Définition

- Une vue permet de donner un nom à une requête qui sera utilisée ensuite comme une table (principalement en consultation).
- A chaque appel de la vue la requête est ré-exécutée.
- Attention ce n'est pas une vraie table!

Syntaxe

- **Création:**
  ```
  create or replace view nomDeVue as requete
  ```
- **Destruction:**
  ```
  drop view nomDeVue
  ```
- **Exemple**:
  ```
  create or replace view LesPrenoms(prenom) as
  select distinct prenom from CLIENT;
  ```

Exemple

```
create or replace view AgeCli(numCli, nom, prenom, age) as
    select numCli, nomCli, prenomCli, (CURDATE()-dateNaisCli)/365
    from CLIENT;
```

Utilisation des vues

- Afficher l'âge des clients qui ont effectué une commande en 2023
- ```
  select nom, age
  from AgeCli natural join COMMANDE
  where YEAR(dateCom)=2023;
  ```
  ```

Utilité

- Construction de requêtes complexes
- Structuration de la requête

Exemple

Afficher l'âge des clients qui ont effectué une commande en 2023

```
with AgeCli(numCli, nom, prenom, age) as
  (select numCli, nomCli, prenomCli, (CURDATE()-dateNaisCli)/365
  from CLIENT)
select nom, age
  from AgeCli natural join COMMANDE
  where YEAR(dateCom)=2023;
```

Les vues temporaires

Syntaxe

`with nomDeVue as ( requeteVue ) requete`

Si besoin de plusieurs vues, on les sépare par des `,`

Exemple

Afficher l'âge des clients qui ont effectué une commande en 2023

```
with AgeCli(numCli, nom, prenom, age) as
  (select numCli, nomCli, prenomCli, (CURDATE()-dateNaisCli)/365
  from CLIENT),
  Commande2023 as (select numCli, nomCli from COMMANDE where YEAR(dateCom)=2023)
select nom, age
  from AgeCli natural join Commande2023;
```

Les requêtes préparées

Principe

C'est une autre manière de donner un nom à une requête mais avec beaucoup de différences avec une vue:

- La requête préparée est une sorte de variable de type chaine de caractères
- La requête préparée **n'est pas persistante dans la base**
- La requête préparée **peut comporter des paramètres**
- La requête préparée **ne peut pas être utilisée comme une table dans une autre requête**

Exemple

```
prepare ClPrenom from
        'select distinct nom from CLIENT where prenom=?';
```

On recherche le nom de clients qui portent un certain prénom

Les requêtes préparées

Syntaxe

- **Création:**

  ```
  prepare nomDeReq from 'requeteSansPointVirgule';
  ```

  Les paramètres sont représentés par des ?
- **Destruction:**

  ```
  DEALLOCATE PREPARE nomDeReq;
  ```
- **Exécution**:

  ```
  execute nomDeReq using @param1 ...;
  ```

Exemple

- ```
  prepare ClPrenom from
      'select distinct nom from CLIENT where prenom=?';
  ```
- ```
  set @nom := 'Jean'; /* creation d'une variable */
  execute ClPrenom using @nom;
  ```

Les requêtes préparées

Intérêt

- Permet de créer des requêtes paramétrées
- Facilite l'utilisation de certaines requêtes (notamment lors de l'intégration de SQL dans un langage de programmation)

Attention!

- Les requêtes préparées ne sont pas des vues
- Elles ne sont pas persistantes (elles disparaissent avec la session)

Les différents types de jointures

Introduction

- Jointure naturelle: acolle les lignes qui sont égales sur les colonnes de même nom
- Permet d'exprimer les requêtes de type au moins un
- Ne permet pas d'exprimer les requêtes de type aucun

Autres jointures

- Permettre de tester l'égalité de colonnes de noms différents
- Permettre de garder toutes les lignes d'une table même si elles n'ont pas de correspondantes dans l'autre table.

Produit cartésien: `cross join`

```
select *
from CLIENT cl cross join COMMANDE co
where cl.numCli = co.numCli;
```

est équivalent à

```
select *
from CLIENT cl, COMMANDE co
where cl.numCli = co.numCli;
```

Jointure interne: `inner join`

- C'est la jointure classique avec quelques éléments syntaxiques
- Les trois requêtes suivantes sont équivalentes
  - ```
    select *
    from CLIENT cl inner join COMMANDE co on (cl.numCli=co.numCli);
    ```

    Le mot clé `on` indique les conditions de jointure $\approx$ condition de la clause `where`
  - ```
    select *
    from CLIENT cl inner join COMMANDE co using (numCli);
    ```

    `using` indique les colonnes sur lesquelles se fait la jointure
  - ```
    select *
    from adherent natural join participe;
    ```

    C'est la jointure naturelle comme on la connait.
    ```

Jointure externe: `outer join`

- Permet de garder toutes les lignes d'une des deux tables même celles qui n'ont pas d'équivalent dans la deuxième table
- Trois types de jointures
  - `left` garde toutes les lignes de la table de gauche
  - `right` garde toutes les lignes de la table de droite
  - `full` garde toutes les lignes des deux tables
- Le mot `outer` peut être omis.
- **Exemple**

```
select *
from CLIENT natural left outer join COMMANDE;
```

Jointure externe: Exemple

Instance

CLIENT

numCLi	nomCli	prenomCli
1	Dupont	Jean
2	Duval	Paul
3	Martin	Martine

COMMANDE

numCom	numCli	dateCom
1	1	2023-03-10
1	3	2023-03-10
3	1	2023-03-12

Requête

```
select * from CLIENT natural left join COMMANDE;
```

numCLi	nomCli	prenomCli	numCom	dateCom
1	Dupont	Jean	1	2023-03-10
1	Dupont	Jean	3	2023-03-12
2	Duval	Paul	NULL	NULL
3	Martin	Martine	1	2023-03-10

Exemples d'utilisation

- Répondre aux requêtes du type aucun sans requêtes imbriquées
- Eviter les fonctions d'agrégat `min` ou `max`
- Gérer les comptages avec des groupes vides (compter les 0)

Le prédicat IN (rappel)

Les clients qui ont effectué au moins une commande en 2023

- Avec IN

```
select * from CLIENT
where numCli in (
 select numCli from COMMANDE
 where YEAR(dateCom)=2023);
```

- Sans IN

```
select DISTINCT cl.*
from CLIENT cl natural join COMMANDE co
where YEAR(dateCom)=2023;
```

Le prédicat NOT IN

Les clients qui n'ont jamais commandé

- Avec NOT IN

```
select * from CLIENT
where numCli not in (
 select numCli from COMMANDE);
```

- Sans IN

```
select DISTINCT cl.*
from CLIENT cl natural left outer join COMMANDE
where numCom is NULL;
```

- **Remarques:**
  - Les clients qui n'ont effectué aucune commande sont ceux pour lesquels la colonne numCom vaut NULL après une jointure externe.
  - Sans notion de vue, on ne peut pas exprimer la requête: Les clients qui n'ont effectué aucune commande en 2023.

La fonction max

Quelle est la commande la plus recente?

- Avec max

```
select * from COMMANDE
where dateCom = (select max(dateCom) from
COMMANDE);
```

- Sans max

```
select DISTINCT *
from COMMANDE c1 left join commande c2
on (c1.dateCom<c2.dateCom)
where c2.dateCom is NULL;
```

- **Remarques:**
  - La commande la plus récente est celle dont aucune autre commande a une date supérieure à la sienne
  - Cette astuce ne fonctionne que si on n'a pas besoin de sélection dans la table de droite ⇒ nécessite une vue

Compter les 0

- Pour chaque client, on veut le nombre de commandes effectuées en 2023

```
select numCli, nomCli, count(numCom)
from CLIENT natural join COMMANDE
where YEAR(dateCom) = 2023
group by numCli, nomCli;
```

- **Remarque**
  - On compte ici le nombre de commandes des clients qui ont effectué au moins une commande
  - **Donc pas de lignes avec 0**

Compter les 0

- Pour chaque client, on veut le nombre de commandes effectuées en 2023

```
 -- la liste des commandes de 2023
with co23 as (
    select * from COMMANDE
    where YEAR(dateCom)=2023)

-- la requête avec les 0
select numCli, nomCli, count(numCom)
from CLIENT natural left join co23
group by numCli,nomCli;
```

- count compte le nombre de valeurs connues c'est à dire le nombre de valeurs différentes de NULL

Traitement des valeurs NULL

Exemple

```
create table LOCATION(numV int, numCli int, dateDebut date, dateFin date);
```

- On veut le nombre de jours de location des véhicules non rendus
- Un véhicule est non-rendu si sa `dateFin` est `null`
- Il faut donc remplacer la valeur `NULL` par une autre valeur

La fonction IFNULL

```
select numV, DATEDIFF(IFNULL(dateFin,CURDATE()),dateDebut)
from LOCATION;
```

- Si la `dateFin` est null alors on la replace par la valeur de `CURDATE()`
- Utile dans les requêtes nécessitant des calculs pouvant comporter des `null`