# IFT 2935 — TRAVAIL FINAL — HIVER 2020
# BASES DE DONNÉES

- *À rendre au plus tard par Studium le mercredi 22 avril 23 h 55*
- *Ceci n'est pas un travail d'équipe.*
- *Toute documentation permise : livres, youtube, doc SQL sur internet.*
- *Vous devez par contre travailler seul ; nous pouvons exiger une entrevue Skype, Zoom ou autre pour vérifier votre niveau de compréhension ou d'expertise.*
- *Les questions peuvent être de difficultés fort inégales.*

Sur le réseau du DIRO, pour utiliser les commandes `\df` et `\sf` qui listent des fonctions faut être sur arche (version 11.7 de psql).

Les question suivantes réfèrent au schéma `releves_notes` utilisé dans le cours sur les vues et fonctions. Sur Arche, en ligne de commande, vous exécutez `psql -h postgres < /u/boyer/ift2935/releves_notes.sql` pour charger ce schéma dans votre base pour pouvoir ensuite y ajouter les fonctionnalités demandées.

**1.** (20 points) Compléter le fichier `exam_no1.sql` qui se trouve dans le répertoire `/u/boyer/ift2935` en définissant la vue `notesinfo` pour qu'elle retourne une table donnant les notes avec le titre dans le format

```
   sigle    | trimc | matricule  |  prog   | nature | note |             titre
------------+-------+------------+---------+--------+------+--------------------------
 IFT1015    | 20074 | 53357      | 117510  | OBL    | B-   | Programmation 1
```

ainsi qu'un déclencheur qui aura pour effet de changer la note d'un étudiant et de garder un journal (log) des modifications. La requête

```
    update notesinfo set note='A'
     where  sigle='IFT3911' and matricule='53357' and trimc='20114';
```

(en fin de fichier) doit avoir pour effet de changer la note dans la table des notes et de rajouter une entrée dans le format

```
 trimc |   sigle    | matricule  |  prog   | nature | note |           ts
-------+------------+------------+---------+--------+------+---------------------------
 20114 | IFT3911    | 53357      | 117510  | OPT    | C    | 2020-03-29 14:25:35.65578
```

dans le journal (table des logs) où `ts` est le jour et l'heure de la modif et la note du journal est la note précédant la modification.

**2.** (25 points) Complétez le fichier `exam_no2.sql` du répertoire `/u/boyer/ift2935` pour que la fonction `releve` prenne en paramètres le matricule d'un étudiant et un numéro de programme et soit telle que

```
set search_path to releves_notes;
select * from releve('51780','117510');
```

retourne une table dans le format suivant :

```
 trimc |   sigle    |                  titre                  | cred | note
-------+------------+-----------------------------------------+------+-------
 20054 | IFT1025    | Programmation 2                         |    3 | B
 20054 | IFT1065    | Struct. discrètes en info               |    3 | C
 20054 | Moy        | Moyenne du trimestre                    |    6 | 2.500
 20054 | MoyCum     | Moyenne cumulative                      |    6 | 2.500
 ...
 20084 | IFT2505    | Optimisation linéaire                   |    3 | C+
 20084 | IFT3545    | Graphes et réseaux                      |    3 | B
 20084 | Moy        | Moyenne du trimestre                    |    6 | 2.650
 20084 | MoyCum     | Moyenne cumulative                      |   57 | 2.404
```

avec pour chaque cours de cet étudiant dans le programme demandé [1] le nombre de crédits s'il contribue à la moyenne, pour chaque trimestre le nombre de crédits contributoires du trimestre et la moyenne du trimestre ainsi que le nombre de crédits contribuant à la moyenne cumulative et cette moyenne cumulative. Votre sortie doit correspondre à ce qui se trouve dans le fichier `/u/boyer/ift2935/testq2.txt`.

Attention, il y a des cours de 3 et de 4 crédits. Pour chaque trimestre vous aurez besoin de la somme des crédits contributoires et de la somme des produits de ces valeurs par la valeur associée à la note littérale. La moyenne est toujours la somme de ces produits divisée par la somme des crédits, que ce soit par trimestre ou cumulatif. Cet exercice exige des jointures externes, des group by, des over et des union en plus de l'utilisation de fonctions et doit être fait en SQL (pas de PL/pgSQL).

Vous devez absolument rendre un fichier qui s'exécute correctement avec la commande `psql -h postgres -a < exam_no2.sql`. Si vous n'arrivez pas à faire le travail demandé vous pouvez écrire plusieurs fonctions et les appeler pour fournir les tables en sortie. Une fonction qui retourne la table désirée mais sans aucune moyenne vaudra 5 points. Une fonction qui retourne uniquement la table donnant pour chaque trimestre la moyenne du trimestre vaudra 5 points additionnels. Une fonction qui retourne uniquement la table donnant pour chaque trimestre la moyenne cumulative vaudra 10 points additionnels. Ceci donne au total 20 points si vous retournez les trois tables. Une fonction unique qui fait le tout de façon correcte avec une table unique en sortie donne 25 points.

1. Attention, un étudiant peut suivre des cours IFT dans plus d'un programme

**3.** (25 points) Complétez le fichier `exam_no3.sql` du répertoire `/u/boyer/ift2935` pour que la requête

```
select * from amort(10000.00,1000.00,0.05);
```

retourne un tableau d'amortissement pour un emprunt dont le montant est donné par le premier argument (ici `10000$`), le paiement periodique par de deuxième (ici `1000$`) et le taux périodique par le troisième (ici `5%`). A la première période il n'y a aucun paiement. A la deuxième période le montant dû est calculé en ajoutant les intérêts à la valeur due de la période précédente puis en soustrayant la somme payée. Quand le montant dû est inférieur au déboursé périodique, on paie cette somme et c'est terminé. Il s'agit ici d'écrire une fonction qui fait appel à un with récursif. La sortie attendue pour l'exemple particulier ci-dessus est

```
 period |   pay   |   rest
--------+---------+----------
      1 |     0.0 | 10000.00
      2 | 1000.00 |  9500.00
 ...
     15 | 1000.00 |   200.68
     16 |  210.72 |        0
```

**4.** (20 points)

(i) (7 points) Soit l'ordonnancement suivant (aucune transaction n'est encore confirmée) où les trois transactions $T_1$, $T_2$ et $T_3$ s'exécutent avec des actions entrelacées dans le temps (où ne sont représentées que les lectures R et les écritures W et où le temps s'écoule de gauche à droite).

| | | | | |
|---|---|---|---|---|
| $T_1$ : | R(X),R(Y),W(X), | | W(X), | |
| $T_2$ : | | R(Y), | | R(Y) |
| $T_3$ : | | | W(Y), | |

Tracez-en le graphe de précédence et en déduire si l'ordonnancement est sérialisable par permutation (justifiez bien ce qui dans le graphe vous permet de tirer la conclusion).

(ii) (6 points) Donnez un exemple d'ordonnancement sérialisable en vue mais pas par permutation et pour lequel (contrairement à l'exemple de cours) le résultat n'est pas équivalent à l'exécution de l'une seule des transactions. Justifiez bien pourquoi il est sérialisable en vue non sérialisable par permutation.

(iii) (7 points) Dans le traitement des quatre transactions suivantes on indique les lectures et écritures et verrous accordées par le SBGD ; le temps s'écoule de gauche à droite ; S désigne une demande (accordée) de verrou partagé (Shared) pour lecture (R) et X une demande (accordée) de verrou exclusif pour écriture (W). Exécutez l'algorithme pour déterminer s'il y a verrou mortel (deadlock) et tirez la conclusion (rem : on ne voit que la partie courante de l'exécution ; s'il y avait eu des confirmations (commit) qui auraient alors libéré des verrous, elles seraient indiquées).

| | | | | | |
|---|---|---|---|---|---|
| $T_1$ : | S(A),R(A), | | S(B),R(B) | | |
| $T_2$ : | | X(B),W(B), | | X(C),W(C) | |
| $T_3$ : | | | S(C),R(C), | | S(B),R(B) |
| $T_4$ : | | | | X(A),W(A), | |

**5.** (10 points) Dans les notes de cours, on présente trois scénarios où le log est soit (a), soit (b), soit (c) au moment de la panne.

| (a) | (b) | (c) |
|---|---|---|
| <$T_0$ start> | <$T_0$ start> | <$T_0$ start> |
| <$T_0$, $A$, 1000, 950> | <$T_0$, $A$, 1000, 950> | <$T_0$, $A$, 1000, 950> |
| <$T_0$, $B$, 2000, 2050> | <$T_0$, $B$, 2000, 2050> | <$T_0$, $B$, 2000, 2050> |
| | <$T_0$ commit> | <$T_0$ commit> |
| | <$T_1$ start> | <$T_1$ start> |
| | <$T_1$, $C$, 700, 600> | <$T_1$, $C$, 700, 600> |
| | | <$T_1$ commit> |

Donnez le détail du log si la reprise réussit (sans panne durant la reprise).