AND y.solde > 2500;
```

## INTERROGATION (exemples)

```
SELECT x.nom, y.solde
FROM t_client x, t_compte_crt y
WHERE x.n_client=y.n_client AND y.solde > 2500;
```



$\Pi_{x.nom, y.solde} (\sigma_{y.solde > 2500} (t\_compte\_crt) \bowtie_{x.n\_client=y.n\_client} t\_client)$

## INTERROGATION

### □ Requêtes imbriquées


```
SELECT ...
FROM ...
WHERE [expression] opérateur (SELECT ...
                                FROM ...
                                WHERE ... )
```

requête englobante

requête imbriquée(sous requête)


## INTERROGATION

### □ Requêtes imbriquées

- *expression* [NOT] IN (*sous requête*)
  -  *expression* doit être **union-compatible** avec le **schéma** du résultat de *sous requête*.
  - Si *expression* **appartient au résultat** de *sous requête* alors **IN** retourne **VRAI** et **NOT IN** retourne **FAUX**
  - Si *expression* **n'appartient pas au résultat** de *sous requête* alors **IN** retourne **FAUX** et **NOT IN** retourne **VRAI**
- [NOT] EXISTS (*sous requête*)
  - Si le résultat de *sous requête* **contient au moins un tuple** alors **EXISTS** retourne **VRAI** et **NOT EXISTS** retourne **FAUX**
  - Si le résultat de *sous requête* **ne contient aucun tuple** alors **EXISTS** retourne **FAUX** et **NOT EXISTS** retourne **VRAI**

## INTERROGATION

### □ Requêtes imbriquées

- *expression* **comparateur ANY/SOME** (*sous requête*)
  - le schéma du résultat de *sous requête* **ne doit contenir qu'une et une seule colonne** 
  - *expression* est comparée aux tuples du résultat de *sous requête*.
  - **ANY** retourne **VRAI** si **une comparaison** de *expression* avec un des tuples du résultat de *sous requête* retourne **VRAI**
  - **ANY** retourne **FAUX** si **toutes les comparaisons** de *expression* avec tous les tuples du résultat de *sous requête* retournent **FAUX**
  - **SOME** est un synonyme de **ANY**

## INTERROGATION

### ☐ Requête imbriquées

- *expression* comparateur **ALL** (*sous requête*)



- le schéma du résultat de *sous requête* **ne doit contenir qu'une et une seule colonne**
- *expression* est comparée à **tous les tuples** du résultat de *sous requête*.
- **ALL** retourne **VRAI** si **toutes les comparaisons** de *expression* avec tous les tuples du résultat de *sous requête* **retournent VRAI**
- **ALL** retourne **FAUX** si **une des comparaisons** de *expression* avec un des tuples du résultat de *sous requête* **retourne FAUX**

## INTERROGATION

### ☐ Requête imbriquée indépendante

- Aucun attribut de la requête englobante n'est utilisé dans la sous requête
- ☐ La sous requête est dite **indépendante**



## INTERROGATION (exemples)

### ❑ Requêtes imbriquées indépendantes

t\_compte\_crt (n\_client , n\_compte, solde)  
t\_client (n\_client , nom, prénom, adresse, ville, code\_postal)

```
SELECT nom, n_client
FROM t_client
WHERE n_client IN (SELECT n_client
                  FROM t_compte_crt)

AND n_client NOT IN (SELECT n_client
                   FROM t_compte_ep) ;
```

requête englobante

requête imbriquée 1

requête imbriquée 2

Le langage SQL

© Michel Soto 33/62

## INTERROGATION

### ❑ Requêtes imbriquées paramétrées

- Un ou plusieurs attributs de la requête englobante sont utilisés dans une sous requête
- ❑ La sous requête est dite **paramétrée**

Le langage SQL

© Michel Soto 34/62

## INTERROGATION (exemples)

### ❑ Requêtes imbriquées paramétrées

```
SELECT nom, n_client
FROM t_client x
WHERE EXISTS (SELECT *
               FROM t_compte_crt
               WHERE x.n_client=n_client)
AND NOT EXISTS (SELECT *
                FROM t_compte_ep
                WHERE x.n_client=n_client);
```

paramètre

requête imbriquée 1

requête imbriquée 2

requête englobante

## INTERROGATION (exemples)

### ❑ Requêtes imbriquées paramétrées

```
SELECT x.nom, x.n_client, x.solde
FROM t_compte_crt x
WHERE x.solde >= ALL (SELECT solde
                     FROM t_compte_crt
                     WHERE x.n_client=n_client);
```

## INTERROGATION (exemples)

### □ Ordonnancement du résultat

```
SELECT nom, prenom  
FROM t_client  
WHERE ville = 'Paris'  
ORDER BY nom, prenom;
```



```
SELECT nom, prenom  
FROM t_client  
WHERE ville = 'Paris'  
ORDER BY 1, 2;
```

## INTERROGATION (exemples)

### □ Agrégats: COUNT, AVG, SUM, MIN, MAX

```
SELECT COUNT (*)  
FROM t_client ;
```

10	NOM1	Prénom1	A1	Paris	75000
34	NOM3	Prénom3	A3	Lyon	69000
42	NOM4	Prénom4	A4	Paris	75000
37	NOM5	Prénom5	A5	Lyon	69000
37	NOM6	Prénom6	A6	Nîmes	30000
22	NOM2	Prénom2	A2	Paris	75000

count (\*)

-----  
6

## INTERROGATION (exemples)

### ❑ Agrégats: COUNT, AVG, SUM, MIN, MAX

```
SELECT COUNT (*)  
FROM t_client  
WHERE ville = 'Paris';
```

10	NOM1	Prénom1	A1	Paris	75000
34	NOM3	Prénom3	A3	Lyon	69000
42	NOM4	Prénom4	A4	Paris	75000
37	NOM5	Prénom5	A5	Lyon	69000
37	NOM6	Prénom6	A6	Nimes	30000
22	NOM2	Prénom2	A2	Paris	75000

count (\*)

3

## INTERROGATION (exemples)

### ❑ Agrégats: COUNT, AVG, SUM, MIN, MAX

```
SELECT COUNT (ville)  
FROM t_client;
```

10	NOM1	Prénom1	A1	Paris	75000
34	NOM3	Prénom3	A3	Lyon	69000
42	NOM4	Prénom4	A4	Paris	75000
37	NOM5	Prénom5	A5	Lyon	69000
37	NOM6	Prénom6	A6	Nimes	30000
22	NOM2	Prénom2	A2	Paris	75000

count(ville)

6



## INTERROGATION (exemples)

### □ Agrégats: COUNT, AVG, SUM, MIN, MAX

```
SELECT COUNT (DISTINCT ville)  
FROM t_client;
```

10	NOM1	Prénom1	A1	Paris	75000
34	NOM3	Prénom3	A3	Lyon	69000
42	NOM4	Prénom4	A4	Paris	75000
37	NOM5	Prénom5	A5	Lyon	69000
37	NOM6	Prénom6	A6	Nîmes	30000
22	NOM2	Prénom2	A2	Paris	75000

```
count(distinct ville)  
-----  
3
```

## INTERROGATION (exemples)

### □ Agrégats: COUNT, AVG, SUM, MIN, MAX

```
SELECT AVG (solde)  
FROM t_compte_crt;
```

```
SELECT MAX (solde)  
FROM t_compte_crt;
```

```
SELECT MIN (solde)  
FROM t_compte_crt;
```

```
SELECT SUM (solde)  
FROM t_compte_crt;
```

## INTERROGATION (exemples)

### ☐ Regroupement ♦ *Modifie la portée du calcul pour les agrégats*

```
SELECT ville, COUNT (*)  
FROM t_client  
GROUP BY ville;
```

10	NOM1	Prénom1	A1	Paris	75000
34	NOM3	Prénom3	A3	Lyon	69000
42	NOM4	Prénom4	A4	Paris	75000
37	NOM5	Prénom5	A5	Lyon	69000
37	NOM6	Prénom6	A6	Nimes	30000
22	NOM2	Prénom2	A2	Paris	75000

ville	count (*)
Lyon	2
Nimes	1
Paris	3

## INTERROGATION (exemples)

### ☐ Regroupement ♦ *Modifie la portée du calcul pour les agrégats*

```
SELECT ville, COUNT (*)  
FROM t_client  
GROUP BY ville  
HAVING COUNT (*) >= 2;
```

10	NOM1	Prénom1	A1	Paris	75000
34	NOM3	Prénom3	A3	Lyon	69000
42	NOM4	Prénom4	A4	Paris	75000
37	NOM5	Prénom5	A5	Lyon	69000
37	NOM6	Prénom6	A6	Nimes	30000
22	NOM2	Prénom2	A2	Paris	75000

ville	count (*)
Lyon	2
Paris	3

## INTERROGATION (exemples)

### ☐ Regroupement *♦ Modifie la portée du calcul pour les agrégats*

```
SELECT x.ville, y.agence, AVG (x.solde)
FROM t_client x, t_compte_crt y,
WHERE x.n_client=y.n_client
GROUP BY x.ville , y.agence,;
```

## INTERROGATION (exemples)

### ☐ Puissance du langage

```
SELECT COUNT (*), x.nom, x.prenom
FROM t_client x, t_client y
WHERE (x.nom=y.nom
      AND x.prenom >= y.prenom)
      OR (x.nom > y.nom)
GROUP BY x.nom, x.prenom
ORDER by 1;
```

?

## Mise à jour d'une table

## INSERT: ajout de tuples

```
INSERT INTO nom_table [(nom_colonne)*]  
VALUES (valeur* | requête ;
```

```
valeur ::=  
    expression  
    | NULL  
    | USER  
    | CURRENT_DATE  
    | CURRENT_TIME  
    | CURRENT_TIMESTAMP
```



## INSERT: exemples

**t\_client (n\_client , nom, prénom, adresse, ville, code\_postal)**

❑ **INSERT INTO t\_client (n\_client, adresse, nom, prénom, ville, code\_postal)**  
**VALUES (10, '5, rue Hoch', 'Dupont', 'Jean', 'Paris', NULL);**

- **n\_client :** 10
- **nom :** 'Dupont'
- **prénom :** 'Jean'
- **adresse :** '5, rue Hoch'
- **ville :** 'Paris'
- **code\_postal :** NULL

## INSERT: exemples

**t\_client (n\_client , nom, prénom, adresse, ville, code\_postal)**

❑ ~~**INSERT INTO t\_client**~~  
~~**VALUES (10, '5, rue Hoch', 'Dupont', 'Jean', 'Paris', NULL);**~~

- **n\_client =** 10
- **nom =** '5, rue Hoch'
- **prénom =** 'Dupont'
- **adresse =** 'Jean'
- **ville =** 'Paris'
- **code\_postal =** NULL

## INSERT: exemples

**t\_client (n\_client, nom, prénom, adresse, ville, code\_postal)**

```
❑ INSERT INTO t_client  
VALUES (10, 'Dupont', 'Jean', '5, rue Hoch', 'Paris', NULL);
```

- n\_client = 10
- nom = 'Dupont'
- prénom = 'Jean'
- adresse = '5, rue Hoch'
- ville = 'Paris'
- code\_postal = NULL

## INSERT: exemples

**t\_bon\_client (num\_client, solde\_client)**

```
❑ INSERT INTO t_bon_client (num_client, solde_client)  
select n_client, solde  
from t_compte_crt  
where solde > 15 000;
```



- **Les schemas de t\_bon\_client et le schéma du résultat de la requête doivent être union-compatibles.**

## UPDATE: mise à jour de tuples

---

```
UPDATE nom_table  
SET (nom_colonne = expression)+  
[WHERE condition];
```

Exemple:

```
update t_compte_crt  
set solde = 0  
where solde < 0;
```

## DELETE: suppression de tuples

---

```
DELETE FROM nom_table  
[WHERE condition];
```

Exemple:

```
delete from t_compte_crt  
where solde < 0;
```

# INDEX

- Accélère l'accès aux tuples à partir de certains attributs

- Clé primaire
- Clés étrangères
- Attributs de sélection

- Création d'un index

```
CREATE [UNIQUE] INDEX nom_index  
ON nom_de_table (nom_colonne+);
```

- Suppression d'un index

```
DROP INDEX nom_index;
```

## INDEX: exemples

- **create unique index i\_poste  
on t\_client (ville, code\_postal);**

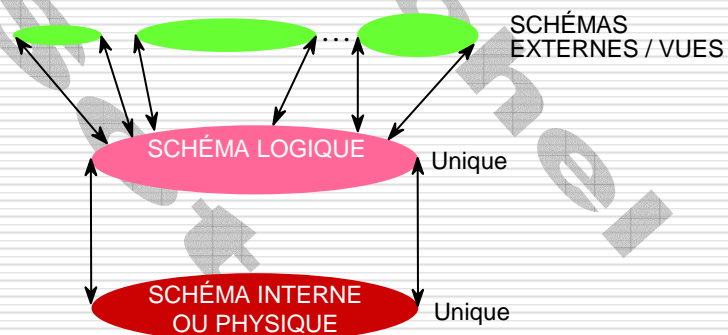
- **drop index i\_poste;**

## INDEX: discussion

- ❑ En consultation, un index permet d'accélérer la l'accès aux données
- ❑ En mise à jour des données, un index engendre un surcoût dû à sa propre mise à jour
- ❑ En règle générale, un index apporte globalement une amélioration lorsque le taux de consultation est supérieur au taux de mise à jour
  - Ces taux ne sont pas toujours connus au moment de la conception de la BD et peuvent évoluer au cours de la vie de la BD

## LES VUES (schémas externes)

- Le modèle ANSI-SPARC (1975)



# LES VUES

- ❑ **Table virtuelle** construite à l'aide d'une requête
- ❑ Utilisable comme une table réelle:
  - consultation: sans restriction
  - mise à jour: avec les contraintes suivantes:
    - ❑ La vue ne doit pas inclure la clause DISTINCT.
    - ❑ Chaque élément de la clause SELECT doit être un nom de colonne.
    - ❑ La clause FROM ne doit contenir qu'une seule table qui doit être modifiable.
    - ❑ La clause WHERE ne doit pas contenir de sous-requête.
    - ❑ La vue ne doit contenir ni clause GROUP BY, ni clause HAVING.
- ⚠
- ◆ Utilisable à des fins de confidentialité
  - REVOKE et GRANT possibles

## VUE: création, suppression

- ❑ Création d'une vue
  - CREATE VIEW nom [(liste de nom\_colonne)] AS requête;
- ❑ Suppression d'une vue
  - DROP VIEW nom\_vue;

## VUE: exemples

```
create view info_bon_client as
  select x.n_client, x.adresse, x.ville, x.postal, x.solde
  from t_client x, t_compte_crt y
  where y.solde > 15 000 and x.n_client = y.n_client;
```

ou bien

```
create view cumul_solde (n_client, cumul)
  as select x.n_client, sum (solde)
  from t_client x, t_compte_crt y
  where x.n_client = y.n_client
  group by x.n_client;
```

Schéma de la vue

Schéma de la vue

## VUE: exemples d'utilisation

```
select avg (solde) from info_bon_client;
```

ou bien

```
select y.nom, y.prenom
  from cumul_solde x, t_client y
  where x.cumul > 45 000
        and x.n_client = y.n_client;
```

```
drop view info_bon_client;
```