Année 2016-17
Responsable : Nicolas Lumineau

U.E. BDW1
Bases de Données et programmation WEB

Partie "Requêtes SQL"

CC BY NC ND

Nicolas Lumineau

Fascicule : Interrogation en SQL

Dans ce fascicule, vous trouverez quelques rappels sur la syntaxe SQL. Ces rappels ne sont pas exhaustifs et il est fortement conseillé de consulter les spécifications du langage SQL correspondant au SGBD que vous utiliserez. Pour ce semestre, nous utiliserons MySQL. Après ces rappels, vous trouverez des exercices applicatifs avec leurs corrigés. Pour finir, vous trouverez à la fin de ce document les sujets des TD1, TD2, TD3 et TD4.

Pourquoi ? Comment ?

Vous trouverez ci-dessous les différents mots-clés de la syntaxe SQL pour manipuler les données. Pour illustrer les différents opérateurs nous considèrons les schémas de relations suivants R(A,B,C,#D), S(G,H,I,J), U(D, E, F), V(K,L,M) et W(M) où les clés primaires sont soulignées et la clé étrangère est préfixée par un #.

R

A	B	C	D
1	2	3	4
2	3	1	4
3	4	1	2
4	3	1	2

S

E	F	G	H
1	2	3	4
2	4	1	3
3	4	NULL	2
4	2	1	3

U

D	I	J
2	3	TOTO
3	1	TITI
4	2	PIM

V

K	L	M
1	2	1
1	2	2
2	1	1
2	1	2
2	1	3
3	2	1

W

M
1
2
3

Eléments de base

Comment exprimer une requête en SQL ?
Une requête SQL ne peut pas avoir moins qu'une clause **SELECT** et une clause **FROM**. La clause SELECT permet d'introduire les attributs qui seront retournés par la requête et la clause FROM permet de spécifier les tables qui sont interrogées. Il est important que tous les attributs qui aparaissent dans le SELECT proviennent d'une des tables spécifiées dans le FROM. Une requête se termine par un pouint vigule (;). Dans la clause SELECT, les attributs sont séparés par des virgules. Dans le FROM, les tables sont séparées par des virgules.

Requête:
Donner les valeurs de B dans R.

```
SELECT B
FROM R;
```

T

B
2
3
4
3

Requête:
Donner les couples de valeurs (B,C) dans R.

```
SELECT B,C
FROM R;
```

T

B	C
2	3
3	1
4	1
3	1

Comment retourner entièrement R ?
Pour retourner toute la relation, il est possible de spécifier tous les attributs dans la clause SELECT ou de spécifier le caractère * dans cette même clause SELECT.

Requête:
Donner R.

```
SELECT A, B, C, D
FROM R;
```

T

A	B	C	D
1	2	3	4
2	3	1	4
3	4	1	2
4	3	1	2

Requête:
Donner R.

```
SELECT *
FROM R;
```

T

A	B	C	D
1	2	3	4
2	3	1	4
3	4	1	2
4	3	1	2

Comment supprimer les doublons ?
Il est possible de supprimer les tuples résultats qui sont répétés plusieurs fois en ajoutant le mot-clé **DISCTINCT** dans la clause SELECT.

Requête:
Donner les valeurs possibles de B dans R.

```
SELECT DISTICNT B
FROM R;
```

T

B
2
3
4

Requête:
Donner les couples de valeurs possibles (B,C) dans R.

```
SELECT DISTINCT B,C
FROM R;
```

T

B	C
2	3
3	1
4	1

Alias et renommage

Comment changer le nom d'un attribut dans le résultat d'une requête ?
Pour afficher un nom d'attribut dans le résultat de la requête qui soit différent du nom de l'attribut dans la table, il est nécessaire de définir un alias pour l'attribut. Pour cela, il es possible d'utiliser le mot-clé **AS** dans la clause SELECT de la requête. Ainsi, il suffit d'ajouter AS <nomAffichage> après l'attribut dans le SELECT pour que celui soit renommé en <nomAffichage> dans le résultat. Evidemment la table n'est pas modifiée [1]

Requête:
Afficher A selon le schéma T(A', B', X, Y).

```
SELECT A AS A',␣B␣AS␣B', C AS X, D AS Y
FROM R;
```

T

A'	B'	X	Y
1	2	3	4
2	3	1	4
3	4	1	2
4	3	1	2

Pourquoi changer le nom d'une table dans la définition d'une requête ?
Il est possible de définir des alias dans le FROM de la requête pour pouvoir donner des noms souvent plus courts aux tables utiles à la définition de la requête ou tout simplement quand la définition de la requête nécessite de manipuler plus d'une instance d'une même table. Dans ce dernier cas, le renommage via un alias est obligatoire pour enlever toute ambiguité.

Comment changer le nom d'une table dans la définition d'une requête ?
Pour définir l'alias d'une table, il n'y a pas de mots clés particuliers. Il suffit simplement de mettre l'alias après le nom de la table dans le FROM. Les deux (le nom de la table et l'alias) sont simplement séparé par un espace.
Requête:
Afficher le produit cartésien de R avec lui-même. Le résultat aura le schéma suivant T(A,B,C,D,A',B',C',D'))

1. Pour persister le renommage de l'attribut, il faudrait faire un ALTER TABLE (cf Fascicule 1).

```
SELECT R1.A, R1.B, R1.C, R1.D,
  R2.A AS A', R2.B AS B', R2.C AS C', R2.D AS D'
FROM R R1, R R2;
```

T

A	B	C	D	A'	B'	C'	D'
1	2	3	4	1	2	3	4
1	2	3	4	2	3	1	4
1	2	3	4	3	4	1	2
1	2	3	4	4	3	1	2
2	3	1	4	1	2	3	4
2	3	1	4	2	3	1	4
2	3	1	4	3	4	1	2
2	3	1	4	4	3	1	2
3	4	1	2	1	2	3	4
3	4	1	2	2	3	1	4
3	4	1	2	3	4	1	2
3	4	1	2	4	3	1	2
4	3	1	2	1	2	3	4
4	3	1	2	2	3	1	4
4	3	1	2	3	4	1	2
4	3	1	2	4	3	1	2

Tri des résultats

Comment trier le résultat d'un requête ?
Il est possible de spécifier l'ordre d'affichage du résultat d'une requête. Pour cela, on utilise le mot-clé **ORDER BY <liste(attribut sens)>** en fin de requête. Le paramètre <liste(attribut sens)> correspond à une liste de d'attributs sur lesquels vont s'appliquer l'ordre. Le principe est le suivant : le résultat est trié selon le premier attribut de la liste, en cas de valeurs identiques sur cet attribut, le second attribut de la liste est considéré. En cas d'égalité sur le premier attribut, les tuples sont triés en fonction du deuxième attribut. En cas de valeurs identiques sur les deux premiers attributs, le troisième attribut est considéré et ainsi de suite. Pour chaque attribut, on spécifie le sens du tri, si c'est croissant (ASC) ou décroissant (DESC). Par défaut le tri est croissant.

Requête:
Donner les tuples de R en les triant selon l'ordre croissant sur B, puis par ordre décroissant sur C en cas d'égalité, puis par ordre croissant sur D en cas d'égalité.

```
SELECT *
FROM R
ORDER BY B, C DESC, D;
```

T

A	B	C	D
1	2	3	4
4	3	1	2
2	3	1	4
3	4	1	2

Critères de sélection de tuples

Comment affecter des contraintes pour sélectionner des tuples ?
Pour sélectionner des tuples, il est possible de définir des conraintes dans la clause **WHERE**. Les conditions exprimées dans cette clause vont s'appliquer à chacun des tuples des relations spécifiées dans le FROM et seuls les tuples vérifiants les contraintes (*i.e.* rendant vraies les contraintes) sont retournées.

Requête:
Donner les tuples de R pour lesquels B > 2.

```
SELECT *
FROM R
WHERE B > 2;
```

T

A	B	C	D
2	3	1	4
3	4	1	2
4	3	1	2

Requête:
Donner les valeurs de A pour lesquelles B = 3 et D > 2.

```
SELECT A
FROM R
WHERE B = 3
AND D > 2;
```

T

A
2

Requête:
Donner les tuples de R pour lesquels B > 3 ou C != 1.

```
SELECT *
FROM R
WHERE B > 3
OR C != 1;
```

T

A	B	C	D
1	2	3	4
3	4	1	2

Requête:
Donner les valeurs de A pour lesquelles soit B = 3 et D > 2, soit B < 3 et D = 4.

```
SELECT A
FROM R
WHERE (B = 3 AND D > 2)
OR (B < 3 AND D = 4);
```

T

A
1
2

Comment sélectionner des tuples par intervalle ?
Pour spécifier l'appartenance d'un attribut à un intervalle de valeurs dans la clause WHERE, il est possible d'utiliser **BETWEEN <borneMin> AND <borneMax>**. Le BETWEEN s'appliquer pour des valeurs numériques, des chaines de caractères (dans ce cas l'ordre lexicographique sert de relation d'ordre) et les dates.

Requête:
Donner les tuples de R pour lesquels A est entre 2 et 4.

```
SELECT *
FROM R
WHERE A BETWEEN 2 AND 4;
```

T

A	B	C	D
2	3	1	4
3	4	1	2
4	3	1	2

Comment sélectionner des tuples à partir d'une absence de valeurs ?
Il est possible de spécifier une contrainte dans le WHERE qui porte sur le fait qu'un attribut n'a pas de valeurs. Pour cela, on utilise la condition **IS NULL**

Requête:
Donner les tuples de S pour lesquels G n'a pas de valeur.

```
SELECT *
FROM S
WHERE G IS NULL;
```

T

E	F	G	H
3	4	NULL	2

Requête:
Donner les tuples de S pour lesquels G a une valeur.

```
SELECT *
FROM S
WHERE G IS NOT NULL;
```

T

E	F	G	H
1	2	3	4
2	4	1	3
4	2	1	3

Comment sélectionner des tuples à partir d'un motif ?
Il est possible de spécifier dans le WHERE une contrainte de sélection basée sur un motif et non une valeur exacte. Pour cela, il est possible d'utiliser le mot-clé **LIKE** combiné avec les caractères joker : '_' pour spécifier un caractère quelconque et '%' pour spécifier un nombre quelconque de caractères.

Requête:
Donner les tuples de U pour lesquelles la valeur de J commence par un T.

```
SELECT *
FROM U
WHERE J LIKE 'T%';
```

T

D	I	J
2	3	TOTO
3	1	TITI

Requête:
Donner les tuples de U pour lesquels la deuxième lettre de J est un I.

```
SELECT *
FROM U
WHERE J LIKE '_I%';
```

T

D	I	J
3	1	TITI
4	2	PIM

Requête:
Donner les tuples de U pour lesquelles J est un mot de 3 lettres.

```
SELECT *
FROM U
WHERE J LIKE '___';
```

T

D	I	J
4	2	PIM

Requête:
Donner les tuples de U pour lesquels J contient la lettre M.

```
SELECT *
FROM U
WHERE J LIKE '%M%';
```

T

D	I	J
4	2	PIM

Les jointures

Comment mettre en relation deux tables ?
Il est possible de mettre en relation deux tables par une jointure. La jointure se caractérise par le fait que l'on ne compare plus un attribut avec une constante, mais dans une jointure, on compare un attribut d'une relation avec un autre attribut d'une autre relation. Il exsite deux types de jointure :

- les **jointures internes** : où seuls les tuples vérifiant la condition de jointure sont retournés
- les **jointures externes** : où tous les tuples d'une relations sont retournés et les informations de l'autre relation sont ajoutées si la condition de jointure est vérifiée.

Requête:
Donner les valeurs de A et J tel que l'attribut D de R soit égale à l'attribut D de U.

```
SELECT R.A, U.J
FROM R JOIN S
 ON R.D = U.D;

--ou

SELECT R.A, U.J
FROM R, S
WHERE R.D = U.D;
```

T

A	J
1	PIM
2	PIM
3	TOTO
4	TOTO

Requête:
Donner les valeurs de A et E pour lesquelles B > F.

```
SELECT R.A, S.E
FROM R JOIN S
 ON R.B > S.F;

--ou

SELECT R.A, S.E
FROM R, S
WHERE R.B > S.E;
```

T

A	E
2	1
2	4
3	1
3	4
4	1
4	4

Requête:
Donner toutes les valeurs de A dans R et si la valeur de A correspond à une valeur de K, on précisera alors la valeur de L correspondante.

```
SELECT R.A, V.L
FROM R LEFT JOIN V
 ON R.A = V.K
```

T

A	L
1	2
1	2
2	1
2	1
2	1
3	2
4	NULL

Requête:
Donner les valeurs de tous les attributs de R et de S pour lesquels le couple (A,B) correspond au couple (E,F).

```
SELECT *
FROM R JOIN S
 ON R.A=S.E
  AND R.B=S.F
```

T

A	B	C	D	G	H
1	2	3	4	3	4
3	4	1	2	NULL	2

Les requêtes imbriquées

Comment sélectionner des tuples à partir de valeurs se trouvant dans une autre table ?
Il est possible de spécifier une contrainte dans le WHERE pour dire qu'un ou plusieurs attributs doivent se trouver dans une collection de valeurs retournées par une requête imbriquée. Pour cela, il est possible d'utiliser le mot-clé **IN**.

Requête:
Donner les tuples de R pour lesquels la valeur de B dans R est une valeur de F dans S.

```
SELECT *
FROM R
WHERE B IN
  (SELECT F
   FROM S);
```

T

A	B	C	D
1	2	3	4
3	4	1	2

Requête:
Donner les tuples de R pour lesquels la valeur de B dans R n'est pas une valeur de F dans S.

```
SELECT *
FROM R
WHERE B NOT IN
  (SELECT F
   FROM S);
```

T

A	B	C	D
2	3	1	4
4	3	1	2

Remarque : La seule possibilité pour avoir dans le WHERE un attribut égale au résultat d'une requête (*i.e.* pour avoir un = à la place du IN), est de s'assurer que la requête imbriquée ne retourne qu'un seul tuple.

Requête:
Donner les valeurs de A pour lesquelles le couple (A,B) soit une valeur de (D,I) dans U.

```
SELECT A
FROM R
WHERE (A,B) IN (SELECT D, I FROM U);
```

T

A
2

Comment s'assurer que la comparaison entre un attribut et une collection de valeurs est toujours vraie ?
Dans le cas où une requête imbriquée retourne plusieurs résultats et que l'on souhaite vérifier que la valeur d'un attribut est égale/différente/inférieure/supérieure à toutes les valeurs retournées par la requête imbriquée, il est possible d'utiliser le mot-clé **ALL** pour introduire la requête imbriquée. La syntaxe dans le WHERE de la requête principale est la suivante : attribut op ALL (requêteImbriquée) où op $\in \{=, !=, <, \leq, \geq, >\}$ et où la requête imbriquée ne retourne qu'un attribut.
Requête:
Donner les tuples de R pour lesquels la valeur de B est supérieure ou égale à n'importe quelle valeur de F dans S.

```
SELECT *
FROM R
WHERE B >= ALL (SELECT F FROM S);
```

T

A	B	C	D
3	4	1	2

Comment s'assurer que la comparaison entre un attribut et une collection de valeurs est vérifiée au moins une fois ?
Dans le cas où une requête imbriquée retourne plusieurs résultats et que l'on souhaite vérifier que la valeur d'un attribut est égale/différente/inférieure/supérieure à au moins une des valeurs retournées par la requête imbriquée, il est possible d'utiliser le mot-clé **ANY** pour introduire la requête imbriquée. La syntaxe dans le WHERE de la requête principale est la suivante : attribut op ANY (requêteImbriquée) où op $\in \{=, != , <, \leq, \geq, >\}$ et où la requête imbriquée ne retourne qu'un attribut.
Requête:
Donner les tuples de R pour lesquels la valeur de B est strictement supérieure à au moins une des valeurs de F dans S.

```
SELECT *
FROM R
WHERE B > ANY (SELECT F FROM S);
```

T

A	B	C	D
2	3	1	4
3	4	1	2
4	3	1	2

Comment s'assurer qu'il existe au moins un tuple dans une collexion de valeurs ?
Pour répondre à certaines requêtes, il est nécessaire pour sélectionner certains tuples de s'assurer que d'autres tuples existent dans une autre table. Dans ce cas, il est possible d'utiliser le mot-clé **EXISTS** pour introduire la requête imbriquée. Il est important de noter que contrairement au IN, ANY et ALL, le mot-clé EXISTS n'est pas précédé par un attribut.
Requête:
Donner les tuples de S pour lesquels il existe une valeur de D dans R qui soit égale à la valeur de leur H.

```
SELECT *
FROM S
WHERE EXISTS(SELECT *
        FROM R WHERE S.H = R.D);
```

T

E	F	G	H
1	2	3	4
3	4	NULL	2

Regroupement et critères de sélection de groupes

Pourquoi créer des groupes en SQL ?
Il est possible de créer des groupes de tuples pour leur appliquer généralement des fonctions d'agrégats qui permettront de retourner un résultat par groupe. Les principales fonctions d'agrégats peuvent être :

- **COUNT(Att)** : pour dénombrer les valeurs de l'attribut Att dans le groupe
- **AVG(Att)** : pour calculer la moyenne des valeurs de l'attribut Att dans le groupe
- **SUM(Att)** : pour calculer la somme cumulée des valeurs de l'attribut Att dans le groupe
- **MIN(Att)** : pour calculer le minimum des valeurs de l'attribut Att dans le groupe
- **MAX(Att)** : pour calculer le maximum des valeurs de l'attribut Att dans le groupe

Comment créer des groupes ?
Il est possible de créer des groupes grâce au mot-clé **GROUP BY <listeAttributsCommuns>** où <listeAttributsCommuns> correspond à la liste des atributs qui ont une valeur commune au sein d'un groupe.

Requête:
Pour chaque valeur de D, donner le nombre de valeurs distinctes pour l'attribut C. La valeur calculée sera renommée 'nbC'.

```
SELECT D, COUNT(DISTINCT C) AS nbC
FROM R
GROUP BY D;
```

T

D	nbC
4	2
2	1

Requête:
Pour chaque valeur de C, donner la somme cumulée des valeurs de B. Le résultat sera renommé 'sumB'

```
SELECT C, SUM(B) AS sumB
FROM R
GROUP BY C;
```

T

C	sumB
3	2
1	10

Comment définir des conraintes au niveau des groupes (et non plus au niveau des tuples) ?
Il est possible de définir des contraintes qui vont permettre de sélectionner des groupes grâce au mot-clé **HAVING <contrainteGroupe>** où <contrainteGroupe> correspond une comparaison d'une valeur calculée au sein des groupes avec un constante ou le résultat d'une requête imbriquée.

Requête:
Donner les valeurs de D pour lesquelles il existe au moins deux valeurs de C différentes. On précisera ce nombre de valeurs distinctes de C et on le renommera 'nbC'.

```
SELECT D, COUNT(DISTINCT C) AS nbC
FROM R
GROUP BY D
HAVING nbC > 1;
```

T

D	nbC
4	2

Requête:
Pour chaque valeur de C, donner la somme cumulée des valeurs de B. On ne gardera que les valeurs de C pour lesquelles la valeur calculée est plus grande que la moyenne des valeurs de F.Le résultat sera renommé 'sumB'

```
SELECT C, SUM(B) AS sumB
FROM R
GROUP BY C
HAVING sumB >= (SELECT AVG(F) FROM S);
```

T

C	sumB
1	10

car la moyenne des F est 3.

Opérateur inter-requêtes

Comment faire l'union de deux requêtes ?
Il est possible d'unir le résultat de deux requêtes grâce au mot-clé **UNION**. Il est important que les deux requêtes retournent le même nombre d'attributs.

Requête:
Donner les tuples de R et les tuples de S.

```
(SELECT * FROM R) UNION (SELECT * FROM R)
```

T

A	B	C	D
1	2	3	4
2	3	1	4
3	4	1	2
4	3	1	2
1	2	3	4
2	4	1	3
3	4	NULL	2
4	2	1	3

Exercices applicatifs

Contexte

La base de données de l'enseigne de pâtissiers 'LETEMPSMIS' implanté dans toute la France est définie selon le schéma suivant :

Patisserie (idP, nom, categorie, prixunitaire) est une relation stockant le catalogue de pâtisseries proposées par les patissiers, avec :

- *idP* est un entier qui identifie une patisserie ;
- *nom* désigne le nom de la pâtisserie ;
- *categorie* représente la catégorie de la pâtisserie tel que 'gâteau', 'macaron' ...
- *prixunitaire* représente le prix unitaire de la patisserie.

Exemple : (1, 'Baba au Rhum', 'gâteau', 4.15)

Recette (idP, numD, ingredients, description, auteur, annee) est une relation stockant les recettes des pâtisserie, avec :

- *idP* référence une patisserie,
- *numD* est un entier qui identifie de manière relative la déclinaison de la recette. L'attribut vaut 0 pour indiquer qu'il s'agit de la recette initiale. Une recette dispose d'une déclinaison du moment qu'un ingrédient est changé ou le procédé est modifié.
- *ingredients* désigne une liste d'ingrédients nécessaires à la recette ;
- *description* désigne un texte descriptif du procédé associé à la recette ;
- *auteur* désigne le prénom et le nom (séparé par un espace) de l'auteur(e) de la recette ;
- *annee* désigne l'année où la recette a été proposée la première fois.

Exemple : (1, 0, 'Beurre, eau, farine, lait, levure, oeufs, rhum', 'Mélanger...', 'Nicolas Stohrer', 1835)

Personne (idPe, nom, prenom, anneeNaiss) est une relation stockant les pâtissiers, avec :

- *idPe* est un entier qui identifie une personne ;
- *nom* et *nom* désignent respectivement le noms et le prénom de la personne ;
- *anneeNaiss* désigne l'année de naissance de la personne ;

Exemple : (10, 'Prendunepart', 'Jean', '1979')

Realisation (idPe, idP, numD, dateR, nbRealisation) est une relation permettant de stocker les recettes qui ont été réalisée, avec :

- *idPe* référence une personne ;
- *idP, numD* référence une recette ;
- *dateR* désigne la date à laquelle la personne a réalisé la recette ;
- *nbRealisation* désigne le nombre de pâtisseries qui ont été réalisées.

Exemple : (10, 1, 0, '2014-12-02', 20)

Boutique (idB, nom, adresse, codepostal, ville) est une relation stockant les différentes boutiques distribuant les pâtisseries, avec :

- *idB* est un entier qui identifie une boutique ;
- *nom* désigne le nom de la boutique ;
- *adresse* désigne le numéro et nom de rue de la boutique ;
- *codepostal* désigne le code postal de la boutique ;
- *ville* désigne la ville où se trouve la boutique.

Exemple : (2, 'LETEMPSMIS Royale', '16-18 rue Royale', '75008', 'Paris')

EstRattache (idB, idPe, dateEmbauche) est une relation permettant de relier les pâtissiers aux boutiques dans lesquelles ils travaillent, avec :

— *idB* référence une boutique ;

— *idPe* référence une personne ;

— *dateEmbauche* désigne la date à laquelle le pâtissier a été embauchée dans la boutique ;

Exemple : (2, 10, '2006-09-01')

Questions

Ecrire en SQL, les requêtes suivantes :

1. Donner la liste des catégories de pâtisseries triée selon l'ordre lexicographique (les doublons devront être supprimés).
2. Donner le nom des pâtisseries de catégorie 'gâteau' ayant des marrons parmi ses ingrédients. Le résultat sera trié selon l'ordre lexicographique inverse sur le nom de la pâtisserie.
3. Donner le nom et le prénom des pâtissiers ayant réalisé plus de 100 macarons dans une journée entre le 1 septembre 2015 et le 31 décembre 2015. On précisera également la date la réalisation.
4. Donner les recettes initiales dont on ne connaît pas l'auteur. On précisera le nom de la pâtisserie avec la recette.
5. Donner les boutiques parisiennes qui ont employé le pâtissier 'Marc Arron'. On précisera pour chaque boutique la date d'embauche et le résultat sera trié par rapport à cette date selon l'ordre chronologique inverse.
6. Donner les personnes de moins de 40 ans (cette année) qui sont auteurs d'au moins une recette.
7. Donner les pâtissiers qui n'ont jamais réalisé de 'Baba au Rhum' en 2014.
8. Donner le nombre de recettes différentes pour réaliser un Succès. Le résultat sera renommé en 'nbSuccesDifférents'.
9. Pour chaque nom de pâtisserie, donner le nombre total de réalisations durant l'année 2015. Le résultat sera renommé en 'total2015'.
10. Pour chaque boutique provinciale (i.e. hors Paris), donner le montant en euros des pâtisseries dans le mois de décembre 2015. Le résultat sera renommé en 'prod12-2015-euros'.
11. Donner les boutiques n'ayant jamais proposé de pâtisserie à base de Rhum.
12. Donner le nom et le prix unitaire de la pâtisserie la plus chère (utilisation de MAX ou EXISTS interdite).
13. Donner le nom et le prix unitaire de la pâtisserie la plus chère pour chaque catégorie de pâtisserie.
14. Donner les pâtisseries réalisées par plus de 8 pâtissiers différents en 2015.
15. Donner les pâtissiers qui ont déjà réalisés au moins une recette de chacune des pâtisseries du catalogue. (indice : Pour ces pâtissiers, il n'existe aucune pâtisserie pour laquelle il n'existe aucune réalisation effectuée par le pâtissier)

Réponses

1.
```
SELECT DISTINCT p.categorie
FROM Patisserie p
ORDER BY p.categorie;
```

2.
```
SELECT DISTINCT p.nom
FROM Patisserie p
JOIN Recette r ON p.idP = r.idP
WHERE ingredients LIKE '%marrons%'
ORDER BY p.nom DESC;
```

3.
```
SELECT pa.nom, pa.prenom, r.dateR
FROM (Patissier pa NATURAL JOIN Realisation r)
        JOIN Patisserie p ON r.idP = p.idP
WHERE r.nbRealisation >= 100
AND p.categorie = 'Macaron'
AND r.dateR BETWEEN '2015-09-01' AND '2015-12-31';
```

4.
```
SELECT *
FROM recette
WHERE numD = 0 AND auteur IS NULL;
```

5.
```
SELECT b.*, e.dateEmbauche
FROM (Boutique b NATURAL JOIN EstRattache e)
        JOIN Patissier p ON e.idPe = p.idPe
WHERE p.prenom = 'Marc' AND p.nom = 'Arron'
AND  b.ville LIKE 'Paris'
ORDER BY e.dateEmbauche DESC;
```

6.
```
SELECT DISTINCT r.auteur
FROM Recette r
WHERE r.auteur IN (SELECT concat(p.prenom, ' ', p.nom)
                   FROM Patissier p
                   WHERE YEAR(now()) - anneeNaiss < 40);
```

7.
```
SELECT DISTINCT p.*
FROM Patissier p
WHERE p.idPe NOT IN (SELECT r.idPe
                     FROM Realisation r NATURAL JOIN Patisserie pa
                     WHERE pa.nom = 'Baba au Rhum'
                     AND r.dateR LIKE '2014%');
```

8.
```
SELECT count(r.numD) AS nbSuccesDifferents
FROM Patisserie pa NATURAL JOIN Recette r
WHERE pa.nom = 'Succ\'es';
```

9.
```
SELECT pa.nom, sum(r.nbRealisation) AS total2015
FROM Patisserie pa NATURAL JOIN Realisation r
WHERE YEAR(r.dateR) = '2015'
GROUP BY pa.nom, pa.idP;
```

10.
```
SELECT b.nom, sum(pa.prixunitaire*r.nbRealisation) AS 'prod12-2015-euros'
FROM Patisserie pa NATURAL JOIN Realisation r
        JOIN EstRattache e ON e.idPe = r.idPe
        JOIN Boutique b ON b.idB=e.idB
```

```
WHERE b.ville != 'Paris' AND r.dateR LIKE '2015-12-__'
AND r.dateR > e.dateEmbauche
GROUP BY b.nom, b.idB;
```

11.
```
SELECT b.*
FROM Boutique b
WHERE NOT EXISTS(SELECT *
                 FROM EstRattache e
                 JOIN Realisation r ON e.idPe = r.idPe
                 JOIN Recette re ON r.idP = re.idP AND  r.numD = re.numD
                 WHERE e.idB = b.idB AND re.ingredients LIKE '%rhum%');
```

12.
```
SELECT pa.nom, pa.prixunitaire
FROM Patisserie pa
WHERE pa.prixunitaire >= ALL( SELECT prixunitaire FROM patisserie );
```

13.
```
SELECT pa.categorie, pa.nom, pa.prixunitaire
FROM Patisserie pa
WHERE pa.prixunitaire = (
SELECT max( p.prixunitaire )
FROM Patisserie p
WHERE p.categorie = pa.categorie ) ;
```

14.
```
SELECT pa.*
FROM Patisserie pa NATURAL JOIN Realisation r
WHERE YEAR(r.dateR) = 2015
GROUP BY pa.idP
HAVING count(DISTINCT r.idPe) > 8;
```

15.
```
SELECT DISTINCT p.*
FROM Patissier p
WHERE NOT EXISTS( SELECT pa.idP
                  FROM Patisserie pa
                  WHERE NOT EXISTS( SELECT r.*
                          FROM Realisation r
                          WHERE pa.idP = r.idP AND r.idPe = p.idPe));
```