TROUVER MON MASTER (version light)

Groupe 6

LAZRAK Ilona - 22001576

CHARREAU Léo - 21913600

CHEVALIER Maël - 22003139

BARRERE Loreena - 22206698

Notre base de données est le produit de notre imagination. Toute ressemblance avec des personnes ou événements réels serait purement fortuite.

SOMMAIRE

DESCRIPTION ET EXPLICATION DE LA BASE DE DONNÉES

Description :
L'Éducation Nationale propose un modèle simplifié de la candidature d'un master.

Chaque personne est définie par un numéro, et possède un nom, prénom, la ville où il habite, téléphone, et une date de naissance. Une personne peut-être soit étudiant, soit enseignant. Un étudiant est caractérisé par son numéro étudiant, et possède une filière, et sa moyenne générale en licence. Un enseignant est défini par son numéro (le NUMEN), et sa spécialité. Il peut également être responsable de plusieurs masters, mais chacun à des années différentes.

Un étudiant peut candidater à un master, à une date donnée, et reçoit un résultat, accepté ou refusé. S'il n'a pas reçu de résultats dans les 1 mois : il est automatiquement accepté. Un Master est décrit par son numéro, son domaine d'étude et son parcours. Chaque master est rattaché à une et une seule université, qui est décrite par son nom, sa ville, et le numéro de téléphone du secrétariat.

Enfin, chaque master est également composé de plusieurs UE, caractérisées par leur code possédant un titre. Une UE peut être utilisée dans plusieurs masters, en spécifiant dans chaque cas le volume horaire et les crédits attribués.

Pour simplifier, l'adresse de l'université et d'une personne sera uniquement composée de la ville.

Explication :
Les étudiants qui sortent de licences ont, pour certains, besoin de faire des candidatures pour un ou plusieurs masters. Il est donc pertinent de créer une base de données permettant de rassembler les étudiants ainsi que les masters dans lesquels ils candidatent. Il est intéressant de préciser qui est le responsable du master afin que les étudiants puissent adresser leur lettre de motivation à ce responsable. La localisation de l'université est également importante pour que les étudiants sachent où se situera leur prochaine école. Le contenu de la formation permet d'avoir de l'information supplémentaire sur un master.

DICTIONNAIRE DES DONNÉES

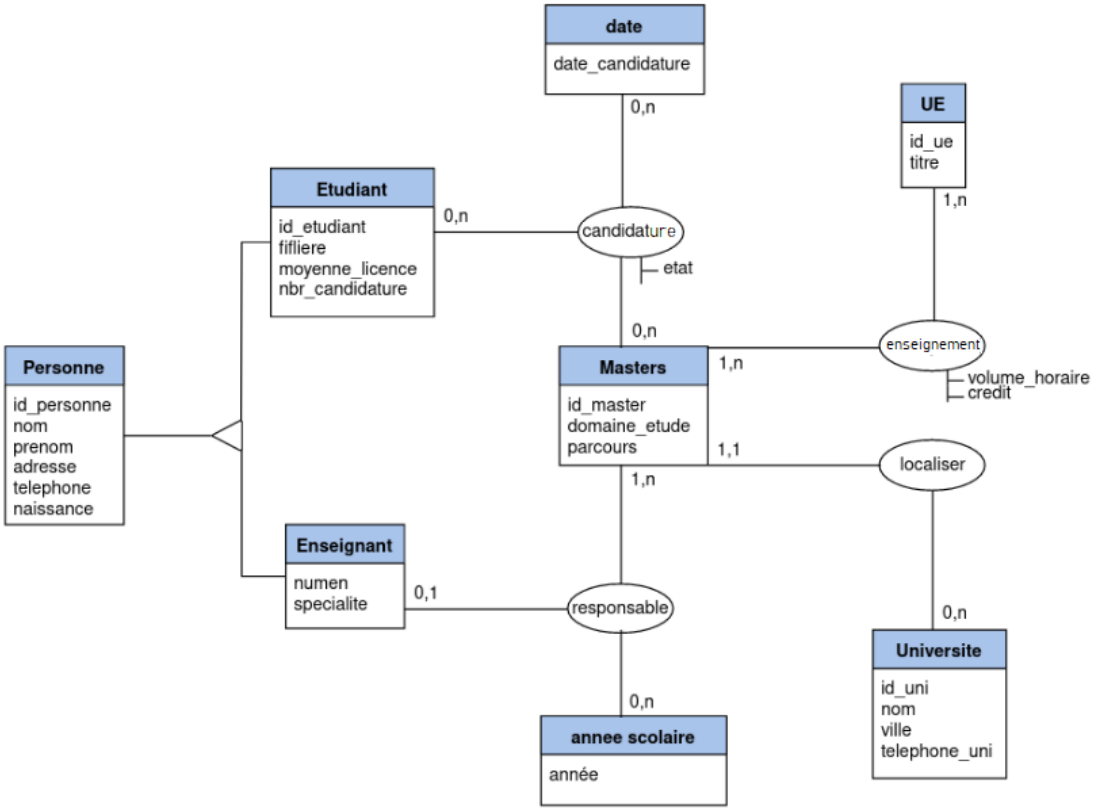
NOM	DESCRIPTION	TYPE	LONGUEUR	NATURE *	CONTRAINTE
ID_PERSONNE	Identifiant de la personne	char	10	Unique	identifiant
PERSONNE.NOM	Nom de famille de la personne	varchar	20	Obligatoire	not null
PRENOM	Prénom d'usage de la	varchar	25	Obligatoire	not null

	personne				
ADRESSE	Ville où habite la personne	varchar	50	Obligatoire	not null
TELEPHONE	Numéro de téléphone de la personne	numeric	10	Obligatoire	not null
NAISSANCE	Date de naissance de la personne	date		Obligatoire	
ID_ETUDIANT	Identifiant de l'étudiant	char	10	Unique	identifiant
FILIERE	Nom de la filière de l'étudiant	varchar	30	Obligatoire	not null
MOYENNE_LICENCE	Moyenne de l'étudiant sur ses 3 années	numeric	4	Obligatoire	entre 0,00 et 20,00
NBR_CANDIDATURE	nombre de candidature de l'étudiant	numeric	3	facultatif	calculer a l'aide d'un trigger
NUMEN	Identifiant du responsable	char	10	Unique	identifiant
SPECIALITE	Spécialité du responsable	varchar	30	Obligatoire	not null
ID_UE	Identifiant de l'unité d'enseignement	char	10	Unique	identifiant
TITRE	Intitulé de L'UE	varchar	30	obligatoire	not null
ID_UNI	Identifiant de l'université	char	10	Unique	identifiant
UNIVERSITE.NOM	Nom de l'université	varchar	30	Obligatoire	not null
VILLE	Ville ou se situe l'université	varchar	30	Obligatoire	not null
TELEPHONE_UNI	Numéro de téléphone de l'administration de l'université	numeric	10	Obligatoire	not null
ID_MASTER	Numéro d'identification d'un master	char	10	Unique	identifiant
DOMAINE_ETUDE	Domaine (ou sujet) du master	varchar	30	Obligatoire	not null
PARCOURS	Parcours (ou mention) du master	varchar	100	Obligatoire	not null

DATE_CANDIDATURE	Date de la candidature	date		Obligatoire	not null
ETAT	État du traitement de la candidature	varchar	10	obligatoire	entre “ACCEPTE”,” REFUSE” ou “TRAITEMENT”
ANNEE	Année à laquelle l’enseignant était responsable du master	numeric	4	Unique	identifiant
VOLUME_HORAIRE	Nombre d’heures semestrielles de l’UE dans le MASTER	numeric	3	Obligatoire	not null
CREDITS	Nombre de crédits accordés par l’UE dans le MASTER	numeric	2	Obligatoire	entre 0 et 30

*parmis obligatoire, facultatif, unique

SCHÉMA ENTITÉ ASSOCIATION



Les explications du schéma se trouvent dans la partie DESCRIPTION ET EXPLICATION DE LA BASE DE DONNÉES (page 3).

SCHÉMA RELATIONNEL

Notre schéma entité/association possède la relation "année scolaire" composée uniquement d'une date. Pour la suite de notre projet, il n'est pas utile de créer cette table. Elle n'apparaîtra plus comme clé étrangère, mais uniquement comme composante de la clé primaire de la relation “Responsable” afin de simplifier notre base de donnée. Le même phénomène se produit avec la table “date” pour la date de candidature d’un étudiant à un master.

Personne(num_personne, nom, prenom, adresse, telephone, naissance)
Etudiants(*#num_étudiant*, filière, moyenne_licence)
Enseignant(*#numen*, spécialité)
Responsable(*#numen, #num_master,* année)
Master(num_master, domaine_étude, parcours, *#num_uni*)
Candidature(*#num_étudiant, #num_master*, date_candidature, état)
Université(num_uni, nom, ville, téléphone)
Enseignements(*#num_master, #num_ue*, volume_horaire, crédits)
UE(num_ue, titre)

SCHÉMA PHYSIQUE

```
ALTER session SET NLS_DATE_FORMAT='DD-MM-YYYY';

CREATE TABLE PERSONNE (
  ID_PERSONNE CHAR(10),
  NOM VARCHAR(20) CONSTRAINT NN_PERSONNE_NOM NOT NULL,
  PRENOM VARCHAR(25) CONSTRAINT NN_PERSONNE_PRENOM NOT NULL,
  ADRESSE VARCHAR(50) CONSTRAINT NN_PERSONNE_ADRESSE NOT NULL,
  TELEPHONE VARCHAR(10) CONSTRAINT NN_PERSONNE_TELEPHONE NOT NULL,
  NAISSANCE DATE,
  CONSTRAINT PK_PERSONNE PRIMARY KEY (ID_PERSONNE)
);

CREATE TABLE ETUDIANT (
  ID_ETUDIANT CHAR(10),
  FILIERE VARCHAR(30) CONSTRAINT NN_ETUDIANT_FILIERE NOT NULL,
  MOYENNE_LICENCE NUMERIC(4,2) CONSTRAINT CK_ETUDIANT_MOYENNE CHECK
(MOYENNE_LICENCE BETWEEN 0 AND 20),
  NBR_CANDIDATURE NUMERIC(3,0),
  CONSTRAINT PK_ETUDIANT PRIMARY KEY (ID_ETUDIANT),
  CONSTRAINT FK_ETUDIANT_NUM_PERSONNE FOREIGN KEY (ID_ETUDIANT)
      REFERENCES PERSONNE(ID_PERSONNE)
      ON DELETE CASCADE
);


CREATE TABLE ENSEIGNANT (
  NUMEN CHAR(10),
  SPECIALITE VARCHAR(30) CONSTRAINT NN_ENSEIGNANT_SPECIALITE NOT NULL,
  CONSTRAINT PK_ENSEIGNANT PRIMARY KEY (NUMEN),
  CONSTRAINT FK_ENSEIGNANT FOREIGN KEY (NUMEN)
      REFERENCES PERSONNE(ID_PERSONNE)
      ON DELETE CASCADE
);

CREATE TABLE UE (
  ID_UE CHAR(10),
  TITRE VARCHAR(150) CONSTRAINT NN_UE_TITRE NOT NULL,
  CONSTRAINT PK_UE PRIMARY KEY (ID_UE)
);
```

```sql
CREATE TABLE UNIVERSITE (
  ID_UNI CHAR(10),
  NOM VARCHAR(30) CONSTRAINT NN_UNIVERSITE_NOM NOT NULL,
  VILLE VARCHAR(30) CONSTRAINT NN_UNIVERSITE_VILLE NOT NULL,
  TELEPHONE_UNI VARCHAR(10) CONSTRAINT NN_UNIVERSITE_TEL NOT NULL,
  CONSTRAINT PK_UNIVERSITE PRIMARY KEY (ID_UNI)
);

CREATE TABLE MASTERS (
  ID_MASTER CHAR(10),
  DOMAINE_ETUDE VARCHAR(30) CONSTRAINT NN_MASTER_DOMAINE NOT NULL,
  PARCOURS VARCHAR(100) CONSTRAINT NN_MASTER_PARCOURS NOT NULL,
  NUM_UNI CHAR(10),
  CONSTRAINT PK_MASTER PRIMARY KEY (ID_MASTER),
  CONSTRAINT FK_MASTER_NUM_UNI FOREIGN KEY (NUM_UNI)
      REFERENCES UNIVERSITE(ID_UNI)
      ON DELETE CASCADE
);

CREATE TABLE CANDIDATURE (
  NUM_ETUDIANT CHAR(10),
  NUM_MASTER CHAR(10),
  DATE_CANDIDATURE DATE CONSTRAINT NN_CANDIDATURE_DATE NOT NULL,
  ETAT VARCHAR(11) CONSTRAINT CK_CANDIDATURE_ETAT CHECK (ETAT IN
('ACCEPTE', 'REFUSE' , 'TRAITEMENT' )),
  CONSTRAINT PK_CANDIDATURE PRIMARY KEY (NUM_ETUDIANT, NUM_MASTER,
DATE_CANDIDATURE),
  CONSTRAINT FK_CANDIDATURE_NUM_ETUDIANT FOREIGN KEY (NUM_ETUDIANT)
      REFERENCES ETUDIANT(ID_ETUDIANT)
      ON DELETE CASCADE,
  CONSTRAINT FK_CANDIDATURE_NUM_MASTER FOREIGN KEY (NUM_MASTER)
      REFERENCES MASTERS(ID_MASTER)
      ON DELETE CASCADE
);

CREATE TABLE RESPONSABLE (
  NUM_ENSEIGNANT CHAR(10),
  NUM_MASTER CHAR(10),
  ANNEE NUMERIC(4) CONSTRAINT NN_RESPONSABLE_DATE NOT NULL,
  CONSTRAINT PK_RESPONSABLE PRIMARY KEY (NUM_ENSEIGNANT, NUM_MASTER,
ANNEE),
  CONSTRAINT FK_RESPONSABLE_NUMEN FOREIGN KEY (NUM_ENSEIGNANT)
      REFERENCES ENSEIGNANT(NUMEN)
```

```
      ON DELETE CASCADE,
  CONSTRAINT FK_RESPONSABLE_NUM_MASTER FOREIGN KEY (NUM_MASTER)
      REFERENCES MASTERS(ID_MASTER)
      ON DELETE CASCADE
);

CREATE TABLE ENSEIGNEMENT (
  NUM_MASTER CHAR(10),
  NUM_UE CHAR(10),
  VOLUME_HORAIRE NUMERIC(3,0) CONSTRAINT NN_VOLUME_HORAIRE NOT NULL,
  CREDITS NUMERIC(2,0) CONSTRAINT CK_ENSEIGNEMENT_CREDITS CHECK
(CREDITS BETWEEN 0 AND 30),
  CONSTRAINT PK_ENSEIGNEMENT PRIMARY KEY (NUM_MASTER, NUM_UE),
  CONSTRAINT FK_ENSEIGNEMENT_NUM_MASTER FOREIGN KEY (NUM_MASTER)
      REFERENCES MASTERS(ID_MASTER)
      ON DELETE CASCADE,
  CONSTRAINT FK_ENSEIGNEMENT_NUM_UE FOREIGN KEY (NUM_UE)
      REFERENCES UE(ID_UE)
      ON DELETE CASCADE
);
```

REQUÊTES SQL

Requête 1 :

Quels sont les étudiants qui se sont inscrits à tous les masters ?

Comme c'est une requête avec une division, on peut voir la question sous une autre forme pour nous aider à écrire la requête : quels sont les étudiants tels qu'il n'existe aucun masters pour lequel ils n'ont pas candidaté.

```
SELECT ID_PERSONNE, PRENOM, NOM
FROM PERSONNE P
WHERE NOT EXISTS (
     SELECT * FROM MASTERS M WHERE NOT EXISTS (
          SELECT * FROM CANDIDATURE WHERE P.ID_PERSONNE =
          CANDIDATURE.NUM_ETUDIANT AND M.ID_MASTER =
          CANDIDATURE.NUM_MASTER));
```

Explication :

On sélectionne le numéro, le prénom et le nom de la personne sur la table "Personne". On pourrait aussi récupérer l'ID_PERSONNE via la table ETUDIANT, mais cela nous forcerait à faire un jointure avec la table PERSONNE pour récupérer le nom et le prénom, qui se trouve seulement dans cette table. Ensuite, on dit qu'il n'existe pas de master pour lequel cet

étudiant n'est pas candidat. Ensuite on assemble toutes les tables ensemble afin de préciser qu'on parle bien du même étudiant et des mêmes masters.

Résultat attendu :

ID_PERSONNE	PRENOM	NOM
0000000001	Loreena	BARRERE

Requête 2 :

Quels sont les étudiants qui ont candidaté uniquement à tous les masters d'informatique ?

```
SELECT NUM_ETUDIANT, PRENOM, NOM
FROM CANDIDATURE C
JOIN PERSONNE ON NUM_ETUDIANT = ID_PERSONNE
WHERE NOT EXISTS (
SELECT * FROM MASTERS M WHERE DOMAINE_ETUDE = 'Informatique' and
NOT EXISTS (
   SELECT * FROM CANDIDATURE WHERE C.NUM_ETUDIANT =
CANDIDATURE.NUM_ETUDIANT
   AND M.ID_MASTER = CANDIDATURE.NUM_MASTER))
Group By NUM_ETUDIANT, PRENOM, NOM
having count(NUM_MASTER) = (select count(*) from Masters where
DOMAINE_ETUDE = 'Informatique');
```

Explication :

Cette requête est inspirée de la requête précédente, nous allons juste expliquer ici les choses rajoutées. Un petit peu comme pour la requête précédente, nous avons cherché quels sont les étudiants tels qu'il n'existe aucun masters d'informatique pour lequel ils n'ont pas candidaté. Ensuite, un GROUP BY est utilisé pour pouvoir compter le nombre de master d'informatique dans lequel un étudiant à candidater, puis nous comparons ce résultats avec le nombre total de master informatique, en comptant les lignes qui ont pour domaine d'étude 'Informatique' dans la table MASTERS

Résultat attendu :

NUM_ETUDIANT	PRENOM	NOM
0000000002	Leo	CHARREAU

Requête 3

Pour chaque étudiant, dans quels master a-t-il été accepté (état = ACCEPTE)? Donné par ordre alphabétique des noms de famille.

```
SELECT NUM_ETUDIANT, NOM, PRENOM, NUM_MASTER
FROM CANDIDATURE
JOIN PERSONNE ON NUM_ETUDIANT = ID_PERSONNE
WHERE ETAT = 'ACCEPTE'
GROUP BY NUM_ETUDIANT, NOM, PRENOM, NUM_MASTER
ORDER BY NOM ASC;
```

Explication :
Parmi les informations sélectionnées, certaines se trouvent dans la table candidature, d'autres dans la table personne. Il nous faut donc faire une jointure entre ces tables. Dans le GROUP BY, on retrouve tous les attributs sélectionnés (dans le SELECT). Pour ordonner les résultats par ordre alphabétique des noms de familles, on utilise ORDER BY sur l'attribut NOM en précisant ASC. Par défaut, les résultats sont données par ordre croissant (on rappelle : ASC pour un ordre croissant, DESC pour un ordre décroissant)

Résultat attendu :

NUM_ETUDIA	NOM	PRENOM	NUM_MASTER
0000000001	BARRERE	Loreena	MAS0000013
0000000001	BARRERE	Loreena	MAS0000009
0000000001	BARRERE	Loreena	MAS0000005
0000000001	BARRERE	Loreena	MAS0000003
0000000001	BARRERE	Loreena	MAS0000001
0000000001	BARRERE	Loreena	MAS0000004
0000000001	BARRERE	Loreena	MAS0000007
0000000001	BARRERE	Loreena	MAS0000010
0000000009	HUNTER	Erin	MAS0000007
0000000009	HUNTER	Erin	MAS0000009

Requête 4 :

Quelles sont les UE qui intéressent l'étudiant Erin HUNTER ?

```
SELECT distinct TITRE
FROM UE
JOIN ENSEIGNEMENT ON NUM_UE = ID_UE
JOIN CANDIDATURE ON CANDIDATURE.NUM_MASTER =
ENSEIGNEMENT.NUM_MASTER WHERE NUM_ETUDIANT = (SELECT ID_PERSONNE
FROM PERSONNE
WHERE NOM = 'HUNTER' AND PRENOM = 'Erin');
```

Explication :
Cette requête n'est pas compliquée dans l'écriture mais sert essentiellement à montrer la facilité de navigation de notre base de données. Concrètement, on part d'un nom d'étudiant et on remonte à l'aide des JOIN le long des candidatures au Masters, puis depuis les Masters on retrouve les UE qui y sont enseignées.

Résultat attendu:

TITRE
Imagerie cellulaire et resonance magnetique

Metabolisme integre et regulations
Genetique moleculaire des maladies hereditaires
Stage academique
Physique des nanostructures
Synthese des biomolecules
Cancers et aspects moleculaires
Stage industriel
Acquisition et traitements des donnees 1
Pre Professionnalisation en informatique
Modelisation et Simulation en Physique

TITRE

Physique et technologie des composants
Physique experimentale

Requête 5 :
Quels sont les étudiants qui ont une moyenne supérieure à la moyenne de tous les étudiants ?

```
SELECT NOM, PRENOM
FROM ETUDIANT
JOIN PERSONNE ON ID_ETUDIANT = ID_PERSONNE
WHERE MOYENNE_LICENCE > (SELECT AVG(MOYENNE_LICENCE) FROM
ETUDIANT);
```

résultat attendu :

NOM	PRENOM
BARRERE	Loreena
CHARREAU	Leo
BARRERE	Vanessa
HUNTER	Erin

EXPLICATIONS PROCÉDURES, FONCTIONS, TRIGGERS

Trigger 1 :
L'âge d'une personne. Avant chaque modification de la table, on regarde si la personne est bien née, et si elle a moins de 100 ans. Si ce n'est pas le cas, on lève une exception et on indique à l'utilisateur l'erreur qu'il a commise. On a créé ce trigger afin de garder une base de données crédible, évitant d'avoir des étudiants de 130 ans par exemple.

```
CREATE OR REPLACE TRIGGER check_age_personne
  BEFORE INSERT OR UPDATE ON PERSONNE
```

```
  FOR EACH ROW
DECLARE
  ERREUR_AGE_MAX EXCEPTION;
  ERREUR_AGE_MIN EXCEPTION;

BEGIN

  IF add_months(:new.NAISSANCE, 0) > add_months(SYSDATE, 0)
      THEN RAISE ERREUR_AGE_MIN;

  ELSIF add_months(:new.NAISSANCE, 0) < add_months(SYSDATE,
-12*100)
      THEN RAISE ERREUR_AGE_MAX;
  END IF;

  EXCEPTION
      WHEN ERREUR_AGE_MAX THEN RAISE_APPLICATION_ERROR(-20254,'L
age est trop grand');
      WHEN ERREUR_AGE_MIN THEN RAISE_APPLICATION_ERROR(-20255,'L
age est trop petit');

END;
/
```

Trigger 2 :

Remplit l'attribut "NBR_CANDIDATURE" de la table ETUDIANT après insert ou update sur CANDIDATURE. À chaque insert, on va récupérer le numéro d'étudiant à la candidature, puis on va compter son nombre d'apparition dans la table, on sauvegarde ce nombre avant de venir l'injecter dans la table ETUDIANT, en vérifiant bien que le numéro d'étudiant que l'on modifie est celui de l'étudiant qui vient de s'inscrire. L'intérêt de ce trigger est essentiellement pour du suivi, savoir où en est un étudiant dans ses candidatures, on pourrait éventuellement placer une limite de candidature maximum par exemple.

```
CREATE OR REPLACE TRIGGER maj_nbr_candidature
  AFTER UPDATE OR INSERT ON CANDIDATURE
  DECLARE
      etu ETUDIANT%rowtype;
      nb_cand ETUDIANT.NBR_CANDIDATURE%TYPE;
  BEGIN
      FOR etu IN (SELECT ID_ETUDIANT FROM ETUDIANT)
      LOOP
          SELECT count(*) into nb_cand FROM CANDIDATURE WHERE
NUM_ETUDIANT=etu.ID_ETUDIANT;
          UPDATE ETUDIANT
              SET NBR_CANDIDATURE=nb_cand
              WHERE ID_ETUDIANT=etu.ID_ETUDIANT;
      end loop;
  END;
```

```
/
```

Trigger 3 :

Si la candidature est dépassée d'un mois, la personne est automatiquement accepté. À chaque insert sur la table candidature, on va passer en revue chaque candidature, à chaque fois qu'on rencontre une candidature toujours en traitement et dont le dépot s'est fait il y a un mois ou plus, alors la candidature passe automatiquement en acceptée. Ce trigger a été créé afin de correspondre au plus au fonctionnement réel d'une inscription dans un master.

```
CREATE OR REPLACE TRIGGER CHECK_CANDIDATURE_PERIME
   FOR INSERT
   ON CANDIDATURE
   COMPOUND TRIGGER

   AFTER STATEMENT IS
   BEGIN
       UPDATE CANDIDATURE
       SET ETAT = 'ACCEPTE'
       WHERE DATE_CANDIDATURE < SYSDATE - INTERVAL '1' MONTH AND
ETAT = 'TRAITEMENT';
   END AFTER STATEMENT;
END;
/
```

Fonction 1 :

Prends en paramètre l'ID d'un master et renvoie le nombre de candidature reçues.

```
CREATE OR REPLACE FUNCTION nbr_total_cand_par_master( ID_MASTER IN
MASTERS.ID_MASTER%TYPE )
RETURN NUMBER is total NUMBER:=0;
  BEGIN
      SELECT count(*) into total FROM CANDIDATURE WHERE
NUM_MASTER=ID_MASTER;
      RETURN total;
  end;
/
```

Fonction 2 :

prends en paramètre l'ID d'un master et renvoie le nombre de candidats que le master a accepté.

```
CREATE OR REPLACE FUNCTION nbr_cand_accepte_par_master( ID_MASTER
IN MASTERS.ID_MASTER%TYPE )
RETURN NUMBER is total NUMBER:=0;
  BEGIN
      SELECT count(*) into total FROM CANDIDATURE WHERE
NUM_MASTER=ID_MASTER AND ETAT='ACCEPTE';
```

```
      RETURN total;
  end;
/
```

Fonction 3 :

Prends en paramètre deux nombres a et b - avec a<=b, et renvoie le pourcentage de a par rapport à b.

```
CREATE OR REPLACE FUNCTION pourcentage(a IN NUMBER, b IN NUMBER)
RETURN NUMBER is pourcentage NUMBER;
  BEGIN
      pourcentage:=round((a/b)*100);
      RETURN pourcentage;
  end;
/
```

Procédure 1 :

On donne le nom d'une université, ça nous renvoie des statistiques. Pour chaque université, on va cycler tous les masters qui y sont enseignés, et on va afficher pour chaque master, le nombre de candidature qu'il a reçu, le nombre qu'il en a accepté, puis un pourcentage d'accès au master (candidatures acceptées/candidatures reçues)*100.
Une fois toutes les stats des masters affichées, on va afficher le total de candidature sur l'université, le total d'acceptées et le pourcentage d'accès à l'université.

```
CREATE OR REPLACE PROCEDURE statistique_uni (nom_uni IN
UNIVERSITE.NOM%TYPE) IS
  id_univ UNIVERSITE.ID_UNI%TYPE;
  master MASTERS%rowtype;
  nbr_cand_total NUMBER;
  nbr_cand_accepte NUMBER;
  total_cand NUMBER:=0;
  total_cand_accepte NUMBER:=0;
BEGIN
  select ID_UNI into id_univ FROM UNIVERSITE WHERE NOM=nom_uni;
  dbms_output.put_line('Pour '  || nom_uni );
  FOR master IN (SELECT * FROM MASTERS WHERE NUM_UNI=id_univ)
      LOOP

nbr_cand_total:=nbr_total_cand_par_master(master.ID_MASTER);
          total_cand:=total_cand+nbr_cand_total;

nbr_cand_accepte:=nbr_cand_accepte_par_master(master.ID_MASTER);
          total_cand_accepte:=total_cand_accepte+nbr_cand_accepte;

dbms_output.put_line('Le master de/d '  || master.DOMAINE_ETUDE ||
' : ' || master.PARCOURS || ' a reçu '
```

```
                                    || nbr_cand_total || '
candidature/s. Il en a accepté ' || nbr_cand_accepte ||
                                    ' soit ' ||
pourcentage(nbr_cand_accepte,nbr_cand_total) || '%');
      end loop;


  dbms_output.put_line('De manière générale, cette université a
donc reçu '  || total_cand ||
                        ' candidatures, en a accepté ' ||
total_cand_accepte || ' soit ' ||
pourcentage(total_cand_accepte,total_cand) || '%');
END;
/
```

TESTS

Tests du trigger sur l'âge d'une personne, d'abord quand l'âge est trop grand, puis quand il est trop petit :

```
INSERT INTO PERSONNE VALUES
('9999999999','TEST_ERREUR','trigger','MTP','9999999999','01-01-1800');
INSERT INTO PERSONNE VALUES
('9999999999','TEST_ERREUR','trigger','MTP','9999999999','01-01-2200');
```

Test du trigger sur le nombre de candidatures d'un étudiant. Prenons l'étudiant d'ID 0000000006 qui a effectué 3 candidatures :

```
SELECT * FROM ETUDIANT WHERE ID_ETUDIANT='0000000006';
```

Après insertion d'une candidature pour cet étudiant dans la table candidature, son nombre de candidatures a augmenté de 1 :

```
INSERT INTO CANDIDATURE VALUES
('0000000006','MAS0000003','01-12-2022','ACCEPTE');
SELECT * FROM CANDIDATURE WHERE NUM_ETUDIANT='0000000006';
SELECT * FROM ETUDIANT WHERE ID_ETUDIANT='0000000006';
```

Test de la fonction qui prend en paramètre l'ID d'un master et renvoie le nombre de candidature reçues par celui-ci. Prenons le master d'ID MAS0000001 :

```
DECLARE
   res1 NUMBER;
BEGIN
   res1:=nbr_total_cand_par_master('MAS0000001');
   DBMS_OUTPUT.put_line('Donc le master MAS0000001 a recu en tout '||
res1 || ' candidatures.');
END;
/
```

Test de la fonction qui renvoie le nombre de candidatures acceptées cette fois-ci. Toujours avec le même master ça donne :

```
DECLARE
   res2 NUMBER;
BEGIN
   res2:=nbr_cand_accepte_par_master('MAS0000001');
   DBMS_OUTPUT.put_line('le master MAS0000001 a accepte en tout '||
res2 || ' candidature.');
END;
/
```

Test du trigger pour la fin de validité d'une candidature. On insère une candidature et toutes les candidatures sont mises à jour

```
INSERT INTO CANDIDATURE VALUES('0000000009', 'MAS0000010',
'04-06-2022', 'TRAITEMENT');
SELECT * FROM CANDIDATURE WHERE NUM_ETUDIANT='0000000009' AND
NUM_MASTER='MAS0000010';
```

Test de la fonction pourcentage basique. Ici on va regarder le pourcentage de candidatures acceptées, toujours avec le même master :

```
DECLARE
   res3 NUMBER;
   res1 NUMBER;
   res2 NUMBER;
BEGIN
   res2:=nbr_cand_accepte_par_master('MAS0000001');
   res1:=nbr_total_cand_par_master('MAS0000001');
   res3:=pourcentage(res2,res1);
   DBMS_OUTPUT.put_line('Donc le master MAS0000001 a accepte '|| res3
|| '% de candidatures.');
END;
/
```

Test de la procédure. On va chercher les statistiques de l'UM fictive de notre BDD:

```
EXEC statistique_uni('UM')
```