INFO0009 - Bases de données

Répétition 7 : Le langage SQL (2)

Maxime Meurisse (slides de Remy Vandaele)
m.meurisse@student.uliege.be

Année académique 2020-2021

Création de tables

Format général :

```
CREATE TABLE IF NOT EXISTS <NOM_REL.>(
    <nom_att1> <type_att1> <option_att1>,
    ...
    <nom_attN> <type_attN> <option_attN>,
    PRIMARY KEY(<nom_attI>),
    FOREIGN KEY(<nom_attJ>) REFERENCES <REL.>(<nom_attFJ>),
    ...
    FOREIGN KEY(<nom_attK>) REFERENCES <REL.>(<nom_attFK>)
) ENGINE=InnoDB;
```

Type : INT, BIGINT, VARCHAR, DATE, ...

Options : NOT NULL, PRIMARY KEY, AUTO_INCREMENT, ...

Exercices

Créer les tables suivantes :

- Client(id_client, prenom_client, nom_client, nationalite_client)
- Commande(id_commande,#id_client, date_commande)
- Produit(id_produit, nom_produit)
- Employé(id_employé, nom_employé, prenom_employé)
- DetailsCommande(#id_commande,# id_produit, #id_employé, quantite, prix_piece)
- Paiement(#id_commande, montant)

Création de tables

Client(<u>id_client</u>, prenom_client, nom_client, nationalite_client)

```
CREATE TABLE IF NOT EXISTS Client(
```

Création de tables

Client(<u>id_client</u>, prenom_client, nom_client, nationalite_client)

```
CREATE TABLE IF NOT EXISTS Client(
    id_client INT AUTO_INCREMENT,
```

Création de tables

Client(<u>id_client</u>, prenom_client, nom_client, nationalite_client)

```
CREATE TABLE IF NOT EXISTS Client(
    id_client INT AUTO_INCREMENT,
    prenom_client VARCHAR(50) NOT NULL,
    nom_client VARCHAR(50) NOT NULL,
    nationalite_client VARCHAR(50) NOT NULL,
```

Création de tables

Client(<u>id_client</u>, prenom_client, nom_client, nationalite_client)

```
CREATE TABLE IF NOT EXISTS Client(
    id_client INT AUTO_INCREMENT,
    prenom_client VARCHAR(50) NOT NULL,
    nom_client VARCHAR(50) NOT NULL,
    nationalite_client VARCHAR(50) NOT NULL,
    PRIMARY KEY(id_client)
```

Création de tables

Client(<u>id_client</u>, prenom_client, nom_client, nationalite_client)

```
CREATE TABLE IF NOT EXISTS Client(
    id_client INT AUTO_INCREMENT,
    prenom_client VARCHAR(50) NOT NULL,
    nom_client VARCHAR(50) NOT NULL,
    nationalite_client VARCHAR(50) NOT NULL,
    PRIMARY KEY(id_client)
) ENGINE=InnoDB;
```

Exercices

Créer les tables suivantes :

- Client(id_client, prenom_client, nom_client, nationalite_client)
- Commande(id_commande,#id_client, date_commande)
- Produit(id_produit, nom_produit)
- Employé(id_employé, nom_employé, prenom_employé)
- DetailsCommande(#id_commande,# id_produit, #id_employé, quantite, prix_piece)
- Paiement(#id_commande, montant)

```
CREATE TABLE IF NOT EXISTS <NOM_REL.>(
    <nom_att1> <type_att1> <option_att1>,
    ...
    <nom_attN> <type_attN> <option_attN>,
    PRIMARY KEY(nom_attI),
    FOREIGN KEY(nom_attJ) REFERENCES <REL.>(<nom_attJ>),
    ...
    FOREIGN KEY(nom_attK) REFERENCES <REL.>(<nom_attK>)
) ENGINE=InnoDB;
```

Commande(<u>id_commande</u>, #id_client, date_commande)

```sql
CREATE TABLE IF NOT EXISTS Commande(
    id_commande INT AUTO_INCREMENT,
    id_client INT NOT NULL,
    date_commande DATE NOT NULL,
    PRIMARY KEY(id_commande),
    FOREIGN KEY(id_client) REFERENCES Client(id_client)
) ENGINE=InnoDB;
```

Produit(id_produit, nom_produit)

```sql
CREATE TABLE IF NOT EXISTS Produit(
    id_produit INT AUTO_INCREMENT,
    nom_produit VARCHAR(50) NOT NULL,
    PRIMARY KEY(id_produit)
) ENGINE=InnoDB;
```

Employé(id_employé, nom_employé, prenom_employé)

```sql
CREATE TABLE IF NOT EXISTS Employe(
    id_employe INT AUTO_INCREMENT,
    nom_employe VARCHAR(50) NOT NULL,
    prenom_employe VARCHAR(50) NOT NULL,
    PRIMARY KEY(id_employe)
) ENGINE=InnoDB;
```

DetailsCommande(#id_commande,#id_produit,#id_employé,quantité, prix_piece)

```sql
CREATE TABLE IF NOT EXISTS DetailsCommande(
    id_commande INT NOT NULL,
    id_produit INT NOT NULL,
    id_employe INT NOT NULL,
    quantite INT NOT NULL,
    prix_piece FLOAT NOT NULL,
    PRIMARY KEY (id_commande,id_produit),
    FOREIGN KEY(id_commande) REFERENCES Commande(id_commande),
    FOREIGN KEY(id_produit) REFERENCES Produit(id_produit),
    FOREIGN KEY(id_employe) REFERENCES Employe(id_employe)
) ENGINE=InnoDB;
```

Paiement(#id_commande , montant)

```sql
CREATE TABLE IF NOT EXISTS Paiement(
    id_commande INT NOT NULL,
    montant FLOAT NOT NULL,
    PRIMARY KEY id_commande,
    FOREIGN KEY(id_commande) REFERENCES Commande(id_commande)
) ENGINE=InnoDB;
```

Exercice 1

2. Requêtes

Format de base d'une requête :

```
SELECT <att_i>,...,<att_k> | *
FROM <nom_table>
```

- Sélectionne tous les tuples de la table
- Les tuples sont composés des attributs (noms de colonnes)
  - Tous si *
- Possible d'utiliser une fonction dans le select (`DISTINCT`, `COUNT`, `SUM`, `AVG`,...)
- Le résultat d'une requête sera une table

Exercice 1

2. Requêtes

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT DISTINCT Nom
FROM ETUDIANT
```

Nom
Dupont
Durant
Leclercq

Exercice 1

2. Requêtes

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT AVG(Age)
FROM ETUDIANT
```

AVG(Age)
16.2

Exercice 1

2. Requêtes

AS :

```
SELECT <att_i> AS nouveau_nom_colonne
FROM <nom_table>
```

- AS permet de renommer une colonne
- Utile pour les jointures et les unions

Exercice 1

2. Requêtes

Exemple :

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT DISTINCT(Nom) AS NOMDIFF
FROM ETUDIANT
```

NOMDIFF
Dupont
Durant
Leclercq

Exercice 1

2. Requêtes

WHERE :

```
SELECT *
FROM <nom_table>
WHERE CSTR(col)
```

WHERE impose des contraintes sur les tuples en fonction des valeurs des attributs

- $>, <, =$, (NOT) IN, (NOT) EXISTS

Exercice 1

2. Requêtes

Table ETUDIANT :

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16
Durant	Francis	17
Durant	Gustave	17
Leclercq	Alain	14

```
SELECT *
FROM ETUDIANT
WHERE Nom="Dupont"
```

Nom	Prénom	Age
Dupont	Georges	17
Dupont	Henri	16

Exercice 1

2. Requêtes

TABLE ETUDIANT :

ID	Nom	Prénom
1	Leclercq	Alain
2	Durant	Georges
3	Vermeulen	Eddy
4	Delbecq	Nadine

TABLE INTERRO :

ID	Cours	Points
1	Math	16
2	Francais	12
3	Geographie	17
4	Francais	14
3	Math	11

```
SELECT Nom,Prenom
FROM ETUDIANT
WHERE ID IN
    (SELECT ID FROM INTERRO WHERE Points>15)
```

Nom	Prénom
Leclercq	Alain
Vermeulen	Eddy

Exercice 1

2. Requêtes

TABLE ETUDIANT :

ID	Nom	Prénom
1	Leclercq	Alain
2	Durant	Georges
3	Vermeulen	Eddy
4	Delbecq	Nadine

TABLE INTERRO :

ID	Cours	Points
1	Math	16
2	Francais	12
3	Geographie	17
4	Francais	14
3	Math	11

```
SELECT Nom,Prenom
FROM ETUDIANT e
WHERE NOT EXISTS
    (SELECT * FROM INTERRO WHERE (Points<15 AND e.ID=ID))
```

Nom	Prénom
Leclercq	Alain

Exercice 1

2. Requêtes

```
SELECT *
FROM <nom_table_1> NATURAL JOIN <nom_table_2>
```

NATURAL JOIN fait le produit cartésien des entrées de deux tables en se limitant aux tuples dont les entrées ont la même valeur pour les attributs de même nom.

Exercice 1

2. Requêtes

TABLE ETUDIANT :

ID	Nom	Prénom
1	Leclercq	Alain
2	Durant	Georges

TABLE INTERROS :

ID	Cours	Points
1	Math	17
1	Francais	10
2	Geographie	14
2	Francais	12

```
SELECT *
FROM ETUDIANT NATURAL JOIN INTERROS
```

ID	Nom	Prénom	Cours	Points
1	Leclercq	Alain	Math	17
1	Leclercq	Alain	Francais	12
2	Durant	Georges	Geographie	14
2	Durant	Georges	Francais	12

Exercice 1

2. Requêtes

```
SELECT *
FROM <NOM_TABLE>
GROUP BY <colonne>
```

GROUP BY regroupe les résultats par un ou plusieurs attributs de même valeur

Requêtes

TABLE ETUDIANT :

Nom	Prénom	Age	Points
Dupont	Georges	17	12
Dupont	Henri	16	13
Durant	Francis	17	15
Durant	Gustave	17	14
Leclercq	Alain	14	12

```
SELECT Nom, SUM(Age), AVG(Points)
FROM ETUDIANT
GROUP BY Nom
```

Nom	SUM(Age)	AVG(Points)
Dupont	33	12.5
Durant	34	14.5
Leclercq	14	12

Exercice 1

2. Requêtes

```
SELECT *
FROM <NOM_TABLE>
GROUP BY <colonne>
HAVING contrainte_groupe
```

HAVING permet de restreindre les groupes en leur imposant des contraintes

Requêtes

TABLE ETUDIANT :

Nom	Prénom	Age	Points
Dupont	Georges	17	12
Dupont	Henri	16	13
Durant	Francis	17	15
Durant	Gustave	17	14
Leclercq	Alain	14	12

```
SELECT Nom, SUM(Age), AVG(Points)
FROM ETUDIANT
GROUP BY Nom
HAVING AVG(Points)<13
```

Nom	SUM(Age)	AVG(Points)
Dupont	33	12.5
Leclercq	14	12

Exercice 1

2. Requêtes

```
TABLE_1 UNION TABLE_2
```

UNION permet de regrouper deux tables en une seule (si les colonnes sont de même nom)

Exercice 1

2. Requêtes

TABLE Etudiant :

Nom	**Prénom**	Grade
Dupont	Henri	PGD
Vermeiren	Nadine	S

TABLE Professeur :

Nom	**Prénom**	**Cours**
Delobelle	Peter	Informatique-1
Deroy	Marc	Analyse numérique

```
(SELECT Nom, Prenom FROM Etudiant)

UNION

(SELECT Nom, Prenom FROM Professeur)
```

Nom	**Prénom**
Dupont	Henri
Vermeiren	Nadine
Delobelle	Peter
Deroy	Marc

Exercices

- Client(id_client, prenom_client, nom_client, nationalite_client)
- Commande(id_commande,id_client, date_commande)
- Produit(id_produit, nom_produit)
- Employé(id_employé, nom_employé, prenom_employé)
- DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)
- Paiement(id_commande, montant)

a) Pour **chaque** client : nom, prénom et nombre de commandes passées
b) Chercher les commandes passées par des belges ou francais
c) Donner le coût total pour chaque commande
d) Pour **chaque** client : nom, prénom, coût **total** de ses commandes
e) Pour **chaque** client : nom, prénom, coût **moyen** de ses commandes

a) Pour **chaque** client : nom, prénom et nombre de commandes passées

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)

1. Les clients qui ont passé au moins une commande

```
SELECT prenom_client, nom_client, cnt
FROM Client NATURAL JOIN
    (SELECT id_client, COUNT(*) AS cnt
     FROM Commande
     GROUP BY id_client)
```

2. Les clients qui n'ont pas passé de commande

```
SELECT prenom_client, nom_client, 0 AS cnt
FROM Client
WHERE id_client NOT IN
    (SELECT DISTINCT id_client
     FROM Commande)
```

a) Pour **chaque** client : nom, prénom et nombre de commandes passées

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)

3. Union

```
SELECT prenom_client, nom_client, cnt
FROM Client NATURAL JOIN
    (SELECT id_client, COUNT(*) AS cnt
     FROM Commande
     GROUP BY id_client)

UNION

SELECT prenom_client, nom_client, 0 AS cnt
FROM Client
WHERE id_client NOT IN
    (SELECT DISTINCT id_client
     FROM Commande)
```

Exercices

- Client(id_client, prenom_client, nom_client, nationalite_client)
- Commande(id_commande,id_client, date_commande)
- Produit(id_produit, nom_produit)
- Employé(id_employé, nom_employé, prenom_employé)
- DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)
- Paiement(id_commande, montant)

a) Pour **chaque** client : nom, prénom et nombre de commandes passées
b) Chercher les commandes passées par des belges ou francais
c) Donner le coût total pour chaque commande
d) Pour **chaque** client : nom, prénom, coût **total** de ses commandes
e) Pour **chaque** client : nom, prénom, coût **moyen** de ses commandes

b) Chercher les commandes passées par des clients belges ou français

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)

```
SELECT id_client
FROM Client
WHERE nationalite_client="Belge" OR
    nationalite_client="Francais"
```

b) Chercher les commandes passées par des clients belges ou français

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)

```
SELECT id_commande
FROM Commande NATURAL JOIN
    (SELECT id_client
     FROM Client
     WHERE nationalite_client="Belge" OR
         nationalite_client="Francais")
```

c) Donner le cout total pour chaque commande

DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT id_commande, SUM(prix_piece*quantite) AS cout_total
FROM DetailsCommande
GROUP BY id_commande
```

d) Pour **chaque** client : nom, prénom, coût **total** de ses commandes

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

1. Cout de chaque commande

```
SELECT id_commande, SUM(prix_piece*quantite) AS cout
FROM DetailsCommande
GROUP BY id_commande
```

2. Cout total de chaque id_client

```
SELECT  id_client, SUM(cout) AS cout_client
FROM Commande NATURAL JOIN
    (SELECT id_commande, SUM(prix_piece*quantite) AS cout
    FROM DetailsCommande
    GROUP BY id_commande)
GROUP BY id_client
```

d) Pour **chaque** client : nom, prénom, coût **total** de ses commandes

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

3. Natural Join avec client pour le prénom

```
SELECT nom_client, prenom_client, cout_client
FROM Client NATURAL JOIN
    (SELECT  id_client, SUM(cout) AS cout_client
     FROM Commande NATURAL JOIN
        (SELECT id_commande, SUM(prix_piece*quantite) AS cout
        FROM DetailsCommande
        GROUP BY id_commande)
     GROUP BY id_client)
```

d) Pour **chaque** client : nom, prénom, coût **total** de ses commandes

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

4. Union avec ceux n'ayant rien commandé

```
SELECT nom_client, prenom_client, cout_client
FROM Client NATURAL JOIN
    (SELECT  id_client, SUM(cout) AS cout_client
     FROM Commande NATURAL JOIN
        (SELECT id_commande, SUM(prix_piece*quantite) AS cout
        FROM DetailsCommande
        GROUP BY id_commande)
     GROUP BY id_client)

UNION

SELECT nom_client, prenom_client, 0 as cout_client
FROM Client
WHERE id_client NOT IN
    (SELECT DISTINCT id_client FROM Commande)
```

e) Pour **chaque** client : nom, prénom, coût **moyen** de ses commandes

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

1. Coût total des commandes de chaque client (requête précédente)

```
SELECT nom_client, prenom_client, cout_client
FROM Client NATURAL JOIN
    (SELECT  id_client, SUM(cout) AS cout_client
     FROM Commande NATURAL JOIN
        (SELECT id_commande, SUM(prix_piece*quantite) AS cout
        FROM DetailsCommande
        GROUP BY id_commande)
     GROUP BY id_client)

UNION

SELECT nom_client, prenom_client, 0 as cout_client
FROM Client
WHERE id_client NOT IN
    (SELECT DISTINCT id_client FROM Commande)
```

e) Pour **chaque** client : nom, prénom, coût **moyen** de ses commandes

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

2. Il suffit de changer le SUM par AVG (moyenne)

```
SELECT nom_client, prenom_client, cout_client
FROM Client NATURAL JOIN
    (SELECT  id_client, AVG(cout) AS cout_client
     FROM Commande NATURAL JOIN
        (SELECT id_commande, SUM(prix_piece*quantite) AS cout
        FROM DetailsCommande
        GROUP BY id_commande)
     GROUP BY id_client)

UNION

SELECT nom_client, prenom_client, 0 as cout_client
FROM Client
WHERE id_client NOT IN
    (SELECT DISTINCT id_client FROM Commande)
```

Exercices

- Client(id_client, prenom_client, nom_client, nationalite_client)
- Commande(id_commande,id_client, date_commande)
- Produit(id_produit, nom_produit)
- Employé(id_employé, nom_employé, prenom_employé)
- DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)
- Paiement(id_commande, montant)

f) Donner les produits qui n'ont pas encore été commandés
g) Donner les produits commandés par tout le monde
h) Donner les produits commandés par au moins 30 clients
i) Pour les clients qui n'ont pas encore paye toutes leurs commandes : nom, prénom, somme totale des ses commandes, somme qu'ils ont paye, et somme qui reste à être payée
j) Pour **chaque** produit : nom du produit, nombre total de pièces commandées

f) Donner les produits qui n'ont pas encore été commandés

Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT id_produit FROM DetailsCommande
```

f) Donner les produits qui n'ont pas encore été commandés

Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT id_produit,nom_produit
FROM Produit
WHERE id_produit NOT IN
    (SELECT id_produit FROM DetailsCommande)
```

g) Donner les produits commandés par tout le monde

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

Solution 1 : Double négation
1. Sélection des clients qui n'ont pas commandé un produit P

```
SELECT *
FROM Client
WHERE id_client NOT IN
    ((SELECT id_client
      FROM Commande)

    NATURAL JOIN

    (SELECT id_commande
     FROM DetailsCommande
     WHERE id_produit=P.id_produit))
```

g) Donner les produits commandés par tout le monde

Client(id_client, prenom_client, nom_client, nationalite_client)
Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

Solution 1 : Double négation
2. Sélection des produits P qui n'ont pas été commandé par aucun client

```
SELECT id_produit,nom_produit
FROM Produit P
WHERE NOT EXISTS
    (SELECT *
     FROM Client
     WHERE id_client NOT IN
        ((SELECT id_client
          FROM Commande)

        NATURAL JOIN

        (SELECT id_commande
         FROM DetailsCommande
         WHERE id_produit=P.id_produit)))
```

g) Donner les produits commandés par tout le monde

Commande(id_commande,id_client, date_commande)
Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

Solution 2 : Regroupement par produit, contrainte sur le nombre de clients différents par groupe.

```
SELECT nom_produit,id_produit
FROM (DetailsCommande NATURAL JOIN  Commande) NATURAL JOIN
    Produit
GROUP BY id_produit
HAVING COUNT(DISTINCT id_client) IN (SELECT COUNT(*) FROM
    Client)
```

h) Donner les produits commandés par au moins 30 clients

Commande(id_commande,id_client, date_commande)
Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

Semblable à la solution 2 de g)...

```
SELECT nom_produit
FROM (Commande NATURAL JOIN DetailsCommande) NATURAL JOIN
    Produit
GROUP BY id_produit
HAVING COUNT(DISTINCT id_client)>=30
```

i) Pour les clients qui n'ont pas encore paye toutes leurs commandes, donner la somme totale des ses commandes, la somme qu'ils ont paye, et la somme qui reste à être payée

Commande(id_commande,id_client, date_commande)
Client(id_client, prenom_client, nom_client, nationalite_client)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)
Paiement(id_commande, montant)

1. Regrouper le prix de chaque commande, et ce qui a déjà été payé par le client

```
SELECT id_commande, montant AS deja_paye, prix_commande
FROM Paiement NATURAL JOIN
    (SELECT id_commande, SUM (prix_piece*quantite) AS
        prix_commande
     FROM DetailsCommande
     GROUP BY id_commande)
WHERE deja_paye < prix_commande
```

i) Pour les clients qui n'ont pas encore paye toutes leurs commandes : nom, prénom, somme totale des ses commandes, somme qu'ils ont paye, et somme qui reste à être payée

2. Jointures pour les clients et prénoms, regroupement par id_client

```
SELECT nom, prenom, SUM(deja_paye), SUM(prix_commande),
    SUM(prix_commande)-SUM(deja_paye)
FROM
    (SELECT id_commande, montant AS deja_paye, prix_commande
     FROM Paiement NATURAL JOIN
        (SELECT id_commande, SUM (prix_piece*quantite) AS
            prix_commande
         FROM DetailsCommande
         GROUP BY id_commande)
     WHERE deja_paye < prix_commande)

    NATURAL JOIN Commande NATURAL JOIN Client
GROUP BY id_client
```

j) Pour **chaque** produit : nom du produit, nombre total de pièces commandées

Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT nom_produit, total
FROM Produit NATURAL JOIN
    (SELECT id_produit, SUM(quantite) as total
     FROM DetailsCommande
     GROUP BY id_produit))

UNION

SELECT nom_produit, 0 as total
FROM Produit
WHERE id_produit NOT IN
    (SELECT DISTINCT id_produit FROM DetailsCommande)
```

- Client(id_client, prenom_client, nom_client, nationalite_client)
- Commande(id_commande,id_client, date_commande)
- Produit(id_produit, nom_produit)
- Employé(id_employé, nom_employé, prenom_employé)
- DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)
- Paiement(id_commande, montant)

k) Donner le produit le + commandé

- Suggestion : `ORDER BY x DESC LIMIT y`

l) Pour chaque produit, donner la valeur totale des pièces commandees

m) Donner le nombre de produits par employé

n) Donner le nombre d'employes travaillant pour chaque client

k) Donner le produit le + commandé

DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)
En triant...

```
SELECT nom_produit,SUM(quantite)
FROM DetailsCommande
GROUP BY id_produit
ORDER BY SUM(quantite) DESC
LIMIT 1
```

Avec MAX :

```
SELECT nom_produit, SUM(quantite)
FROM DetailsCommande
GROUP BY id_produit
HAVING SUM(quantite)=MAX(SUM(quantite))
```

l) Pour chaque produit, donner la valeur totale des pièces commandees

Produit(id_produit, nom_produit)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT nom_produit, valeur_totale
FROM Produit NATURAL JOIN
    (SELECT id_produit, SUM(quant.*prix_piece) AS valeur_totale
     FROM DetailsCommande
     GROUP BY id_produit)

UNION

SELECT nom_produit, 0 AS valeur_totale
FROM Produit
WHERE id_produit NOT IN
    (SELECT id_produit FROM DetailsCommande)
```

m) Donner le nombre de produits par employé

Employé(id_employé, nom_employé, prenom_employé)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT nom_employe,SUM(quantite) as n_par_employe
FROM Employe NATURAL JOIN DetailsCommande
GROUP BY id_employe

UNION

SELECT nom_employe, 0 as n_par_employe
FROM Employe
WHERE id_employe NOT IN
    (SELECT id_employe FROM DetailsCommande)
```

n) Donner le nombre d'employes travaillant pour chaque client

Client(id_client, prenom_client, nom_client, nationalite_client)
Commande(id_commande,id_client, date_commande)
DetailsCommande(id_commande, id_produit, id_employé, quantite, prix_piece)

```
SELECT prenom_client,nom_client,COUNT(DISTINCT id_employe) AS
    ntravailleurs
FROM DetailsCommande NATURAL JOIN Commande NATURAL JOIN Client
GROUP BY id_client

UNION

SELECT prenom_client, nom_client, 0 AS ntravailleurs
FROM Client
WHERE id_client NOT IN
    (SELECT DISTINCT id_client FROM Commande)
```