Troisième partie III

Le langage SQL DML (1)

Plan du cours

Le langage SQL

- SQL : *Structured Query Language*
  - Langage d'interrogation des SGBD
  - Une instruction SQL constitue une requête SQL
  - 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 est composé de deux sous-langages
  1. DDL (*Data Définition Language*)
  2. DML (*Data Manipulation Language*)

Le langage SQL

- Le language SQL DML comporte 2 grandes classes de fonctions :
  1. l'extraction de données
  2. la modification de données
- L'instruction qui permet de faire les extractions ou sélections est l'instruction **SELECT**

L'instruction SELECT

- L'instruction **SELECT** contient 3 parties principales :
  1. la clause **SELECT** précise les valeurs (nom des colonnes) qui constitue chaque ligne du résultat attendu
  2. la clause **FROM** indique la table ou les tables à partir desquels le résultat doit être extrait
  3. la clause **WHERE** spécifie la condition de sélection que doit être satisfaite par les lignes du résultat
- Le résultat d'une requête **SELECT** ou SFW (**SELECT FROM WHERE**) est une table fictive qui sera considérée comme s'affichant à l'écran

Les différents types de tables

- Il existe plusieurs types de tables dans un SGBD :
  1. Les tables de bases : Les tables que nous avons manipulée jusqu'à présent. Elles sont stockées de manière pérenne dans la mémoire externe et accessibles à tous les utilisateurs.
  2. Les résultats des requêtes SFW : La table résultat de l'éxécution d'une requête. Ce type de table est volatile.
  3. Les tables dérivées : Elles servent à stocker le résultat d'une requête pendant une courte durée (quelques jours). Leurs utilisations sont limitées à un utilisateur. On les appelle aussi *snapshot*.
  4. Les vues : Une vue est une table virtuelle dont le contenu est défini comme le résultat d'une requête SFW. Seule la définition est stockée. Son contenu est recalculé à chaque consultation.

L'exemple fil rouge du cours

- Soit le schéma relationel d'un site de ventes en ligne
  - CLIENT (<u>NCLI</u>, NOM, ADRESSE, LOCALITE, CAT, COMPTE)
  - PRODUIT (<u>NPRO</u>, LIBELLE, PRIX, QSTOCK)
  - COMMANDE (<u>NCOM</u>, NCLI, DATECOM)
  - DETAIL (<u>NCOM</u>, <u>NPRO</u>, QCOM)

Requêtes élémentaires

- Une requête simple consiste à demander l'affichage des valeurs de certaines colonnes A des lignes d'une table R
- Equivalence en algèbre relationnelle : la projection notée $R[A]$

Requêtes élémentaires

Soit la requête :

```
SELECT NCLI, NOM, LOCALITE
FROM CLIENT ;
```

sélectione les valeurs NCLI, NOM et LOCALITE des lignes de la table CLIENT.

La réponse à cette requête se présenterait comme suit à l'écran :

NCLI	NOM	LOCALITE
B062	GOFFIN	Namur
B112	HANSENNE	Poitier
B332	MONTI	Genève
B512	GILLET	Toulouse
...	...	...

La sélection de toutes les colonnes s'écrira :

```
SELECT *
FROM CLIENT ;
```

Requêtes élémentaires

- La sélection consiste à extraire les lignes d'une table respectant une certaine propriété
- Equivalence en algèbre relationnelle : La sélection notée $R : C$
- La condition de sélection utilise les opérateurs de comparaison suivants :
  - $=$: l'égalité
  - $>$: plus grand que
  - $<$: plus petit que
  - $<>$: différent de
  - $>=$: plus grand ou égal
  - $<=$: plus petit ou égal
- Remarques
  - L'ordre naturel est utilisé dans les comparaisons
  - Les chaînes de caractère sont entre ' et '
  - Le format des dates utilisées est le suivant : '2013-09-01'

Requêtes élémentaires

Soit la requête :

```
SELECT NCLI, NOM
FROM CLIENT
WHERE LOCALITE = 'Toulouse' ;
```

sélectione les valeurs NCLI et NOM des lignes de la table CLIENT qui habitent Toulouse.

La réponse est la table suivante :

NCLI	NOM
B512	GILLET
C003	AVRON
D063	MERCIER
F011	PONCELET
K729	NEUMAN

Requêtes élémentaires

- Problème
  - Une requête monotable peut contenir autant de lignes qu'il y de ligne vérifiant la condition dans la table
  - Il se peut donc, si la sélection ne contient aucune clé, que le résulat possède plusieurs lignes identiques

- Exemple avec duplication

```
SELECT LOCALITE
FROM CLIENT
WHERE CAT = 'C1' ;
```

La réponse est la table suivante :

LOCALITE
Poitiers
Namur
Poitiers
Namur
Namur

- Exemple sans duplication

```
SELECT DISCTINCT LOCALITE
FROM CLIENT
WHERE CAT = 'C1' ;
```

La réponse est la table suivante :

LOCALITE
Poitiers
Namur

Requêtes élémentaires

Supposons que l'on veuille la liste des numéros de clients ayant passés au moins une commande. On écrirait la requête suivante :

```
SELECT NCLI
FROM COMMAND ;
```

Cependant, le nombre d'éléments du résulats de cette requête n'est pas égal à celui des clients qui ont passé une commande. Les numéros clients sont dupliqués autant de fois qu'ils ont passé de commande. Il faut donc mieux écrire :

```
SELECT DISTINCT NCLI
FROM COMMAND ;
```

Remarque

Si la clause SELECT cite tous les composants d'un identifiant de la table, l'unicité des lignes résultats est garantie. Il est donc inutil d'utiliser DISTINCT.

Requêtes élémentaires

- Une condition élémentaire peut porter sur
  - la présence de valeur NULL :

    ```
    CAT IS NULL
    CAT IS NOT NULL
    ```

  - sur l'appartenance à une liste :

    ```
    CAT IN ('C1', 'C2', 'C3')
    LOCALITE NOT IN ('Toulouse', 'Namur', 'Breda')
    ```

  - sur un intervalle :

    ```
    COMPTE BETWEEN 1000 AND 4000
    CAT NOT BETWEEN 'B2' AND 'C1'
    ```

  - sur la présence de certains caractères dans une valeur :

    ```
    CAT LIKE '_1'
    ADRESSE LIKE '\%Neuve\%'
    ```

    Ces dernières conditions utilisent des masques.

Requêtes élémentaires

- Les symboles utilisés dans les masques :
  - "_" désigne un caractère quelconque
  - "%" désigne un suite de caractères éventuellement vide
- Pour utiliser les caractères "_" et "%" en tant que caractère il faut les préfixe rd'un caratère spécial

  ```
  LIBELLE LIKE '\%\$\_CHENE\%' ESCAPE '\$'
  ```

- Un masque peut s'appliquer à une date

  ```
  DATE LIKE '\%2009\%'
  ```

- Les conditions admettent la forme négative

  ```
  ADRESSE NOT LIKE '\%NEUVE\%'
  ```

Requêtes élémentaires

- La condition de la clause WHERE peut être compossée d'une expression booléenne
- Exemple :

  ```
  SELECT NOM, ADRESSE, COMPTE
  FROM CLIENT
  WHERE LOCALITE = 'Toulouse' AND COMPTE < 0 ;
  ```

- Soit les expressions P et Q
  - **WHERE P AND Q** sélectionne les lignes qui vérifient simultanément P et Q
  - **WHERE P OR Q** sélectionne les lignes qui vérifient P ou Q ou les deux
  - **WHERE NOT P** sélection les lignes qui ne vérifient pas P
- L'usage de paranthèses permet d'exprimer des expressions plus complexes

  ```
  WHERE COMPTE < 0
     AND (CAT = 'C1' OR LOCALITE = 'Paris')
  ```

Données extraites et données dérivées

- En SQL, il est possible de spécifier des données dérivées ou des constantes
- Exemple :

```
SELECT 'TVA de ', NPRO, ' = ',0,21*PRIX*QSTOCK
FROM PRODUIT
WHERE QSTOCK > 500 ;
```

Le résultat est le suivant :

TVA de	NPRO	=	0,21*PRIX*QSTOCK
TVA de	CS264	=	67788
TVA de	PA45	=	12789
TVA de	PH222	=	37770.6
TVA de	PS222	=	47397

Données extraites et données dérivées

- Par défaut les colonnes du résultat prennent le nom utilisé dans la clause SELECT
- Pour utiliser un autre nom ou alias il fait utiliser la clause AS
- Exemple :

```
SELECT NPRO AS PRODUIT , 0,21*PRIX*QSTOCK AS TVA
FROM PRODUIT
WHERE QSTOCK > 500 ;
```

Le résultat est le suivant :

PRODUIT	TVA
CS264	67788
PA45	12789
PH222	37770.6
PS222	47397

Les fonctions SQL

- Il s'agit des 4 opérateurs arithmétiques : $+$, $-$, $*$ et $/$
- D'autres fonctions pourront être présentes en fonction des SGBD
  - Par exemple : exponentielle, logarithme, trigonométrie, etc.

Les fonctions SQL

- **CHAR_LENGTH(*s*)** : donne le nombre de caractères de la chaîne *s*
- **POSITION(*s1* IN *s2*)** : donne la position de la chaîne *s1* dans la chaîne *s2* ; 1 si *s1* est vide et 0 si *s1* n'apparaît pas dans *s2*
- *s1* || *s2* : construit une chaîne composée de la concaténation de *s1* et *s2*
- **LOWER(*s*)** : convertit la chaîne *s* en minuscule
- **UPPER(*s*)** : convertit la chaîne *s* en majuscule
- **SUBSTRING(*s* FROM *l* FOR *L*)** : construit une chaîne de longueur *L* à partir de la chaîne *s* débutant à l'indice *l*
- **TRIM(*e c* FROM *s*)** : supprime les caractères *c* à l'extrémité *e* de la chaîne *s* ; *e* peut prendre pour valeur LEADING, TRAILING et BOTH
  - Exemple :

```
TRIM (BOTH ' ' FROM ADRESSE || ' ' || UPPER(LOCALITE)
```

Les fonctions SQL

- **BIT_LENGTH(*s*)** : donne le nombre de bits de la chaîne *s*
- **OCTET_LENGTH(*s*)** : donne le nombre d'octets occupés par la chaîne de bits *s*

Les fonctions SQL

- **CAST(*v* AS *t*)** : convertit la valeur *v* selon le type *t*
- Exemple :

```
CAST (DATECOM AS CHAR(12)
CLI.COMPTE - CAST(QCOM*PRIX AS DECIMAL(9,2))
```

Les fonctions SQL

- **EXTRACT(*u* FROM *dt*)** : donne, sous la forme numérique, le composant *u* de ma valeur temporelle *dt* ; les valeurs de *u* sont : YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
- Exemple :

```
EXTRACT (YEAR FROM DATECOM) + 1
EXTRACT (HOUR FROM CURRENT_TIME) > 18
```

Les fonctions SQL

- Les fonctions de sélection renvoient une valeur choisie parmi plusieurs
- Exemple :

```
SELECT NCLI,
   CASE SUBSTRING(CAT FROM 1 FOR 1)
      WHEN 'A' then 'bon'
      WHEN 'B' then 'moyen'
      WHEN 'C' then 'occasionnel'
      ELSE 'inconnu'
   END, LOCALITE
FROM CLIENT ;
```

Les registres du système

- Le SGBD fournit des valeurs courantes relatives à l'utilisateur au moment de l'éxécution de la requête :
  - **CURRENT_USER** : l'utilisateur courant
  - **CURRENT_DATE** : la date courante
  - **CURRENT_TIME** : l'heure courante
  - **CURRENT_TIMESTAMP** : date + heure courante
- Exemple :

```
SELECT NPRO, LIBELLE, QSTOCK
FROM CLIENT
WHERE QSTOCK < 0
   AND EXTRACT(DAY FROM CURREN_DATE) = 1 ;
```

Les fonctions agrégatives (ou statistiques)

- En SQL, il existe des fonctions qui donnent une valeur agrégée calculée pour les lignes sélectionnées :
  - **COUNT(*)** : donne le nombre de lignes sélectionnées
  - **COUNT(***colonne***)** : donne le nombre de valeurs de la colonne
  - **AVR(***colonne***)** : donne la moyenne des valeurs de la colonne
  - **SUM(***colonne***)** : donne la somme des valeurs de la colonne
  - **MIN(***colonne***)** : donne le minimum des valeurs de la colonne
  - **MAX(***colonne***)** : donne le maximum des valeurs de la colonne
- Remarque :
  - La colonne peut être remplacée par une expression arithmétique
  - Exemple :

```
SELECT SUM(QSTOCK*PRIX)
FROM PRODUIT
WHERE LIBELLE LIKE '%SAPIN%'
```

Les fonctions agrégatives (ou statistiques)

- Donner la répartition (moyenne, écart maximum, nombre) des montants des comptes des clients habitant Namur

```
SELECT 'Namur', AVR(COMPTE) AS MOYENNE,
   MAX(COMPTE) - MIN(COMPTE) AS ECART_MAX,
   COUNT(*) AS NOMBRE
FROM CLIENT
WHERE LOCALITE = 'Namur'
```

Le résultat est la table suivante :

Namur	MOYENNE	ECART_MAX	NOMBRE
Namur	-2520	4580	4

Les fonctions agrégatives (ou statistiques)

- Supposons que l'on veuille obtenir le nombre de clients ayant passés au moins une commande

- Requête erronée

```
SELECT COUNT(NCLI)
FROM COMMAND ;
```

- Requête correcte

```
SELECT COUNT(DISCTINCT NCLI)
FROM COMMAND ;
```

Les fonctions agrégatives (ou statistiques)

- Question
  - Quelle est la valeur retournée par une fonction agrégative produit à partir d'un ensemble vide ?
- Réponse
  - 0 pour la fonction COUNT et inconnu pour les autres
- Exemple :

```
SELECT COUNT(*) AS NOMBRE, SUM(COMPTE) AS SOMME
   MAX(CAT) AS MAX
FROM CLIENT
WHERE LOCALITE = 'Alger' ;
```

Le résultat sera le suivant :

NOMBRE	SOMME	MAX
0	<null>	<null>

Les sous-requêtes

- Nous avons étudié que des requêtes qui extraient des données d'une table
- Il peut être intéressant d'extraire des lignes en fonction de leur liaison avec les autres tables
- On parle alors de condition d'association ou de joiture

Les sous-requêtes

Supposons que l'on veuille obtenir les commandes des clients habitant Namur.

1. On peut retrouver les numéros des clients habitant Namur en exécutant la requête :

```
SELECT NCLI
FROM CLIENT
WHERE LOCALITE = 'Namur' ;
```

Le résultat de la requête est le suivant :

NCLI
B062
C123
L422
S127

2. Il est alors facile de retrouver les commandes des clients de Namur :

```
SELECT NCOM, DATECOM
FROM COMMAND
WHERE NCLI IN ('B062', 'C123', 'L422', 'S127')
```

Les sous-requêtes

- Cette procédure n'est pas pratique
- On pourra donc écrire :

```
SELECT NCOM, DATECOM
FROM COMMANDE
WHERE NCLI IN (SELECT NCLI
                 FROM CLIENT
                WHERE LOCALITE = 'Namur') ;
```

Références multiples à une même table

- Une sous-requête peut être définie sur la même table que la requête qui la contient
- Exemple :
  - Quels sont les clients qui habitent dans la même localité que le client numéro B512 ?

```
SELECT *
FROM CLIENT
WHERE LOCALITE IN (SELECT LOCALITE
                   FROM CLIENT
                   WHERE NCLI = 'B512') ;
```

Références multiples à une même table

- Un autre exemple
  - Quelles commandes spécifient une quantité de PA60 inférieure à la commande numéro 30182 pour le même produit ?

```
SELECT NCOM
FROM DETAIL
WHERE NRPO = 'PA60'
   AND QCOM < (SELECT QCOM
               FROM DETAIL
               WHERE NPRO = 'PA60'
                  AND NCOM = '30182') ;
```

- Ici, aucune ambiguité les valeurs de NRPO et NCOM sont celles de la table DETAIL de la seconde sous-requête

Références multiples à une même table

- Dans le cas d'ambiguité de noms d'attribut, il est possible :
  1. De préfixer le nom de l'attribut par celui de la relation

```
SELECT PRODUIT.NPRO
FROM PRODUIT
WHERE PRODUIT.NPRO IN (
   SELECT DETAIL.NPRO
   FROM DETAIL) ;
```

  2. D'utiliser des alias

```
SELECT P.NPRO
FROM PRODUIT AS P
WHERE P.NPRO IN (
   SELECT D.NPRO
   FROM DETAIL AS D) ;
```

Références multiples à une même table

- Si la sous-requête renvoie une seule ligne, il est permis d'utiliser les opérateurs de comparaison classique
- Exemple :

```
SELECT *
FROM CLIENT
WHERE COMPTE > (SELECT COMPTE
                FROM CLIENT
                WHERE NCLI = 'C400') ;
```

Références multiples à une même table

- Il est intéressant de sélectionner les lignes d'une table qui sont associées, non pas à au moins une des lignes d'une autre table qui vérifie une certaine condition, mais à un nombre défini de ces lignes
- Exemple :
  - Quels sont les commandes qui possèdent au moins 3 détails ?

```
SELECT NCOM, DATECOM, NCLI
FROM COMMANDE C
WHERE (SELECT COUNT(*)
       FROM DETAIL AS D
       WHERE D.NCOM = C.NCOM) >= 3 ;
```

Les quantifieurs ensemblistes

- Qu'est ce que les quantifieurs ensemblistes ?
  - Les quantifieurs ensemblistes permettent d'imposer qu'un ensemble défini possède au moins un élément ou qu'au moins un élément satisfasse une condition particulière
- Les quantifieurs SQL sont les suivants :
  - **EXISTS**
  - **ANY**
  - **ALL**
- Équivalence :
  - IN $\equiv$ = ANY
  - NOT IN $\equiv$ <> ALL

Les quantifieurs ensemblistes

- Une condition peut porter sur l'existence (**EXITS**) ou l'inexistence (**NOT EXISTS**) d'au moins une des lignes dans le résulat d'une sous requête
- Exemple :
  - Quels sont les produits qui ont déjà été commandés ? (i.e., ceux qui sont dans DETAILS)

```
SELECT NPRO, LIBELLE
FROM PRODUIT AS P
WHERE EXISTS (SELECT *
              FROM DETAIL AS D
              WHERE D.NPRO = P.NPRO) ;
```

Les quantifieurs ensemblistes

- Le quantifieur **ALL** permet de comparer une valeur à celles d'un ensemble défini par une sous-requête
- **ALL** signifie que tous les éléments de l'ensemble doivent satisfaire la comparaison
- Exemple :
  - Donner les commandes spécifiant la plus petite quantité du produit PA60 ?

```
SELECT DISTNCT NCOM
FROM DETAIL
WHERE QCOM <= ALL (SELECT QCOM
                   FROM DETAIL
                   WHERE NPRO = 'PA60')
  AND NPRO  = 'PA60' ;
```

  - La condition s'interprète :
    - la valeur de QCOM (de la ligne de DETAIL courante) est inférieure ou égale à **chacun des éléments** de la table DETAIL relatifs au produit 'PA60'

Les quantifieurs ensemblistes

- Le quantifieur **ANY** permet de comparer une valeur à celles d'un ensemble défini par une sous-requête
- **ANY** signifie qu'au moins un des éléments de l'ensemble doit satisfaire la comparaison
- Exemple :
  - Donner le détail des commandes de PA60 dont la quantité n'est pas minimale ?

```
SELECT *
FROM DETAIL
WHERE QCOM > ANY (SELECT QCOM
                  FROM DETAIL
                  WHERE NPRO = 'PA60')
   AND NPRO = 'PA60' ;
```

  - La condition s'interprète :
    - la valeur de QCOM (de la ligne de DETAIL courante) est supérieure à **au moins un des éléments** de la table DETAIL relatifs au produit 'PA60'

Exercice 1

- Exprimer en SQL les requêtes suivantes :
  1. Afficher la liste des localités dans lesquelles il existe au moins un client.
  2. Afficher le numéro, le nom et la localité des clients de catégorie C1 n'habitant pas Toulouse
  3. Donner le numéro, le nom et le compte des clients de Poitiers et de Bruxelles dont le compte est positif
  4. Quelles catégories de clients trouve-t-on à Toulouse ?

Exercice 1

1. Afficher la liste des localités dans lesquelles il existe au moins un client

```
SELECT DISTINCT LOCALITE FROM CLIENT ;
```

2. Afficher le numéro, le nom et la localité des clients de catégorie C1 n'habitant pas Toulouse

```
SELECT NCLI, NOM, LOCALITE FROM CLIENT
WHERE CAT = 'C1' AND LOCALITE <> 'Toulouse' ;
```

3. Donner le numéro, le nom et le compte des clients de Poitiers et de Bruxelles dont le compte est positif

```
SELECT NCLI, NOM, COMPTE FROM CLIENT
WHERE LOCALITE IN ('Poitiers', 'Bruxelles') AND COMPTE > 0 ;
```

4. Quelles catégories de clients trouve-t-on à Toulouse ?

```
SELECT DISTINCT CAT FROM CLIENT
WHERE LOCALITE = 'Toulouse' AND CAT IS NOT NULL ;
```

Exercice 2

- Exprimer en SQL les requêtes suivantes :
  1. Afficher le numéro, le nom et la localité des clients dont le nom précède alphabétiquement la localité ou ils résident
  2. Afficher les localités des clients qui commande le produit CS464
  3. Quels sont les produits en sapin qui font l'object d'une commande ?
  4. Donner la valeur totale des stocks sans tenir compte des commandes en cours.

Exercice 2

1. Afficher le numéro, le nom et la localité des clients dont le nom précède alphabétiquement la localité ou ils résident

```
SELECT NCLI, NOM, LOCALITE FROM CLIENT WHERE NOM < LOCALITE
```

2. Afficher les localités des clients qui commande le produit CS464

```
SELECT DISTINCT LOCALITE FROM CLIENT
WHERE NCLI IN (SELECT NCLI FROM COMMANDE
  WHERE NCOM in (SELECT NCOM FROM DETAIL WHERE NPRO = 'CS464')) ;
```

3. Quels sont les produits en sapin qui font l'object d'une commande ?

```
SELECT NRPO FROM PRODUIT WHERE LIBELLE LIKE '%SAPIN%'
   AND NPRO IN (SELECT NPRO FROM DETAIL) ;
```

4. Donner la valeur totale des stocks sans tenir compte des commandes en cours.

```
SELECT SUM(QSTOCK*PRIX) AS TOTAL FROM PRODUIT ;
```

Exercice 3

- Exprimer en SQL les requêtes suivantes :
  1. Combien y a t-il de commandes spécifiant un (ou plusieurs) produit(s) en acier ?
  2. Afficher le numéro et le nom des clients qui n'ont pas commandé de produits en sapin.
  3. Quels sont les produits (numéros et libellé) qui n'ont pas été commandés en 2008 ?

Exercice 3

1. Combien y a t-il de commandes spécifiant un (ou plusieurs) produit(s) en acier ?

```
SELECT COUNT(*) FROM COMMANDE WHERE NCOM IN
  (SELECT NCOM FROM DETAIL
   WHERE NRPO in (SELECT NPRO FROM PRODUIT
                  WHERE LIBELLE LIKE '%ACIER%')) ;
```

2. Afficher le numéro et le nom des clients qui n'ont pas commandé de produits en sapin.

```
SELECT NCLI, NOM FROM CLIENT WHERE NCLI NOT IN
  (SELECT NCLI FROM COMMANDE WHERE NCOM IN
    (SELECT NCOM FROM DETAIL WHERE NPRO IN
      (SELECT NPRO FORM PRODUIT WHERE LIBELLE LINKE '%SAPIN%'))) ;
```

3. Quels sont les produits (numéros et libellé) qui n'on pas été commandés en 2008 ?

```
SELECT NPRO, LIBELLE FROM PRODUIT WHERE NPRO NOT IN
  (SELECT NPRO FORM DETAIL WHERE NCOM IN
    (SELECT NCOM FROM COMMANDE WHERE DATECOM LIKE '%2008%')) ;
```

Exercice 4

- Exprimer en SQL les requêtes suivantes :
  1. Rechercher les clients qui ont commandé tous les produits.
  2. Rechercher les localités dont aucun client n'a passé de commande.
  3. Rechercher les localités dont tous les clients ont passé au moins une commande.

Exercice 4

1. Rechercher les clients qui ont commandé tous les produits.

```
SELECT NCLI FROM CLIENT C
WHERE NOT EXISTS (SELECT * FROM PRODUIT WHERE NPRO NOT IN
  (SELECT NPRO FROM DETAIL WHERE NCOM IN
    (SELECT NCOM FORM COMMANDE WHERE NCLI = C.NCLI)) ;
```

2. Rechercher les localités dont aucun client n'a passé de commande.

```
SELECT DISTINCT LOCALITE FROM CLIENT WHERE LOCALITE NOT IN
  (SELECT LOCALITE FROM CLIENT C WHERE EXISTS
    (SELECT * FROM COMMANDE WHERE NCLI = C.NCLI)) ;
```

3. Rechercher les localités dont tous les clients ont passé au moins une commande.

```
SELECT DISTINCT LOCALITE FROM CLIENT C WHERE NOT EXISTS
  (SELECT * FROM CLIENT WHERE LOCALITE = C.LOCALITE
    AND NCLI NOT IN (SELECT NCLI FROM COMMANDE)) ;
```