Les bases de données relationnelles

3.0 31/03/2021

Bloc 1 - Exploitation de données

Table des matières

Bases de données - SQL - Livre 1 : Interroger une base de données

1. L'interrogation de données - Le SELECT - Partie 3

1.1. L'interrogation de données - Les sous-requêtes

SQL permet de comparer une expression ou un attribut au résultat d'une autre requête SELECT. Cette condition est dite condition de sous-requêtes et les deux requêtes sont dites requêtes imbriquées. Il n'y a pas de limite théorique au nombre de requêtes que l'on peut imbriquer : une sous-requête peut faire appel à une autre sous-requête, etc.

a) Les sous-requêtes

La commande SELECT est sous le format suivant :

SELECT [ALL / DISTINCT] <liste d'attributs>
FROM <liste des relations utilisées>
WHERE exp **opérateur_de_comparaison** [**ALL / ANY**] (requête SELECT); ❶
ou : **WHERE** exp [**NOT**] **IN** (requête SELECT); ❷
ou : **WHERE** [**NOT**] **EXISTS** (requête SELECT); ❸

Commande SELECT - Sous-requête

- ALL : vrai si la comparaison est vraie pour **chacune** des valeurs retournées.
- ANY : vrai si la condition est vraie pour **au moins une** des valeurs retournées.
- IN : vrai si la comparaison est vraie pour **au moins une** des valeurs retournées (pas d'utilisation d'opérateur de comparaison avec IN contrairement à ANY).

Utilisation d'opérateurs de comparaison seuls (1) ***Exemple***

L'utilisation d'opérateurs de comparaison seuls (sans ALL ou ANY) implique que **la sous-requête doit renvoyer une valeur unique** contrairement aux opérateurs de comparaison avec ALL ou ANY où la sous-requête peut renvoyer une liste de valeurs.

Lister les lignes de commandes où le nombre d'articles est supérieur au nombre moyen d'articles. ***Syntaxe***

```
SELECT *
FROM Commande
WHERE Qté > (SELECT AVG (Qté)
             FROM Commande);
```

Remarque

On recherche la moyenne des "Qté" de la relation "Commande". On sélectionne alors les tuples de "Commandes" où l'attribut "Qté" est supérieur à cette moyenne.

Prédicat ALL / ANY (1)

Exemple

Les sous-requêtes situées après ALL et ANY **doivent avoir comme résultat un seul attribut**.

ALL : Lister les codes salariés élus au Prud'hommes mais pas au comité d'entreprise

Syntaxe

```
SELECT Code_salarié
FROM Prudh
WHERE Code_salarié != ALL
                    (SELECT Code_salarié
                    FROM Coment);
```

Remarque

On recherche la liste de tous les "Code_salarié" de la relation "Coment". On sélectionne les tuples de la relation "Prudh" pour lesquels le "Code_salarié" est différent de toutes les valeurs de la liste.

Lister les lignes de commandes à l'exception de celles qui ont la plus petite quantité commandée.

Syntaxe

```
SELECT *
FROM Commande
WHERE Qté > ANY
            (SELECT Qté
            FROM Commande);
```

Remarque

On recherche la liste de toutes les "Qté" de la relation "Commande". On sélectionne les tuples de la relation "Commande" où la "Qté" du tuple traité est strictement supérieur à au moins une valeur de cette liste, on ne prend donc pas la plus petite valeur.

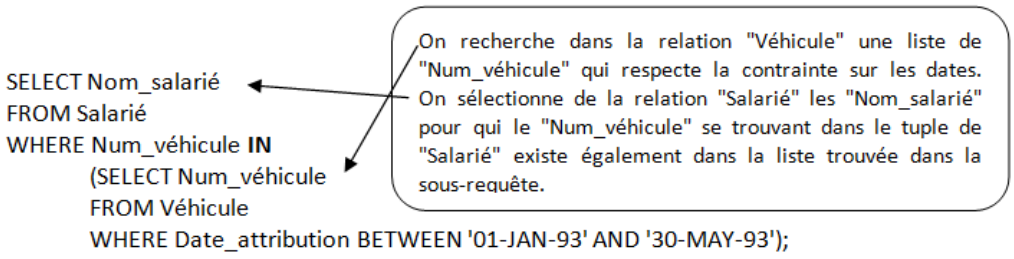
IN / (2)

Exemple

Lister tous les noms de salariés à qui une voiture a été attribuée entre le 01/01/93 et le 30/06/93.

Syntaxe

```
SELECT Nom_salarié
FROM Salarié
WHERE Num_véhicule IN
                    (SELECT Num_véhicule
                    FROM Véhicule
                    WHERE Date_attribution BETWEEN '01-JAN-93' AND '30-MAY-93');
```



Commande SELECT - IN

NOT IN (2) *Exemple*

On souhaite connaître le nom des clients qui n'ont pas passé de commande.

Syntaxe

```
SELECT CLIENT.NomCli
FROM CLIENT
WHERE CLIENT.NumCli = COMMANDE.NumCli
AND CLIENT.NumCli NOT IN
                  (SELECT DISTINCT NumCli FROM COMMANDE) ;
```

EXISTS (3) *Exemple*

EXISTS : renvoie le booléen vrai ou faux. Si le résultat de la sous-requête donne lieu à une ou plusieurs lignes, la valeur retournée est vrai. Dans le cas contraire, la valeur retournée est fausse.

Lister les produits non pris en charge par un représentant. *Syntaxe*

```
SELECT Num_produit
FROM Produit P
WHERE NOT EXISTS
            (SELECT Code_représentant
            FROM Représentant R
            WHERE P.Num_produit = R.Num_produit);
```

Remarque

L'intérêt du EXISTS est qu'il permet de ne pas ramener dans la sous-requête un attribut devant être comparé avec un attribut se trouvant dans le "WHERE". On teste ici l'existence ou l'absence de données dans la sous requête.

b) Exercice d'application

Simulation

Cas : Gestion des employés

1.2. Exercice : Cas : Gestion des employés

A partir des relations ci-dessous, écrire les requêtes SQL demandées.

EMP (ENO, ENOM, PROF, DATEEMB, SAL, COMM, DNO#)

ENO : numéro d'employé, clé primaire
ENOM : nom de l'employé
PROF : profession (directeur n'est pas une profession)
DATEEMB : date d'embauche
SAL : salaire
COMM : commission (un employé peut ne pas avoir de commission)
DNO : numéro de département auquel appartient l'employé, clé étrangère

Gestion des employés - Table EMP

DEPT (DNO, DNOM, DIR, VILLE)

DNO : numéro de département, clé primaire
DNOM : nom du département
DIR : directeur du département
VILLE : lieu du département (ville)

Gestion des employés - Table DEPT

EMP

ENO	ENOM	PROF	DATEEMB	SAL	COMM	DNO
10	Joe	Ingénieur	1.10.93	4000	3000	3
20	Jack	Technicien	1.5.88	3000	2000	2
30	Jim	Vendeur	1.3.80	5000	5000	1
40	Lucy	Ingénieur	1.3.80	5000	5000	3

DEPT

DNO	DNOM	DIR	VILLE
1	Commercial	30	New York
2	Production	20	Houston
3	Développement	40	Boston

Gestion des employés - Tuples

Question 1

Donner les noms et les salaires des employés.

Question 2

Donner les professions des employés (après élimination des duplicats).

Question 3

Donner les dates d'embauche des techniciens.

Question 4

Faire le produit cartésien entre EMP et DEPT.

Question 5

Donner les noms des employés et les noms de leur département des employés travaillant à BOSTON.

Question 6

Donner les noms des directeurs des départements 1 et 3.

Attention : directeur n'est pas une profession !

Question 7

Donner la liste des employés ayant une commission.

Question 8

Donner les noms, emplois et salaires des employés par emploi croissant et, pour chaque emploi, par salaire décroissant.

Question 9

Donner le salaire moyen des employés.

Question 10

Donner le nombre d'employés du département PRODUCTION.

Question 11

Les numéros de département et leur salaire maximum ?

Question 12

Donner les noms des employés travaillant dans un département avec au moins un ingénieur.

Question 13

Donner le salaire et le nom des employés gagnant plus qu'un (au moins un) ingénieur.

Question 14

Donner le salaire et le nom des employés gagnant plus que tous les ingénieurs.

Question 15

Donner les départements qui n'ont pas d'employés.

Question 16

Donner les noms des employés du département COMMERCIAL embauchés le même jour qu'un employé du département PRODUCTION.

Question 17

Donner les noms des employés embauchés avant tous les employés du département 1.

Question 18

Donner les noms des employés ayant le même emploi et le même directeur que JOE.

Question 19

Trouver les noms des employés ayant le même directeur que JIM.

Attention : un employé peut être directeur de plusieurs départements.

Question 20

Donner les noms des employés ayant le salaire maximum de chaque département.

Question 21

Les professions et leur salaire moyen.

Question 22

Le salaire moyen le plus bas (par profession).

Question 23

Donner les emplois ayant le salaire moyen le plus bas ; donnez aussi leur salaire moyen.

2. L'interrogation de données - Le SELECT - Partie 4

2.1. L'interrogation de données - Les opérations ensemblistes

Le langage SQL est basé sur des notions définies par l'algèbre relationnelle. Elle est constituée d'un ensemble d'opérations formelles sur les relations. En plus des opérations de projection, sélection et jointure, on dispose d'opérations ensemblistes : l'union, l'intersection, la différence et le produit cartésien.

a) Les opérations ensemblistes

i) L'Union

Les relations doivent avoir la même structure. On relie le résultat de deux ou plusieurs SELECT effectués en même temps.

? *Exemple*

On désire obtenir l'ensemble des enseignants élus au CA ou représentants syndicaux.

Table E1 : Enseignants au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

Table E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

2 tables à unir

n°enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND
6	MICHEL

Résultat de l'UNION

Q *Remarque*

- Cet opérateur porte sur deux relations qui doivent avoir le **même nombre d'attributs** définis dans le même domaine (ensemble des valeurs permises pour un attribut). On parle de relations ayant le **même schéma**.
- La relation résultat possède les attributs des relations d'origine et les n-uplets de chacune, avec élimination des doublons éventuels.

§ *Syntaxe*

```
<requête de sélection : SELECT... >
UNION [ALL]
<requête de sélection : SELECT... > ;
```

- UNION : supprime tous les tuples redondants (en plus de 1 exemplaires),
- UNION ALL : conserve les tuples redondants (implémenté sous certain SGBDR dont Oracle).

ici :

```
SELECT * FROM ENSEIGANTS
UNION
SELECT * FROM REPRESENTANTS_SYNDICAUX ;
```

ii) L'intersection

Les relations doivent avoir **la même structure**. L'intersection de deux requêtes effectuées en même temps donne comme résultat les lignes appartenant à la fois à ces deux requêtes.

Exemple

On désire connaître les enseignants du CA qui sont des représentants syndicaux.

Table E1 : Enseignants au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

Table E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

2 tables à unir

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN

Résultat de l'INTERSECTION

- Cet opérateur porte sur deux relations de même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets communs à chacune.

Syntaxe

```
<requête de sélection : SELECT... >
INTERSECT
<requête de sélection : SELECT... > ;
```

ici :

```
SELECT * FROM ENSEIGANTS
INTERSECT
SELECT * FROM REPRESENTANTS_SYNDICAUX ;
```

Remarque

certains SGBDR n'implantent pas la commande INTERSECT, dans ce cas il est possible d'utiliser une équijointure entre les deux relations (ou vues) sur l'ensemble des attributs.

```
SELECT E1.n°enseignant, E1.nom_enseignant
FROM ENSEIGNANTS E1, REPRESENTANTS_SYNDICAUX E2
WHERE E1.n°enseignant = E2.n°enseignant
AND E1.nom_enseignant = E2.nom_enseignant ;
```

iii) La différence

Les relations doivent avoir la **même structure**. La différence de deux requêtes effectuées en même temps liste les lignes du résultat de la première requête qui ne sont pas comprise dans le résultat de la deuxième requête.

? Exemple

On désire obtenir la liste des enseignants du CA qui ne sont pas des représentants syndicaux.

Table E1 : Enseignants au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

Table E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

2 tables à unir

n°enseignant	nom_enseignant
3	DURAND
5	BERTRAND

Résultat de la DIFFERENCE

- Cet opérateur porte sur deux relations de même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets de la première relation qui n'appartiennent pas à la deuxième.

! Attention

DIFFERENCE (R1, R2) ne donne pas le même résultat que DIFFERENCE (R2, R1).

n°enseignant	nom_enseignant
6	MICHEL

Autre résultat de différence

§ Syntaxe

```
<requête de sélection : SELCT... >
MINUS
<requête de sélection : SELECT... > ;
```

exemple :

```
SELECT * FROM ENSEIGNANTS
MINUS
SELECT * FROM REPRESENTANTS_SYNDICAUX;
```

Remarque

certains SGBDR (comme Microsoft Access) n'implantent pas la commande MINUS, dans ce cas il est possible d'utiliser une requête imbriquée utilisant une équijointure entre les deux relations sur l'ensemble des attributs.

```
SELECT E1.n°enseignant, E1.nom_enseignant
FROM ENSEIGNANTS E1
WHERE NOT EXISTS
      (SELECT E2.n°enseignant, E2.nom_enseignant
      FROM REPRESENTANTS_SYNDICAUX E2
      WHERE E1.n°enseignant = E2.n°enseignant
```

```
      AND E1.nom_enseignant = E2.nom_enseignant);
```

Attention

La commande MINUS se nomme EXCEPT sous PostGreSQL.

iv) Le produit cartésien

Exemple

On veut connaître pour chaque étudiant les épreuves de l'examen.

Table ETUDIANTS

n°étudiant	nom
101	DUPONT
102	MARTIN

Table EPREUVES

libellé épreuve	coefficient
Informatique	2
Mathématiques	3
Gestion financière	5

Tables à joindre

n°étudiant	nom	libellé épreuve	coefficient
101	DUPONT	Informatique	2
101	DUPONT	Mathématiques	3
101	DUPONT	Gestion financière	5
102	MARTIN	Informatique	2
102	MARTIN	Mathématiques	3
102	MARTIN	Gestion financière	5

Résultat du produit cartésien

- Cet opérateur porte sur deux relations.
- La relation résultat possède les attributs de chacune des relations d'origine et ses n-uplets sont formés par la concaténation de chaque n-uplet de la première relation avec l'ensemble des n-uplets de la deuxième.

Syntaxe

```
SELECT *
FROM Relation_1, Relation_2 ;
```

ici :

```
SELECT *
FROM ETUDIANTS, EPREUVES ;
```

Remarque

Remarque : la jointure est un cas particulier du produit cartésien, elle est formulée avec la clause WHERE.

b) Exercices d'application

Simulation

Exercice 1 : Gestion commerciale (QCM)

Simulation

Exercice 2 : Gestion des commandes

Simulation

Exercice 3 : Gestion des clients

Simulation

Exercice 4 : Médiathèque

2.2. Exercice : Cas : Gestion des commandes

Soit une base de données contenant deux tables : CLIENT et COMMANDE dotées des structures suivantes ...

NO_CLIENT	INTEGER
NOM_CLIENT	CHAR(32)

143	DUPONT
212	MARTIN
823	DUBOIS

Exercice 1 - Table CLIENT

REF_COMMANDE	CHAR(16)
DATE_COMMANDE	DATE
MONTANT_COMMANDE	MONEY
NO_CLIENT	INTEGER

2009-11	11/7/2009	1235.52	212
2009-12	17/7/2009	45234.63	823
2009-13	18/7/2009	5485.23	142
2009-14	21/7/2009	11542.23	212

Exercice 1 - Table COMMANDE

Écrire les requêtes SQL suivantes :

Question 1

Donnez la liste des clients (Numéro et nom).

Question 2

Donnez la liste des commandes (références et dates) qui ont coûté plus de 10 000 euros.

Question 3

Donnez toutes les caractéristiques des commandes passées en juillet 2009 par le client nommé "DUBOIS"

Question 4

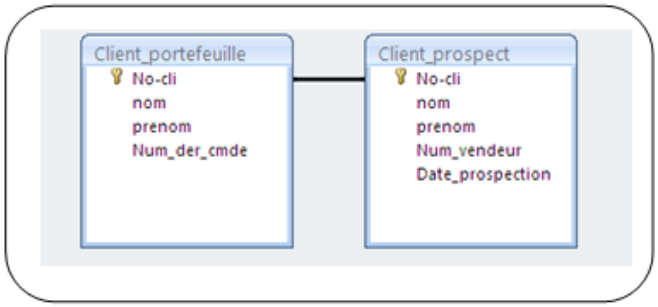
Donnez la liste des clients (numéro et noms) qui n'ont pas passé commande.

2.3. Exercice : Cas : Gestion des clients

Afin de gérer les clients, une entreprise possède une table qui recense les clients prospectés et une table qui liste les clients qui ont passé des commandes.

Client_portefeuille

No-cli	nom	prenom	Num_der_cmde
1	Patamob	Adhémar	766
4	Locale	Anasthasie	989
7	Rivenbusse	Elsa	345
8	Ardelpic	Helmut	987



Client_prospect

No-cli	nom	prenom	Num_vendeur	Date_prospection
1	Patamob	Adhémar	1	03/26/2009
2	Zeublouze	Agathe	1	04/15/2009
3	Kuzbidon	Alex	2	05/05/2009
5	Teutmaronne	Armand	4	06/14/2009
6	Zoudanlkou	Debbie	3	07/04/2009
7	Rivenbusse	Elsa	2	07/24/2009

Exercice 2 - Tables

Exercice 2 - Rappel

Attention : dans le cas des différences on utilisera EXCEPT (PostGreSQL) plutôt que MINUS (SQL standard)

Question 1

Donnez la liste des clients (N°, nom et prénom) dont la prospection s'est avérée fructueuse (qui ont donc passé commande).

Question 2

Donnez la liste des clients (N°, nom et prénom) dont la prospection s'est avérée infructueuse (qui n'ont donc pas passé commande).

Question 3

Donnez la liste de tous les clients (qu'ils aient passé commande ou non) avec toutes leurs caractéristiques.

Question 4

Donnez la liste des clients (N°, nom et prénom) qui ont passé commande sans avoir été prospectés.

Question 5

Donnez la liste de tous les clients (qu'ils aient passé commande ou non) : numéro, nom et prénom.

2.4. Exercice : Cas médiathèque

On considère le schéma relationnel suivant qui modélise une application sur la gestion de livres et de disques dans une médiathèque :

DISQUE (CodeOuv, Titre, Style, Pays, Année, Producteur)
Cette relation regroupe un certain nombre d'informations sur un disque : le code d'ouvrage (CodeOuv) qui est la clé de la relation, le titre, le style (par exemple Jazz ou Rock), le pays, l'année de sortie et le producteur (par exemple Barclay). Ces informations sont générales et pour un enregistrement de la relation DISQUE, on aura n (n > 1) enregistrements dans la relation E_DISQUE correspondant aux exemplaires de ce disque possédés par la médiathèque.

E_DISQUE (CodeOuv#, NumEx, DateAchat, Etat)
Cette relation contient un enregistrement pour chaque exemplaire de disque possédé par la médiathèque. Chaque exemplaire est identifié par son code (CodeOuv) et un numéro d'exemplaire (NumEx). On trouve également la date d'achat (DateAchat) et l'état du disque (par exemple Neuf ou Abimé).

LIVRE (CodeOuv, Titre, Genre, Editeur, Collection)
Cette relation regroupe un certain nombre d'informations sur un livre : le code d'ouvrage (CodeOuv) qui est la clé de la relation, le titre, le genre (par exemple Policier ou Roman), l'éditeur (par exemple Glénat) et la collection (par exemple livre de poche). Ces informations sont générales et pour un enregistrement de la relation LIVRE, on aura n (n > 1) enregistrements dans la relation E_LIVRE correspondant aux exemplaires de ce livre possédés par la médiathèque.

E_LIVRE (CodeOuv#, NumEx, DateAchat, Etat)
Cette relation contient un enregistrement pour chaque exemplaire de livre possédé par la médiathèque. Chaque exemplaire est identifié par son code (CodeOuv) et un numéro d'exemplaire (NumEx). On trouve également la date d'achat (DateAchat) et l'état du livre (par exemple Neuf ou Abimé).

AUTEUR (CodeOuv, Identité)
Chaque enregistrement de cette relation correspond à l'un des auteurs d'un ouvrage particulier (livre ou disque). L'attribut identité peut avoir pour valeur un nom de personne (par exemple Alexandre Dumas) ou un nom de groupe (par exemple Rolling Stones).

ABONNE (NumAbo, Nom, Prénom, Rue, Ville, CodeP, Téléphone)
Cette relation regroupe les informations sur les abonnés de la médiathèque : NumAbo qui identifie tout abonné de manière individuelle, le nom (Nom) et le prénom (Prénom) de l'abonné, son adresse (Rue, Ville, CodeP), son téléphone (Téléphone).

PRET (CodeOuv#, NumEx#, NumAbo#, DatePrêt)
Cette relation contient un enregistrement par prêt effectué. Pour chaque prêt, on trouve l'identifiant du livre ou du disque (code ouvrage CodeOuv et numéro d'exemplaire NumEx), le numéro de l'abonné effectuant le prêt (NumAbo) et enfin la date du prêt (DatePrêt). Cette relation ne contient des informations que pour les prêts en cours c'est à dire pour les emprunts non encore rendus.

PERSONNEL (NumEmp, Nom, Prénom, Adresse, Fonction, Salaire)
Cette relation contient un enregistrement par employé. Chaque employé est identifié par un numéro NumEmp. Pour chaque employé, la relation donne son nom, son prénom, son adresse, sa fonction et son salaire mensuel.

Exercice 3 - Relations

Présenter les requêtes SQL pour répondre aux besoins d'information exprimés :

Question 1

Quel est le contenu de la relation LIVRE ?

Question 2

Quels sont les titres des romans édités par Flammarion ?

Question 3

Liste des titres que l'on retrouve à la fois comme titre de disque et titre de livre ?

Question 4

Quelle est l'identité des auteurs qui ont fait des disques et écrit des livres ?

Question 5

Quels sont les différents styles de disques proposés ?

Question 6

Quel est le salaire annuel des membres du personnel gagnant plus de 150 000 euros en ordonnant le résultat par salaire descendant et nom croissant ?

Question 7

Donnez le nombre de prêts en cours pour chaque famille en considérant qu'une famille regroupe des personnes de même nom et possédant le même numéro de téléphone ?

Question 8

Quel est le code du disque dont la médiathèque possède le plus grand nombre d'exemplaires?

Question 9

Quels sont les éditeurs pour lesquels l'attribut Collection n'a pas été renseigné ?

Question 10

Quels sont les abonnés dont le nom contient la chaîne " ALDO " et habitant en Isère ?

Question 11

Quel est le nombre de prêts en cours ?

Question 12

Quels sont les salaires minimum, maximum et moyen des employés exerçant une fonction de bibliothécaire ?

Question 13

Quel est le nombre de genres de livres différents ?

Question 14

Quel est le nombre de disques achetés en 20088 ?

Question 15

Quel est le salaire annuel des membres du personnel gagnant plus de 150 000 euros ?

Question 16

Quel est le nom, le prénom et l'adresse des abonnés ayant emprunté un disque le 12/01/2009 ?

Question 17

Quels sont les titres des livres et des disques actuellement empruntés par Marlène Sirva ?

Question 18

Quels sont les titres des ouvrages livres policiers ou disques de jazz empruntés par Olivier Lobry ?

Question 19

Quelle est l'identité des auteurs qui n'ont écrit que des romans policiers (genre = policier) ?

Question 20

Quels sont les codes des ouvrages des livres pour lesquels il y a au moins un exemplaire emprunté et au moins un exemplaire disponible ?

3. L'interrogation de données - Le SELECT - Partie 5

3.1. L'interrogation de données - Les jointures

Les jointures permettent de mettre en relation plusieurs tables concourant à rechercher la ré-ponse à des interrogations et donc de combiner les colonnes de différentes tables.

Il existe en fait différentes natures de jointures que nous expliciterons plus en détail.

Retenez cependant que la plupart des jointures entre tables s'effectuent en imposant l'égalité des valeurs d'une colonne d'une table à une colonne d'une autre table. On parle alors de jointure na-turelle et équijointure. Mais on trouve aussi des jointures d'une table sur elle-même (on parle alors d'auto-jointure). Enfin il arrive que l'on doive procéder à des jointures externes, c'est à dire joindre une table à une autre, même si la valeur de liaison est absente dans une table ou l'autre.

Une jointure entre tables peut être mise en œuvre, soit à l'aide des éléments de syntaxe SQL que nous avons déjà vu (SELECT ... FROM T1, T2 WHERE T1.CHAMPS = T2.CHAMPS...) , soit à l'aide d'une clause spécifique du SQL, la clause JOIN.

a) Les requêtes sans jointure

Rappel de la syntaxe du SELECT — *Syntaxe*

SELECT [DISTINCT ou ALL] * ou liste de colonnes

FROM nom des tables ou des vues

Exemple

Récupération des n° des téléphones associés aux clients

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT, T_TELEPHONE ;
```

```
CLI_NOM  TEL_NUMERO
-------  --------------
DUPONT   01-45-42-56-63
DUPONT   01-44-28-52-52
DUPONT   01-44-28-52-50
DUPONT   06-11-86-78-89
DUPONT   02-41-58-89-52
DUPONT   01-51-58-52-50
DUPONT   01-54-11-43-21
DUPONT   06-55-41-42-95
DUPONT   01-48-98-92-21
...
```

SELECT - Produit cartésien

Dans ce cas, que calcule le compilateur SQL ? **Le produit cartésien des deux ensembles**.

Dans notre exemple la requête génère 17 400 lignes... (La table « T_CLIENT » possède 100 enregistrements et la table « T_TELEPHONE » en possède 174).

Il faut donc définir absolument un critère de jointure.

b) Jointure naturelle et équijointure

Nous allons commencer par voir comment à l'aide du SQL de base nous pouvons exprimer une jointure.

Dans le cas présent, ce critère est la correspondance entre les colonnes contenant la référence de l'identifiant du client (CLI_ID). Le résultat final est de 174 enregistrements.

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT, T_TELEPHONE
WHERE T_CLIENT.CLI_ID  = T_TELEPHONE.CLI_ID ;

Ou (avec des alias)

SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT C, T_TELEPHONE T
WHERE C.CLI_ID = T.CLI_ID ;
```

```
CLI_NOM   TEL_NUMERO
-------          --------------
DUPONT    01-45-42-56-63
DUPONT    01-44-28-52-52
DUPONT    01-44-28-52-50
BOUVIER   06-11-86-78-89
DUBOIS    02-41-58-89-52
DREYFUS   01-51-58-52-50
DUHAMEL   01-54-11-43-21
BOYER     06-55-41-42-95
MARTIN    01-48-98-92-21
MARTIN    01-44-22-56-21
...
```

SELECT - Jointure

Définition

Une jointure naturelle est une jointure entre clefs primaires et clefs secondaires basée sur l'égalité des valeurs des colonnes.

cas de non équijointure ***Remarque***

Le fait de placer comme critère de jointure entre les tables, l'opérateur logique « égal » donne ce que l'on appelle une « équijointure ». Il est possible de faire des équijointures qui ne sont pas naturelles. Il s'agit là d'utiliser n'importe quelle condition de jointure entre deux tables, excepté la stricte égalité :

- > : supérieur
- >= : supérieur ou égal
- < : inférieur
- <= : inférieur ou égal
- <> ou != : différent de
- IN : dans un ensemble
- LIKE : correspondance partielle
- BETWEEN : entre deux valeurs
- EXISTS : dans une table

c) Syntaxe normalisée des jointures

Les jointures normalisées s'expriment à l'aide du mot clef JOIN dans la clause FROM.

Jointure interne — *Syntaxe*

```
SELECT ...
FROM <table gauche> [INNER] JOIN <table droite>
ON <condition de jointure>...
```

Jointure externe — *Syntaxe*

```
SELECT ...
FROM <table gauche> LEFT | RIGHT | FULL OUTER JOIN <table droite>
ON <condition de jointure>...
```

i) La jointure interne

Il s'agit de la plus commune des jointures.

Exemple

Reprenons notre exemple de départ :

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT
    ........ INNER JOIN T_TELEPHONE
        ..... ON T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID ;

Ou (avec des alias)


SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
        ........ INNER JOIN T_TELEPHONE T
            ........ ON C.CLI_ID = T.CLI_ID ;
```

```
CLI_NOM    TEL_NUMERO
-------    --------------
DUPONT     01-45-42-56-63
DUPONT     01-44-28-52-52
DUPONT     01-44-28-52-50
BOUVIER    06-11-86-78-89
DUBOIS     02-41-58-89-52
DREYFUS    01-51-58-52-50
DUHAMEL    01-54-11-43-21
BOYER      06-55-41-42-95
MARTIN     01-48-98-92-21
MARTIN     01-44-22-56-21
...
```

SELECT - Jointure interne

Remarque

Le mot clef INNER est facultatif, il est implicite.

ii) La jointure externe

Client BOUVIER — *Exemple*

Voici le résultat obtenu en interrogeant le client « BOUVIER ». Il possède un GSM, un TEL mais pas de FAX.

```
SELECT CLI_NOM, TEL_NUMERO, TYP_CODE
FROM T_CLIENT C, T_TELEPHONE T  WHERE C.CLI_ID = T.CLI_ID
AND CLI_NOM = 'BOUVIER';
```

```
CLI_NOM    TEL_NUMERO      TYP_CODE
-------    --------------  --------
BOUVIER    06-11-86-78-89  GSM
BOUVIER    04-94-41-17-27  TEL
```

Bouvier - Résultat 1

Cherchons à présent les clients dont les numéros de téléphone correspondent à un fax :

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT C, T_TELEPHONE T
WHERE C.CLI_ID = T.CLI_ID
 AND TYP_CODE = 'FAX' ;

⇔

SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
     INNER JOIN T_TELEPHONE  T
      ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX' ;
```

```
CLI_NOM       TEL_NUMERO
-------       --------------
DUPONT        01-44-28-52-50
MARTIN        01-44-22-56-21
DUHAMEL       01-54-11-43-89
DUPONT        05-59-45-72-42
MARTIN        01-47-66-29-55
DUBOIS        04-66-62-95-64
DREYFUS       04-92-19-18-58
DUHAMEL   01-55-60-93-81
PHILIPPE  01-48-44-86-19
DAUMIER   01-48-28-17-95
...
```

Bouvier - Résultat 2

Comme vous pouvez le constater, le nom du client BOUVIER n'apparaît pas. Il n'a pas été « oublié » par le traitement de la requête, mais le numéro de fax de ce client n'est pas présent dans la table T_TELEPHONE.

Or le moteur SQL recherche les valeurs de la jointure par égalité. Comme l'ID_CLI de BOUVIER n'est pas présent dans la table T_TELEPHONE, il ne peut effectuer la jointure et ignore donc la ligne concernant le client BOUVIER.

Méthode

Que faut-il modifier dans la requête pour obtenir une ligne BOUVIER avec aucune référence de téléphone associée dans la réponse ?

Il suffit en fait d'opérer à l'aide d'une jointure externe :

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
     LEFT OUTER JOIN T_TELEPHONE T
          ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX' ;
```

```
CLI_NOM   TEL_NUMERO
-------       --------------
DUPONT        01-44-28-52-50
DUPONT        05-59-45-72-42
MARTIN        01-47-66-29-55
BOUVIER       NULL
DUBOIS        04-66-62-95-64
DREYFUS       04-92-19-18-58
FAURE         NULL
LACOMBE       NULL
DUHAMEL       01-54-11-43-89
DUHAMEL       01-55-60-93-81
...
```

Bouvier - Résultat 3

§ Syntaxe

```
SELECT ...
FROM <table gauche>
LEFT | RIGHT | FULL OUTER JOIN <table droite 1>
ON <condition de jointure>
[LEFT | RIGHT | FULL OUTER JOIN <table droite 2>
ON <condition de jointure 2>]
...

```

Remarque

Les mots clefs LEFT, RIGHT et FULL indiquent la manière dont le moteur de requête doit effectuer la jointure externe. Ils font référence à la table située à gauche (LEFT) du mot clef JOIN ou à la table située à droite (RIGHT) de ce même mot clef. Le mot FULL indique que la jointure externe est bilatérale.

LEFT OUTER JOIN *Syntaxe*

```
SELECT colonnes
FROM TGauche LEFT OUTER JOIN TDroite ON condition de jointure ...
```

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TGauche qui n'ont pas été prises en compte au titre de la satisfaction du critère.

RIGHT OUTER JOIN *Syntaxe*

```
SELECT colonnes
FROM TGauche RIGHT OUTER JOIN TDroite ON condition de jointure...
```

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

FULL OUTER JOIN *Syntaxe*

```
SELECT colonnes
FROM TGauche FULL OUTER JOIN TDroite ON condition de jointure

```

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TGauche et TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

Conseil

Dans la mesure du possible, utiliser toujours un opérateur de jointure normalisé (mot clef JOIN).

En effet :

- Les jointures faites dans la clause WHERE (ancienne syntaxe de 1986 !) ne permettent pas de faire la distinction de prime abord entre ce qui relève du filtrage et ce qui relève de la jointure. La lisibilité des requêtes est donc plus grande en utilisant la syntaxe à base de JOIN.
- L'optimisation d'exécution de la requête est souvent plus pointue du fait de l'utilisation du JOIN.

iii) L'auto jointure

Définition

Une auto-jointure est une jointure qui permet de joindre la table sur elle-même.

Chambres communicantes *Exemple*

Pour donner un exemple concret à nos propos nous allons modéliser le fait qu'une chambre puisse communiquer avec une autre (par une porte). Dès lors, le challenge est de trouver quelles sont les chambres qui communiquent entre elles.

Table T_Chambre

CHB_ID	CHB_COMMUNIQUE
9	7
7	9
12	14
14	12

SELECT - Auto-jointure

Pour formuler la recherche de chambres communicantes, il suffit de faire la requête suivante :

```
SELECT DISTINCT c1.CHB_ID, c1.CHB_COMMUNIQUE
FROM   T_CHAMBRE c1
       INNER JOIN T_CHAMBRE c2
             ON c1.CHB_ID = c2.CHB_COMMUNIQUE
```

```
CHB_ID  CHB_COMMUNIQUE
------  --------------
     7               9
     9               7
    12              14
    14              12
```

Auto-jointure - Résultat

...ou la table T_CHAMBRE figure deux fois par l'entremise de deux surnommages différents c1 et c2.

Notons que cette présentation n'est pas pratique car elle dédouble le nombre de couples de chambres communicantes, donnant l'impression qu'il y a 4 couples de chambres communicantes ce qui est faux...

Il manque juste un petit quelque chose pour que cette requête soit parfaite. Il suffit en effet de ne retenir les solutions que pour des identifiants inférieurs ou supérieur d'une table par rapport à l'autre :

```
SELECT DISTINCT c1.CHB_ID, c1.CHB_COMMUNIQUE
FROM   T_CHAMBRE c1
       INNER JOIN T_CHAMBRE c2
             ON c1.CHB_ID = c2.CHB_COMMUNIQUE
                AND c1.CHB_ID <= c2.CHB_ID
```

```
CHB_ID  CHB_COMMUNIQUE
------  --------------
     7               9
    12              14
```

Auto-jointure - Résultat correct

§ Syntaxe

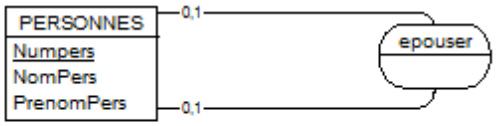
```
SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM laTable t1 INNER JOIN laTable t2
ON t1.laClef = t2.laPseudoClefEtrangère

```

Remarque

L'utilisation des alias est obligatoire dans ce type de jointure.

Il est assez fréquent que l'on ait besoin de telles auto-jointures. Voici quelques exemples de relation nécessitant une auto jointure de tables :

- dans une table des employées, connaître le supérieur hiérarchique de tout employé
- dans une table de composants pour savoir quels sont les composants nécessaires à la réalisation d'autres composants...
- dans une table de personnes, retrouver l'autre moitié d'un couple marié...



PERSONNES (Numpers, NomPers, prenomPers, Numconjoint#)

MCD - Mariage

Pour connaître le nom de la personne N° 1 et de son conjoint, on écrira :

```sql
SELECT p1.nom, p2.nom
FROM personnes p1, personnes p2
WHERE p2.numPers = p1.numconjoint
AND p1.numpers = 1 ;
```

ou :

```sql
SELECT P1.nom, P2.nom
FROM personnes P1, JOIN personnes P2
ON P2.numPers = P1.numconjoint
WHERE P1.numpers = 1 ;
```

4. L'interrogation de données - Les index et les vues

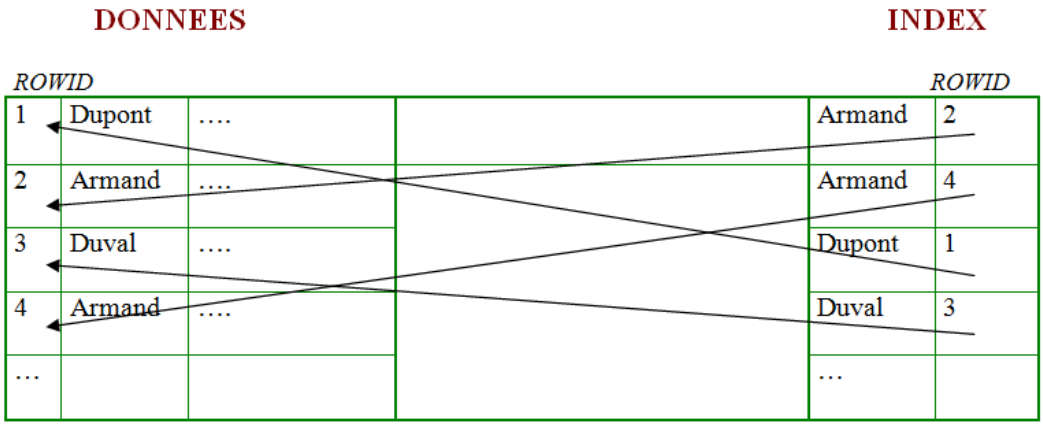
4.1. Interroger une base de données : Les index et les vues

La définition d'index et de vues relève du langage de définition de données (LDD) au même titre que la création de tables.

a) Les index

On crée des index sur les tables pour permettre un accès plus rapide à l'information. Un index concerne une ou plusieurs colonnes de la table et entraîne la création d'un espace comportant pour chaque valeur de la colonne ou du groupe de colonnes, le numéro du (ou des) tuple(s) correspondant(s).

L'illustration présente une table Client avec un index sur le nom.



Tables - Index

Remarque

- les données sont insérées de façon séquentielle dans la table Client au fur et à mesure des ajouts
- chaque ligne de la table est identifiée de façon unique par son identifiant appelé ROWID
- lorsqu'on crée un index sur le nom du client, les valeurs de la colonne indexée sont en permanence classées ; à chacune de ces valeurs, on fait correspondre le ROWID de la ligne de la table.

Un index va permettre d'éviter de balayer toute la table lors de la recherche d'une ligne répondant à un critère de sélection exprimé sur la colonne à indexer.

Exemple

```
1 SELECT *
2 FROM Client
3 WHERE Nom_cli = 'Armand' ;
```

Exécution sans index sur le nom de client : Le SGBDR scrute toute la table Client (bloc par bloc et ligne par ligne) pour retrouver les clients nommés 'Armand'.

Exécution avec index sur le nom de client : Le SGBDR passe dans ce cas par l'index qui le renverra vers les lignes concernées dans la table. Cette seconde solution est nettement plus rapide car le nombre de blocs parcourus est beaucoup moins important.

Création d'un index

Syntaxe

```
1 CREATE [UNIQUE] INDEX NomIndex
2 ON NomTable (NomColonne [ASC DESC] , NomColonne ...]);
```

- UNIQUE est à préciser si les doublons ne sont pas acceptés.
- Par défaut, le classement est croissant (ASC).
- Un index peut être composé de une à 16 colonnes.

Exemple

```
1 CREATE INDEX idxcli ON Client (Nom_cli) ;
```

Suppression d'un index

Syntaxe

```
1 DROP INDEX NomIndex ;
```

Colonnes à indexer

Conseil

- Les clés primaires : La création d'un index sur la clé primaire est automatique sous PostGreSQL.
- Les clés étrangères : Ces clés servent pour les jointures. C'est justement pour accélérer les opérations de jointure qu'un index est nécessaire sur les clés étrangères.
- Les colonnes servant souvent de critère de recherche : Ce sont les colonnes qui sont référencées dans une clause WHERE.
- Les colonnes servant souvent de critère de tri ou de groupement : Ce sont les colonnes qui sont référencées dans une clause ORDER BY ou GROUP BY.

Colonnes à ne pas indexer *Conseil*

Il est recommandé de ne pas indexer :

- les colonnes présentant peu de valeurs distinctes,
- les colonnes souvent modifiées.

Cas où l'index n'est pas utilisé *Remarque*

Bien qu'une colonne soit indexée, l'index ne sera pas utilisé si :

- la colonne indexée ne figure ni dans WHERE, ni dans ORDER BY et ni dans GROUP BY,
- on applique une fonction sur la colonne indexée :

```
SELECT *
FROM Client
WHERE UPPER(Nom_cli) LIKE 'P%';
```

- on teste l'inégalité entre la colonne indexée et une valeur :

```
SELECT *
FROM Client
WHERE Nom_cli != 'Dupont';
```

- on compare la colonne indexée avec un modèle de chaîne dont on ne connaît pas le début :

```
SELECT *
FROM client
WHERE Nom_cli LIKE '_A%';
```

Inconvénients des index

- L'insertion ou la suppression dans une table indexée de même que la modification de la valeur d'une colonne indexée va entraîner une mise à jour de l'espace d'index.

Les opérations DELETE et UPDATE sont néanmoins très souvent précédées par une recherche. Dans ce cas, l'index fait gagner du temps lors de la recherche et en fait perdre lors de l'opération proprement dite de modification ou suppression. Donc, globalement, l'index n'est pas vraiment pénalisant.

- L'opération INSERT ne fait pas de recherche préalable : d'éventuels index ne peuvent être que pénalisants.

Dans les cas de saisie massive initiale d'une table, on recommande de saisir la table d'abord et de créer les index après ; de cette façon, l'étape de saisie est plus rapide et l'espace disque occupé est moindre.

b) Les vues

Les vues sont des tables virtuelles créées à partir de requêtes SQL. Seule la définition de la vue (c'est à dire le texte de la requête) est stockée, l'ensemble de tuples correspondant n'est pas stocké contrairement aux tuples d'une table.

Une vue dans un SGBDR est assimilable à la notion de requête enregistrée sous Access mais offre plus de possibilités qu'une requête Access notamment au niveau de l'attribution des privilèges.

Remarque

- La clause ORDER BY est interdite dans la requête de création de la vue (elle n'a absolument aucun intérêt dans ce contexte : une table est un « sac de billes » non ordonnées).
- La clause WITH CHECK OPTION permet de garantir que les tuples insérés ou modifiés au travers de la vue sont compatibles avec la définition de la vue. Bien sûr, cette clause n'a de sens que si l'insertion ou la modification est possible.

Avec PostGreSQL cette option n'est pas disponible et les **vues ne sont accessibles qu'en lecture seule**.

Création d'une vue

Syntaxe

```
CREATE VIEW NomVue
AS
Requête
[WITH CHECK OPTION] ;
```

Exemple

TABLE CLIENT :

Num_cli	Nom_cli	Adr_cli	Cdp_cli	Vil_cli	Tel_cli
1	Dupont	69 rue de Turbigo	75003	PARIS	0143211234
2	Armand	05 rue Royale	75002	PARIS	0144332211
3	Duval	14 rue des Petits Champs	75002	PARIS	0142622222
4	Armand	06 rue de France	21000	DIJON	0380421312
5	Dupont	10 rue de Conte	75003	PARIS	0143211235
6	Petitjean	25 rue de Turin	80000	AMIENS	NULL
7	Durand	12 rue de la Paix	21000	DIJON	0349999555

VUE V_CLI_PAR :

Num_cli	Nom_cli	Tel_cli
1	Dupont	0143211234
2	Armand	0144332211
3	Duval	0142622222
5	Dupont	0143211235

VUE de table

- on effectue ici une projection et une sélection.
- la manipulation de la vue V_CLI_PAR permet de masquer les lignes des clients 4, 6 et 7 et certaines colonnes pour les clients "visibles".

```
CREATE VIEW V_CLI_PAR
AS
SELECT Num_clin Nom_cli, Tel_cli
FROM Client
WHERE Vil_cli = 'PARIS' ;
```

Pour utiliser la vue afin de consulter les clients parisiens, il suffit d'écrire :

```
SELECT *
FROM V_CLI_PAR;
```

Exemple

```
CREATE VIEW V_CLI_VILLE (V_ville, V_nb)
AS
SELECT Vil_cli, count(*)
FROM Client
GROUP BY Vil_cli;
```

Remarque

On n'est pas obligé de spécifier les noms des colonnes de la vue si la liste de sélection de la requête ne comporte ni expression, ni fonction, ni colonnes en double.

Si on ne spécifie pas de nom, les colonnes de la vue héritent des noms des champs indiqués dans la liste de sélection (sans le nom de la table en préfixe). Il faut soit ne préciser aucun des noms de colonnes de la vue, soit les préciser tous.

Intérêt des vues

- Une vue permet d'assurer la confidentialité de certaines données : avec les vues, il est possible de mettre uniquement certaines colonnes et/ou certaines lignes à disposition de l'utilisateur (cf. : exemple ci-dessus).
- Une vue peut également être utilisée pour simplifier la formulation de requêtes lorsque celles-ci sont complexes.

Exemple

La vue V_COMM permet de simplifier la formulation d'une requête complexe.

```
CREATE VIEW V_COMM
AS
SELECT C.num_com, C.dat_com, L.num_lig, L.num_art, A.des_art, L.qte_lig
FROM Commande C, Lg_comm L, Article A
WHERE C.num_com = L.num_com
AND L.num_art = A.num_art ;
```

- trois tables sont en jeu : LG_COMM, COMMANDE et ARTICLE,
- on effectue une projection et des jointures
- on écrit ensuite simplement la requête suivante pour avoir toutes les info sur une commande donnée (ici la 'C123'). Ce qui donne la requête suivante :

```
SELECT *
FROM V_COMM
WHERE num_com='C123';
```

Mise à jour (modification, insertion, suppression) à travers une vue (pas sous PostGreSQL)

Il n'est pas possible d'effectuer des mises à jour à partir de n'importe quelle vue.

La vue n'est pas modifiable dans les cas suivants :

- si elle ne contient pas tous les attributs définis comme NON NULL dans la table interrogée,
- ou si elle contient une jointure
- ou si elle contient une fonction agrégat.
- ou enfin, si elle ne contient un "SELECT DISTINC".

Exemple

Soit la vue V_CLI_DIJON :

```
CREATE VIEW V_CLI_DIJON
AS
SELECT *
FROM Client
WHERE Vil_cli = 'DIJON' ;
```

Modification à travers V_CLI_DIJON :

```
UPDATE V_CLI_DIJON
SET Tel_cli = '0342123456'
WHERE Num_cli = 4 ;
```

Insertion d'un nouveau client à partir de V_CLI_DIJON :

```
1 INSERT INTO V_CLI_DIJON VALUES
2 (8, 'Portier', '03 rue de Lille', 75003, 'PARIS', NULL) ;
3 1 ligne créée.
```

Attention

Si la vue V_CLI_DIJON avait été définie avec la clause WITH CHECK OPTION, l'insertion n'aurait pas été acceptée :

```
1 INSERT INTO V_CLI_DIJON VALUES (8, 'Portier', '03 rue de Lille', 75003, 'PARIS',
  NULL) ;
2 => message d'erreur :
3 ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE
```

En effet, on essaye d'insérer un client habitant à Paris alors que la vue est créée exclusivement avec des clients habitant à Dijon !

Modification d'une vue

```
1 REPLACE VIEW nomVue AS
2 requête ...
```

On écrit le plus souvent, dans la création d'une vue :

```
1 CREATE OR REPLACE VIEW nomVue AS
2 requête...
```

... de manière à faciliter les corrections éventuelles.

Suppression d'une vue

```
1 DROP VIEW NomVue ;
```

4.2. Exercice : Cas Infosynaps

La société InfoSynaps met à disposition de ses collaborateurs un espace de communication et d'information sous la forme d'un site intranet.

Le site intranet est constitué de différentes rubriques dont l'accès est régi suivant le type d'utilisateur. Par exemple, M. Pierre DUBOIS, intervenant en clientèle, dispose de la rubrique « Formation technique », alors que M. Patrick VENTOUT, commercial, n'y a pas accès. En revanche, M. VENTOUT peut utiliser la rubrique « Gestion clients et prospects ». Par ailleurs, M. DUBOIS et M. VENTOUT ont tous les deux accès à la rubrique « Frais de déplacement ». Une base de données dont le schéma relationnel est fourni ci-après permet d'assurer la gestion des accès aux différentes rubriques du site intranet.

TYPE_UTILISATEUR (code, libellé)
code : clé primaire
Exemple : Le type codé « IC » porte le libellé « Intervenant en clientèle ».

UTILISATEUR (login, nom, prénom, motDePasse, codeTypeUtil)
login : clé primaire
codeTypeUtil : clé étrangère en référence à code de TYPE_UTILISATEUR
Exemple : L'utilisateur de login « DUBOISP » correspond à « DUBOIS Pierre ».

RUBRIQUE (ref, intitulé)
ref : clé primaire
Exemple : La rubrique « FT » correspond à l'intitulé « Formation technique ».

ACCES (codeTypeUtil, refRubrique, nbVisites)
codeTypeUtil, refRubrique : clé primaire
codeTypeUtil : clé étrangère en référence à code de TYPE_UTILISATEUR
refRubrique : clé étrangère en référence à ref de RUBRIQUE
Exemples : La rubrique « FT » est accessible par le type utilisateur codé « IC » et enregistre actuellement 100 visites pour ce type d'utilisateur. La rubrique « GCP » est accessible par le type utilisateur codé « COM » (200 visites) et par le type d'utilisateur codé « IC » (92 visites).

MLR - Infosynaps

Les contraintes d'intégrité de clé primaire et référentielle ont été omises lors de la création du schéma de la table ACCES qui contient maintenant des données.

Question 1

Écrire l'(les) ordre(s) SQL nécessaire(s) à l'ajout des contraintes d'intégrité absentes.

On considérera que la contrainte NOT NULL a été déclarée sur les colonnes refRubrique et codeTypeUtil lors de la création de la table ACCES.

Écrire les requêtes SQL permettant de réaliser les opérations suivantes :

Question 2

A. Obtenir la référence et l'intitulé des rubriques accessibles par l'utilisateur de login "DUBOISP".

Question 3

B. Mettre à jour la table ACCES suite à la visite de la rubrique de référence "FT" par un utilisateur de type codé "IC".

Question 4

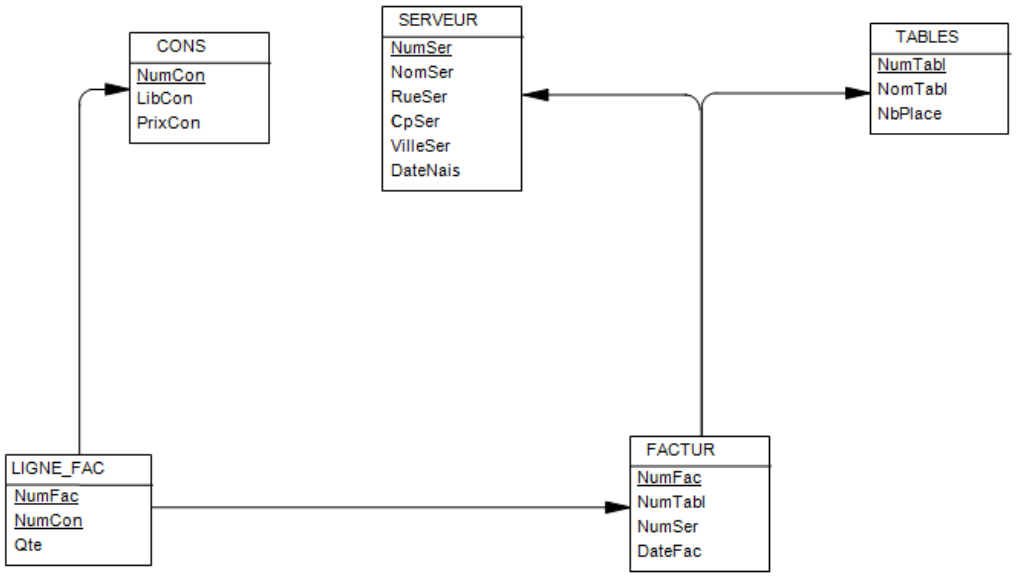
A. Créer la vue appelée "vFrequentationRubrique" qui donne par rubrique le nombre total de visites.

Les colonnes de la vue seront nommées ref, intitulé et nbTotalVisites.

Question 5

B. En utilisant la vue "vFrequentationRubrique", afficher l'(les) intitulé(s) de rubrique présentant le plus grand nombre total de visites.

4.3. Exercice : Cas : Au père tranquille



MLR - Au père tranquille

Question 1

On effectue souvent des recherches à partir du nom du serveur. On vous demande de trouver une solution pour accélérer ces requêtes et d'écrire l'instruction correspondante.

Question 2

On effectue souvent des jointures entre facture et serveur. On vous demande de trouver une solution pour accélérer ces requêtes et d'écrire l'instruction correspondante.

Question 3

• Créer une vue permettant d'afficher le chiffre d'affaires de la journée (vous utiliserez la fonction now() ou current_date() qui rend la date et l'heure système).

Vérifier que la vue est correcte. Pour cela, insérer les données nécessaires pour avoir au moins une facture (et les lignes de factures correspondantes) pour la date du jour (facultatif).

Question 4

Créer une vue permettant d'obtenir les factures établies par le serveur Pilou.

Vérifier que la vue est correcte (facultatif).

Question 5

Créer une vue permettant d'obtenir le chiffre d'affaire généré par les boissons de type "Café".

4.4. Exercice : Cas Tintinophile

Soit le modèle relationnel suivant :

ALBUM (**NOALB**, TITREALB, DATEALB)
PAYS (**NOPAYS**, NOMPAYS)
DEROULER (#**NOALB**, #**NOPAYS**)
PERSONNAGES (**NOPERS**, NOMPERS, PRENOMPERS, FONCTPERS, SEXEPERS, GENTILPERS)
PARTICIPER (#**NOALB**, #**NOPERS**)
JURONS (**NOJURON**, NOMJURON)
PRONONCER (#**NOPERS**, #**NOJURON**, #**NOALB**, **NOPAGE**)

MLR - Tintinophile

Question 1

Retrouver le MCD correspondant :

Question 2

Créer la base de données sous PostGreSQL et intégrer le script suivant :

Cas Tintinophile - Script SQL [cf. script_tintin.zip]

Question 3

Ajouter la contrainte de clé primaire de "dérouler".

Question 4

Un album doit avoir obligatoirement un nom et celui ci doit être unique.

Question 5

Quand on supprime un personnage, on doit également supprimer ses participations aux différents albums.

Écrire et tester les requêtes qui permette de répondre aux questions suivantes :

Question 6

Combien de jurons la Castafiore prononce-t-elle ?

Question 7

Classer des personnages par nombre de jurons prononcés croissant ?

Question 8

Dans quel album le capitaine HADDOCK fait-il son apparition ?

Question 9

Quels sont les numéros des personnages qui ne jurent jamais ?

Question 10

Quels sont les albums (tous les renseignements) où il existe au moins 3 pays concernés par cet album (utiliser 2 formulations : sans puis avec un EXISTS).

Question 11

Quels sont les albums (tous les renseignements) où ne figurent que des personnages qui jurent ?

Question 12

Quels sont les personnages (numéro et nom) qui apparaissent dans tous les albums ?