TD4: BDD1.

Une plateforme propose plusieurs jeux payants. Le prix d'une partie est exprimé par un nombre entier, « nombre de crédits ». Par un moyen de paiement qui n'a pas d'importance ici un joueur peut recharger son nombre de crédits. Sa « cagnotte » désigne le nombre de crédits dont il dispose. Chaque partie jouée rapporte un certain nombre de points au joueur (ne pas confondre points et crédits). La base de données est organisée comme suit

1. une table des utilisateurs, `users` :
   - `pseudo` : pseudo (chaîne)
   - `nom` : nom complet (chaîne)
   - `inscrit` : date d'inscription (date)
   - `cagnotte` : crédits disponibles (entier)
2. une table `jeux` recense tous les jeux disponibles sur la plateforme :
   - `id` : identifiant du jeu (chaîne)
   - `nom` : nom du jeu (chaîne)
   - `prix` : prix d'une partie (entier)
3. une table des `parties` où figurent toutes les parties jouées :
   - `joueur` : pseudo d'un utilisateur
   - `jeu` : identifiant d'un jeu
   - `quand` : date de la partie
   - `points` : points obtenus (entier)

Quelques fonctions SQL :

— extraction d'une partie d'une date : `date_part` (*part*, *date*) où *part* est une chaîne telle que `'month'`, `'day'`, `'year'`

— moyenne : `avg()` (fonction de regroupement)

Exercice 1 :

Q 1 .

Indiquez les commandes SQL permettant de recueillir les résultats suivants. Entre parenthèses, sont précisées, si nécessaire, les informations que l'on souhaite voir apparaître dans la réponse.

NB : Retour sur la fonction **count** (et autres fonctions de regroupement) :
l'expression `count(distinct nom_d_attribut)` compte le nombre de valeurs distinctes pour cet attribut.
Cette forme s'applique également aux autres fonctions de regroupement (mais c'est moins utile).

1. la liste des jeux d'un prix > 100 ayant été joués plus de 1000 fois. (id du jeu, nombre de parties)
2. la liste des jeux d'un prix > 100 ayant été joués par plus de 50 joueurs différents. (id du jeu, nombre de joueurs)
3. le nombre de points totalisés par l'ensemble des joueurs (points)
4. le nombre de points totalisés par l'ensemble des joueurs, par année (année, points)
5. le nombre de points totalisés par l'ensemble des joueurs, par mois (année, mois, points)
6. le nombre de points totalisés par joueur et par année (année, pseudo, points)
7. la liste des joueurs n'ayant jamais joué (pseudo)
8. la liste des joueurs ayant joué à des jeux distincts (pseudo)
9. la liste des jeux ayant été joués moins de 10 fois. (id du jeu, nombre de parties)

1. select jeu, count(jeu) as nb_parties from parties
   join jeux on jeux.id = parties.jeu where prix > 100
   group by jeu having count(jeu) > 1000;
2. select jeu, count(joueurs) from parties.
   join jeux on jeux.id = parties.jeu where prix > 100
   group by jeu having count(joueurs) > 50;

3. select sum(points) from parties;

4. select sum(points), date_part(quand, "year") from parties
group by date_part(quand, "year");

5. select sum(points), date_part(quand, "month") from parties
group by date_part(quand, "month");

6. select sum(points), date_part(quand, "month"), pseudo
from parties
group by date_part(quand, "month"), joueurs;

7. select pseudo from joueurs left join
parties on p.joueurs = j.pseudo
where jeu is null;

8. select joueur from parties group by joueurs having count(distinct jeu)
> 1;

9. select id as nb_parties from parties Right join jeux
group by id having count(*) < 10.

Q 2.

— Proposez une requête affichant la liste des joueurs classés par ordre décroissant du nombre de points qu'ils ont gagné.

— La clause `LIMIT` n (où n est un entier) permet de limiter à n (maxi) le nombre de tuples renvoyés par la requête. L'utilisation de LIMIT n'a réellement de sens que pour des requêtes où l'ordre des tuples est fixé par ORDER BY (si non cela reviendrait à renvoyer n tuples plus ou moins aléatoirement)

Utilisez la clause LIMIT pour ne renvoyer que le « top 3 » (3 joueurs ayant gagné le plus de points) dans l'hypothèse où il n'y a pas d'ex-aequo.

— Cette requête convient-elle en cas d'ex-aequo ? (illustrez votre réponse par un ou plusieurs exemples)

— La notion même de « top 3 » a-t-elle toujours un sens dans ce cas ?

NB : Une bonne réponse pour le cas général fait appel à la fonction `rank() over(...)`, et donc aux opérations de fenêtrage (ou partition), hors du programme du cours.

select joueur, sum(points) group by joueur order by sum(points) desc;

~ limit 3.

Mon ça ne convient pas :

nom	points.
j1	1
j2	1
...	1