❹ UNIDEV™ (suite)

Aller sur Moodle et copier le fichier « Unidev.sql » qui vous permettra de générer les tables de la base de données et d'insérer quelques tuples dans ces tables.

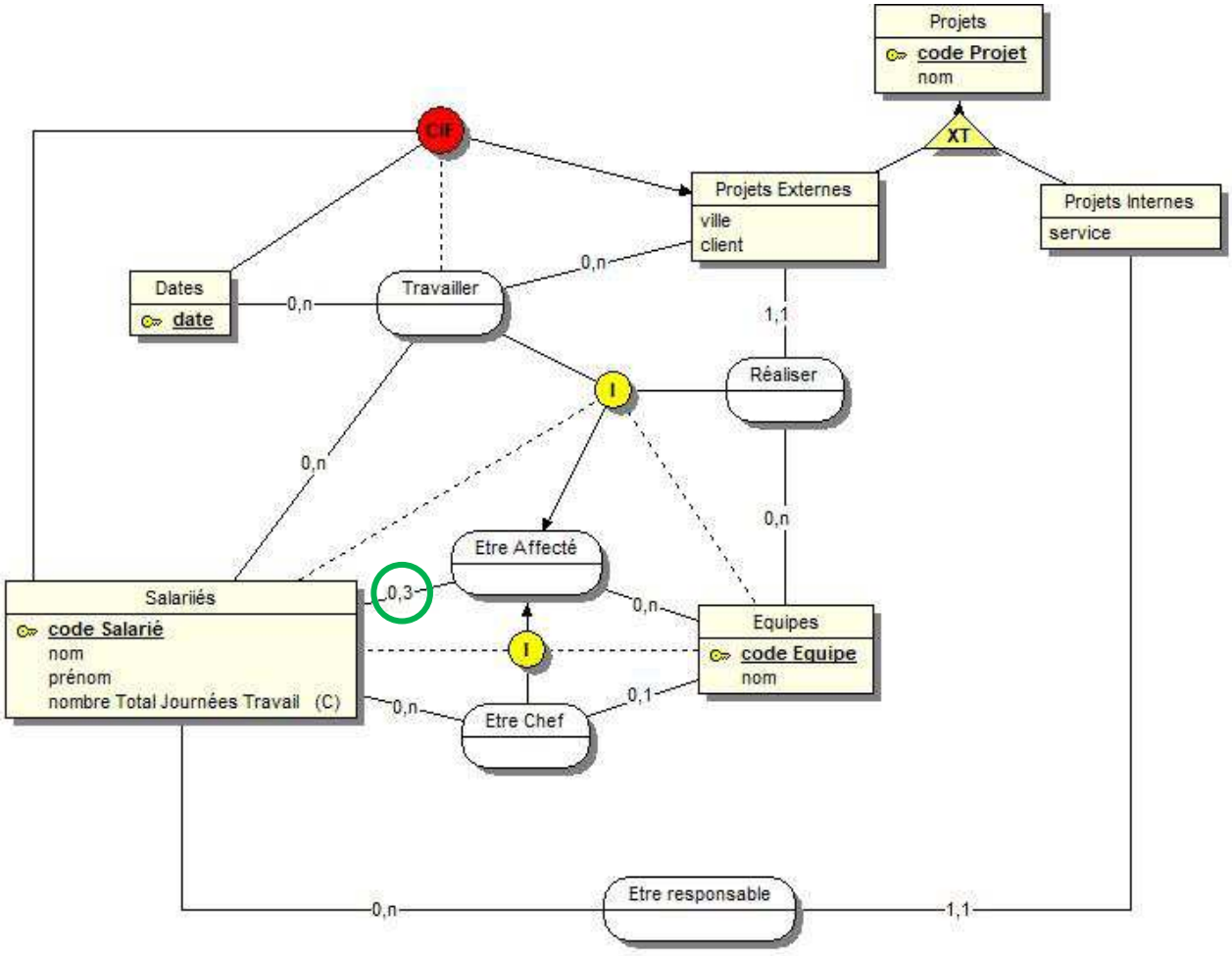
Vous allez ainsi obtenir une base de données correspondant au schéma relationnel suivant :

SALARIES (codeSalarie, nomSalarie, prenomSalarie, *nbTotalJourneesTravail*)
EQUIPES (codeEquipe, nomEquipe, codeSalarieChef#)
PROJETSEXTERNES (codeProjet, nomProjet, villeProjet, clientProjet, codeEquipe#)
PROJETSINTERNES (codeProjet, nomProjet, serviceProjet, codeSalarieResponsable#)
ETREAFFECTE (codeSalarie#, codeEquipe#)
TRAVAILLER (codeSalarie#, dateTravail, codeProjet#)

Dans la relation *Salaries,* l'attribut `nbTotalJourneesTravail` (qui est indiqué en italique) est calculable à partir d'une requête SQL. Il engendre donc de la redondance dans la base de données, mais il a tout de même été maintenu car il permet d'accélérer certaines requêtes qui sont fréquemment exécutées.

Attention, cette base de données ne respecte pas les contraintes A, B, C, D et E vues à la question précédente.
Dans le TP qui va suivre, vous allez donc implémenter ces contraintes soit dans le Langage de Définition de Données SQL quand cela est possible, soit en PL/SQL dans les cas contraires (avec des procédures stockées ou bien des triggers).
Pour rappel, vous trouverez ci-dessous le modèle entités-associations avec toutes les contraintes à prendre en compte.

Il faudra aussi s'assurer que l'attribut calculé *nbTotalJourneesTravail* de la table `Salaries` soit toujours à jour, même lorsque on modifie les données dans la table `Travailler`.



Deuxième partie – Gestion des contraintes du schéma relationnel avec le LDD SQL :

5) Contraintes d'unicité ou CIF (Contrainte d'Intégrité Fonctionnelle) sur une ternaire.
   - On souhaite modifier la base de données afin qu'un salarié ne puisse pas travailler sur plusieurs projets à une même date ; et ce sans écrire de code PL/SQL. On vous rappelle que l'instruction SQL `ALTER TABLE` permet de modifier une table. Et qu'avec cette instruction il est possible de supprimer une contrainte (`ALTER TABLE maTable DROP CONSTRAINT ...`) ou bien de rajouter une contrainte (`ALTER TABLE maTable ADD CONSTRAINT ...`).
   - Résultat attendu :

```
SELECT dateTravail, codeProjet
FROM Travailler
WHERE codeSalarie = 'S6';

DATETRAVAIL  CODEPROJET
------------ -----------
02/01/22     P4
03/01/22     P4
08/01/22     P5
```

   L'instruction suivante doit déclencher **une erreur** (le salarié S6 travaille déjà sur un projet le 8 janvier)

```
INSERT INTO Travailler (codeSalarie, dateTravail, codeProjet)
                VALUES ('S6', '08/01/2022', 'P4') ;
```

6) Contraintes d'inclusion entre deux associations.
   - On souhaite programmer la contrainte qui empêche un salarié d'être chef d'une équipe où il n'est pas affecté, notamment lorsqu'on modifie le chef d'une équipe qui existe déjà (mais il faudrait que cette contrainte soit aussi respectée quand on crée une nouvelle équipe ou lorsqu'on modifie ou supprime des affectations existantes). Ici aussi, on ne veut pas écrire de code PL/SQL.
   - Résultat attendu :

```
SELECT codeEquipe
FROM EtreAffecte
WHERE codeSalarie = 'S3';

CODEEQUIPE
-----------
E3
E4
```

   Si on indique que le salarié S3 est chef de l'équipe E2 il doit y avoir **une erreur** (car le salarié S3 n'est pas affecté à l'équipe E2 ; il ne peut donc pas en être le chef).

```
UPDATE Equipes
SET codeSalarieChef = 'S3'
WHERE codeEquipe = 'E2';
```

   Par contre, si on indique que le salarié S3 est chef de l'équipe E4 il ne doit pas y avoir d'erreur car il est affecté à cette équipe.

```
UPDATE Equipes
SET codeSalarieChef = 'S3'
WHERE codeEquipe = 'E4';
```

Troisième partie – Gestion des contraintes avec des procédures stockées PL/SQL :

7) Mise à jour automatique d'un attribut calculé.
   - On souhaite que l'attribut `nbTotalJourneesTravail` de la table `Salaries` soit mis à jour automatiquement lorsque on insère une nouvelle ligne dans la table `Travailler` (pour simplifier, on ne gèrera pas encore les cas où on modifie ou supprime des lignes de cette table).

     Pour cela, au lieu de faire directement des insertions dans la table `Travailler`, on utilisera une procédure `AjouterJourneeTravail` que l'on vous demande d'écrire et qui doit avoir la signature suivante :

```
PROCEDURE AjouterJourneeTravail (
                        p_codeSalarie Travailler.codeSalarie%TYPE,
                          p_codeProjet Travailler.codeProjet%TYPE,
                        p_dateTravail Travailler.dateTravail%TYPE)
```

- Résultat attendu :

```
SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S2';

NBTOTALJOURNEESTRAVAIL
----------------------
7
```

L'ajout d'une journée de travail pour le salarié S2 doit incrémenter de 1 son nombre total de journées travaillées.

```
CALL AjouterJourneeTravail('S2','P3', '10/01/2022');

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S2';

NBTOTALJOURNEESTRAVAIL
----------------------
8
```

8) Gestion des multiplicités maximales des associations.

- On souhaite maintenant programmer la contrainte qui empêche un salarié d'être affecté à plus de trois équipes lorsqu'on insère une nouvelle ligne dans la table `EtreAffecte` (pour simplifier on ne gèrera pas encore les cas où on modifie des lignes de cette table).

  Pour cela, au lieu de faire directement des insertions dans la table `EtreAffecte`, on utilisera une procédure `AffecterSalarieEquipe` que l'on vous demande d'écrire et dont la signature doit être la suivante :

```
PROCEDURE AffecterSalarieEquipe (
                    p_codeSalarie EtreAffecte.codeSalarie%TYPE,
                      p_codeEquipe EtreAffecte.codeEquipe%TYPE)
```

  Si le salarié dont le code est passé en paramètre n'a pas déjà été affecté à trois équipes, cette procédure l'affecte à l'équipe qui est passée en paramètre. Par contre, s'il est déjà affecté à trois équipes, cette procédure lève une exception avec l'instruction suivante :

```
RAISE_APPLICATION_ERROR(-20001, 'Le salarié est déjà affecté à
                                          au moins 3 équipes');
```

- Résultat attendu :

```
SELECT codeSalarie, codeEquipe
FROM EtreAffecte
WHERE codeSalarie IN ('S1', 'S8');

CODESALARIE  CODEEQUIPE
------------ -----------
S1           E1
S1           E2
S8           E2
S8           E3
S8           E4
```

L'instruction suivante doit déclencher **une erreur** car le salarié S8 est déjà affecté à 3 équipes.

```
CALL AffecterSalarieEquipe('S8', 'E1');
Le salarié est déjà affecté à au moins 3 équipes

SELECT *
FROM EtreAffecte
WHERE codeSalarie = 'S8'
AND codeEquipe = 'E1';

aucune ligne sélectionnée
```

Par contre, l'instruction suivante ne déclenchera pas d'erreur et va rajouter une ligne dans la table `EtreAffecte`.

```
CALL AffecterSalarieEquipe('S1', 'E3');

SELECT *
FROM EtreAffecte
WHERE codeSalarie = 'S1'
AND codeEquipe = 'E3';

CODESALARIE  CODEEQUIPE
------------ -----------
S1           E3
```

9) Gestion des contraintes sur un héritage.

- On souhaite maintenant programmer la contrainte de partition (XT) sur l'héritage entre `Projets`, `ProjetsExternes` et `ProjetsInternes`. La décomposition par distinction a permis d'empêcher d'avoir des projets qui ne sont ni internes, ni externes. Mais il faut encore empêcher d'avoir des projets qui sont à la fois internes et externes lorsqu'on insère une nouvelle ligne dans la table `ProjetsExternes` ou dans la table `ProjetsInternes`.

  Pour cela, au lieu de faire directement des insertions dans ces tables on utilisera les procédures `AjouterProjetExterne` et `AjouterProjetInterne` que l'on vous demande d'écrire :

- Résultat attendu :

```
SELECT codeProjet, 'EXTERNE' AS typeProjet
FROM ProjetsExternes
UNION
SELECT codeProjet, 'INTERNE' AS typeProjet
FROM ProjetsInternes;

CODEPROJET  TYPEPROJET
----------- -----------
P1          EXTERNE
P2          EXTERNE
P3          EXTERNE
P4          EXTERNE
P5          EXTERNE
P6          INTERNE
P7          INTERNE
```

L'utilisation de la procédure `AjouterProjetExterne` pour ajouter un projet externe avec le code P7 doit déclencher **une erreur** car le projet P7 existe déjà dans la table `ProjetsInternes`. Ensuite si vous lancez la requête suivante, vous devriez obtenir ce résultat.

```
SELECT codeProjet, 'EXTERNE' AS typeProjet
FROM ProjetsExternes;

CODEPROJET  TYPEPROJET
----------- -----------
P1          EXTERNE
P2          EXTERNE
P3          EXTERNE
P4          EXTERNE
P5          EXTERNE
```

De même, l'utilisation de la procédure `AjouterProjetInterne` pour ajouter un projet interne P1 doit déclencher **une erreur** car le projet P1 existe déjà dans la table `ProjetsExternes`. Ensuite si vous lancez la requête suivante, vous devriez obtenir ce résultat.

```
SELECT codeProjet, 'INTERNE' AS typeProjet
FROM ProjetsInternes;

CODEPROJET  TYPEPROJET
----------- -----------
P6          INTERNE
P7          INTERNE
```

Par contre, l'utilisation de la procédure `AjouterProjetInterne` pour ajouter un projet interne P8 ayant pour nom 'Programme de comptabilité', comme service 'Service COMPTA' et comme responsable le salarié 'S9' ne déclenchera pas d'erreur et rajoutera une ligne dans la table `ProjetsInternes`. Ensuite si vous lancez la requête suivante, vous devriez obtenir ce résultat.

```
SELECT codeProjet, 'INTERNE' AS typeProjet
FROM ProjetsInternes;

CODEPROJET  TYPEPROJET
----------- -----------
P6          INTERNE
P7          INTERNE
P8          INTERNE
```

10) Contraintes d'inclusion concernant plusieurs associations.

- On souhaite programmer la contrainte qui empêche un salarié de travailler sur un projet qui est réalisé par une équipe dans laquelle n'est pas affecté le salarié en question. On veut que cette contrainte soit vérifiée à chaque fois que l'on ajoute une journée de travail d'un salarié (pour le moment, on ne souhaite pas gérer le cas où on modifie une journée de travail déjà existante).
- Résultat attendu :

```
SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S2';

NBTOTALJOURNEESTRAVAIL
----------------------
8
```

Si on indique que le salarié S2 a travaillé le 11/01/2022 sur le projet P3, il ne doit pas y avoir d'erreur. Ensuite, si on exécute la requête suivante, on obtiendra :

```
SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S2';

NBTOTALJOURNEESTRAVAIL
----------------------
9
```

Puis, si on indique que le salarié S2 a travaillé le 12/01/2022 sur le projet P5, il doit y avoir **une erreur** (car le salarié S2 n'est pas affecté à l'équipe qui travaille sur le projet P5). Ensuite, si on exécute la requête suivante, on obtiendra :

```
SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S2';

NBTOTALJOURNEESTRAVAIL
----------------------
9
```

Quatrième partie – Gestion des contraintes avec des triggers :

11) Mise à jour automatique d'un attribut calculé avec un trigger lors d'une insertion.

- On souhaite programmer un trigger afin que l'attribut `nbTotalJourneesTravail` de la table `Salaries` soit mis à jour automatiquement lorsque on insère une nouvelle ligne directement dans la table `Travailler` (pour commencer, on ne gèrera pas les modifications et les suppressions de lignes de cette table).
- Résultat attendu :

```
SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';

NBTOTALJOURNEESTRAVAIL
----------------------
6
```

L'ajout d'une journée de travail pour le salarié S1 devrait incrémenter de 1 son nombre total de journées travaillées.

```
INSERT INTO Travailler VALUES ('S1', '10/01/2022', 'P1');

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';

NBTOTALJOURNEESTRAVAIL
----------------------
7
```

12) Gestion des multiplicités maximales des associations avec un trigger.

- On souhaite programmer un trigger qui empêche un salarié d'être affecté à plus de trois équipes lorsqu'on insère une nouvelle ligne directement dans la table `EtreAffecte` (pour commencer on ne gèrera pas les modifications de lignes dans cette table).
- Résultat attendu :

```
SELECT codeSalarie, codeEquipe
FROM EtreAffecte
WHERE codeSalarie IN ('S2', 'S8');

CODESALARIE  CODEEQUIPE
------------ -----------
S2           E2
S2           E3
S8           E2
S8           E3
S8           E4
```

L'instruction suivante doit déclencher **une erreur** car le salarié S8 est déjà affecté à 3 équipes.

```
INSERT INTO EtreAffecte VALUES ('S8', 'E1');
Le salarié est déjà affecté à au moins 3 équipes

SELECT *
FROM EtreAffecte
WHERE codeSalarie = 'S8'
AND codeEquipe = 'E1';

aucune ligne sélectionnée
```

Par contre, l'instruction suivante ne déclenchera pas d'erreur et va rajouter une ligne dans la table `EtreAffecte`.

```
INSERT INTO EtreAffecte VALUES ('S2', 'E4');

SELECT *
FROM EtreAffecte
WHERE codeSalarie = 'S2'
AND codeEquipe = 'E4';

CODESALARIE  CODEEQUIPE
------------ -----------
S2           E4
```

13) Mise à jour des attributs calculés lors des modifications, insertions et suppressions.

- On souhaite modifier le trigger de la question 11, afin que l'attribut `nbTotalJourneesTravail` de la table `Salaries` soit mis à jour automatiquement lorsque on insère une nouvelle ligne dans la table `Travailler`, mais aussi lorsqu'on modifie ou on supprime des lignes de cette table.
- Résultat attendu :

```
SELECT codeSalarie, nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie IN ('S1', 'S5');

CODESALARIE  NBTOTALJOURNEESTRAVAIL
---------- - ----------------------
S1             7
S5             8
```

La modification du salarié d'une journée de travail doit modifier le nombre total de journées travaillées des deux salariés impactés par la modification.

```
UPDATE Travailler
SET codeSalarie = 'S5'
WHERE codeSalarie = 'S1'
AND dateTravail = '10/01/2022';


SELECT codeSalarie, nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie IN ('S1', 'S5');

CODESALARIE  NBTOTALJOURNEESTRAVAIL
---------- - ----------------------
S1             6
S5             9
```

La suppression d'une journée de travail du salarié S5 doit décrémenter de 1 son nombre total de journées travaillées.

```
DELETE Travailler
WHERE codeSalarie = 'S5'
AND dateTravail = '10/01/2022';

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S5';

NBTOTALJOURNEESTRAVAIL
----------------------
8
```

14) Trigger permettant de gérer les contraintes sur un héritage

- On souhaite programmer des triggers qui empêchent un projet d'être à la fois interne et externe lorsqu'on insère une nouvelle ligne directement dans les tables `ProjetsExternes` et `ProjetsExternes`.
- Résultat attendu :

```
SELECT codeProjet, 'EXTERNE' AS typeProjet
FROM ProjetsExternes
UNION
SELECT codeProjet, 'INTERNE' AS typeProjet
FROM ProjetsInternes;

CODEPROJET  TYPEPROJET
----------- -----------
P1          EXTERNE
P2          EXTERNE
P3          EXTERNE
P4          EXTERNE
P5          EXTERNE
P6          INTERNE
P7          INTERNE
P8          INTERNE
```

L ajouter un projet externe avec le code P6 doit déclencher **une erreur** car le projet P6 existe déjà dans la table `ProjetsInternes`

```
INSERT INTO ProjetsExternes (codeProjet, nomProjet, villeProjet,
    clientProjet, codeEquipe) VALUES ('P6', 'Robots de discéditation des
                                      footballeurs sur réseaux sociaux',
                                                'Paris', 'PSG', 'S10');
Insertion impossible, un projet interne possède le même code

SELECT nomProjet
FROM ProjetsExternes
WHERE codeProjet = 'P6';

aucune ligne sélectionnée
```

Par contre, il doit être possible de rajouter un projet P9 dans la table `ProjetsExternes`

```
INSERT INTO ProjetsExternes (codeProjet, nomProjet, villeProjet,
                  clientProjet, codeEquipe) VALUES ('P9', 'Site du BUT',
                                                    'Paris', 'IUT', 'E4');

SELECT nomProjet
FROM ProjetsExternes
WHERE codeProjet = 'P9';

NOMPROJET
------------
Site du BUT
```

15) Trigger et contraintes d'inclusion concernant plus de deux associations.

- On souhaite programmer un trigger qui empêche un salarié de travailler sur un projet qui est réalisé par une équipe dans laquelle n'est pas affecté le salarié en question. On veut que ce trigger se déclenche chaque fois que l'on ajoute une journée de travail d'un salarié (si vous avez du temps, vous pourrez ensuite programmer le trigger pour qu'il se déclenche aussi lorsqu'on modifie une journée de travail déjà existante).
- Résultat attendu :

```
SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';

NBTOTALJOURNEESTRAVAIL
----------------------
6
```

Si on indique que le salarié S1 a travaillé le 11/01/2022 sur le projet P1, il ne doit pas y avoir d'erreur.

```
INSERT INTO Travailler VALUES ('S1', '11/01/2022', 'P1');

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';


NBTOTALJOURNEESTRAVAIL
----------------------
7
```

Par contre, si on indique que le salarié S1 a travaillé le 12/01/2022 sur le projet P5, il doit y avoir **une erreur** (car le salarié S1 n'est pas affecté à l'équipe qui travaille sur le projet P5).

```
INSERT INTO Travailler VALUES ('S1', '12/01/2022', 'P5');
Un salarié ne peut pas travailler sur un projet qui est réalisé par
une équipe dans laquelle il n'est pas affecté

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';

NBTOTALJOURNEESTRAVAIL
----------------------
7
```

Cinquième partie – Vues et Triggers :

16) Création d'une vue multitables.

- Créer une vue multitables *Affectations* qui contient toutes les affectations des salariés dans les différentes équipes ; avec pour chaque affectation, le code, le nom et le prénom du salarié ainsi que le code et le nom de l'équipe. Cette vue doit avoir la structure suivante :
  AFFECTATIONS (codeSalarie, nomSalarie, prenomSalarie, codeEquipe, nomEquipe)
- Vérifier que votre vue affiche bien ce qu'il faut. Puis essayer d'insérer des données à travers cette vue. Que se passe-t-il ? Pourquoi ?

```
INSERT INTO Affectations
VALUES ('S9', 'Zétofrais', 'Mélanie', 'E5', 'Indigo');
```

17) Trigger INSTEAD OF

- A l'aide d'un trigger INSTEAD OF, faites qu'il soit possible d'insérer des données à travers la vue multitables précédente. Si le salarié de l'affectation qui est inséré à travers la vue n'existe pas, il doit être créé dans la table *Salariés* (avec un nombre de journées de travail égal à 0). De même, si l'équipe de l'affectation qui est insérée à travers la vue n'existe pas, elle doit être elle aussi créée dans la table *Equipes* (avec un chef d'équipe égal à NULL). Dans tous les cas, une ligne doit être ajoutée dans la table *EtreAffecte*.
- Résultat attendu (les salariés et les équipes indiquées en gras n'existent pas dans la base de données) :

```
INSERT INTO Affectations
VALUES ('S20', 'Zétofrais', 'Mélanie', 'E5', 'Indigo');

INSERT INTO Affectations
VALUES ('S20', 'Zétofrais', 'Mélanie', 'E4', 'Mars');

INSERT INTO Affectations
VALUES ('S5', 'Umule', 'Jacques', 'E20', 'Europa');

INSERT INTO Affectations
VALUES ('S21', 'Zeblouse', 'Agathe', 'E21', 'Galileo');

SELECT *
FROM EtreAffecte
ORDER BY codeSalarie;

CODESALARIE  CODEEQUIPE
------------ -----------
…            …
S21          E21
…            …
S5           E20
…            …
S20           E4
S20           E5
```

- Modifier votre trigger INSTEAD OF pour qu'une vérification soit effectuée sur la cohérence des données insérées à travers votre vue. Ainsi, si le code du salarié passé en paramètre existe dans la table *Salariés* mais que son nom et son prénom ne correspondent pas à ce qu'il y a dans cette table, rien ne doit être inséré dans la base de données (ni salarié, ni équipe, ni affectation) et une exception est levée. De même, si le code de l'équipe passé en paramètre existe dans la table *Equipes* mais que son nom ne correspond pas à ce qu'il y a dans cette table, rien ne doit être inséré dans la base de données (ni salarié, ni équipe, ni affectation) et une exception est levée.
- Résultat attendu :

```
INSERT INTO Affectations
VALUES ('S20', 'Ouzy', 'Jacques', 'E6', 'Europa');
Les données sur le salarié S20 sont fausses

INSERT INTO Affectations
VALUES ('20', 'Zétofrais', 'Mélanie', 'E20', 'Galileo');
Les données sur l'équipe E20 sont fausses
```

Sixième partie pour les plus rapides – Requêtes SQL faciles :

R1 : le nom et le prénom des salariés qui ont travaillé le 4 janvier 2022 et le 8 janvier 2022 (ils doivent avoir travaillé à ces deux dates).

```
NOMSALARIE  PRENOMSALARIE
----------- --------------
Deuf        John
Umule       Jacques
```

R2 : le nom et le prénom des salariés qui partagent le même prénom qu'un autre salarié.

```
NOMSALARIE  PRENOMSALARIE
----------- --------------
Titouplin   Jean
Menage      Jean
```

R3 : le nombre de salariés affectés à chaque équipe.

```
NOMEQUIPE  NBSALARIES
---------- -----------
Luna       6
Galileo    1
Ganymède   4
Europa     1
Juno       5
Indigo     1
Mars       6
```

R4 : le nom de l'équipe qui possède le plus de salariés affectés.

```
NOMEQUIPE
----------
Luna
Mars
```

R5 : le nom et le prénom des salariés qui sont affectés dans tous les projets externes où le salarié Jean Menage est affecté.

```
NOMSALARIE  PRENOMSALARIE
----------- --------------
Deuf        John
Gator       Ali
Stiké       Sophie
```

R6 : le nom et le prénom des salariés qui sont exactement affectés aux même projets externes que Jean Menage.

```
NOMSALARIE  PRENOMSALARIE
----------- --------------
Gator       Ali
```

R7 : pour chacune des villes où il y a un ou des projets externes réalisés, indiquer le nom du projet qui possède le plus de salariés affectés à son équipe.

```
VILLEPROJET  NOMPROJETJ
------------ ----------------------------------------------
Montpellier  Logiciel Sécurité Sociale
Béziers      Outil CRM de la Poste
Nimes        Site du club de billes des crocodiles ramollis
Paris        Site du BUT
```

R8 : le nom et le prénom des salariés qui sont des Ninjas SQL.

```
NOMSALARIE  PRENOMSALARIE
----------- --------------------------
Palleja     Xavier
Palleja     Nathalie
```