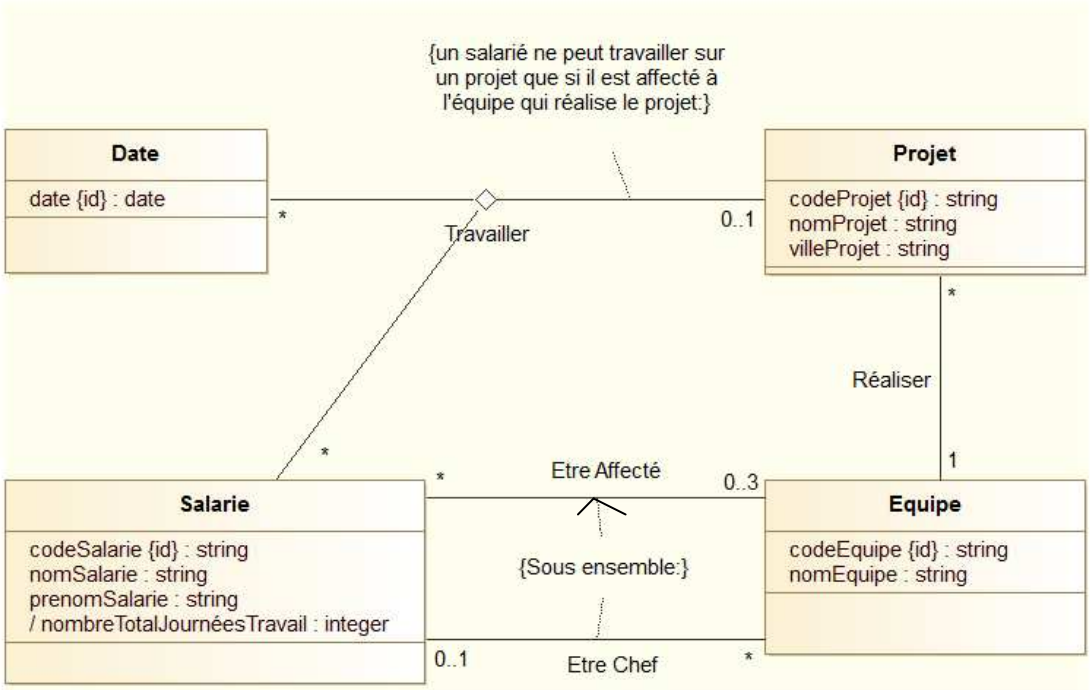
❾ - UNIDEV

Dans cet exercice, on s'intéresse à l'ESN (Entreprise de Services du Numérique) UNIDEV™. Plus exactement, on souhaite implanter une base de données qui permettra de gérer les ressources humaines de cette entreprise afin de savoir à quelles équipes sont affectés les différents salariés et sur quels projets ils ont travaillé.

Lors de la phase d'analyse, on a obtenu le diagramme de classes conceptuel (ou diagramme des classes persistantes) suivant :



A partir de ce diagramme de classes, on a déduit le schéma relationnel suivant :

SALARIES (codeSalarie, nomSalarie, prenomSalarie, *nbTotalJourneesTravail*)
EQUIPES (codeEquipe, nomEquipe, codeSalarieChef#)
PROJETS (codeProjet, nomProjet, villeProjet, codeEquipe#)
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.

Aller sur le 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. On vous rappelle que sous iSQL*Plus, pour ne pas avoir de problème avec les accents, il est recommandé de faire un copier-coller du contenu de ce fichier et de ne pas utiliser le bouton « Charger script ».

Première partie – Gestion des contraintes sur les tables :

1) 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 les cas où on modifie ou supprime des lignes de cette table).

A. Procédure stockée :

- 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 :

```
CALL AjouterJourneeTravail('S2','P3', '10/01/2014');

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S2';

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

B. Trigger :

- 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 :

```
INSERT INTO Travailler VALUES ('S1', '10/01/2014', 'P1');

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';

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

2) 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 les cas où on modifie des lignes de cette table).

A. Procédure stockée :

- 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 :

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

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

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

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
```

B. Trigger :

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

- Résultat attendu :

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

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

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

INSERT INTO EtreAffecte VALUES ('S7', 'E4');
Le salarié est déjà affecté à au moins 3 équipes

SELECT *
FROM EtreAffecte
WHERE codeSalarie = 'S7' AND codeEquipe = 'E4';

aucune ligne sélectionnée
```

3) Mise à jour des attributs calculés lors des modifications, insertions et suppressions.
   - On souhaite modifier le trigger de la question 1.B., 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 :

```
UPDATE Travailler
SET codeSalarie = 'S5'
WHERE codeSalarie = 'S1' AND dateTravail = '10/01/2014';

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S1';

NBTOTALJOURNEESTRAVAIL
----------------------
6

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S5';

NBTOTALJOURNEESTRAVAIL
----------------------
9

DELETE Travailler
WHERE codeSalarie = 'S5'
AND dateTravail = '10/01/2014';

SELECT nbTotalJourneesTravail
FROM Salaries
WHERE codeSalarie = 'S5';

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

4) Contraintes entre 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 (on ne souhaite pas gérer le cas où on modifie une journée de travail déjà existante).
   - Résultat attendu :

     Si on indique que le salarié S1 a travaillé le 11/01/2014 sur le projet P1, il ne doit pas y avoir d'erreur. Ensuite, si on exécute la requête suivante, on obtiendra :

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

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

     Puis, si on indique que le salarié S1 a travaillé le 12/01/2014 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). Ensuite, si on exécute la requête suivante, on obtiendra :

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

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

Deuxième partie – Vues et Triggers :

5) 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 ?

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

6) 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 :

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

INSERT INTO Affectations
VALUES ('S9', 'Zétofrais', 'Mélanie', 'E4', 'Mars');

INSERT INTO Affectations
VALUES ('S5', 'Umule', 'Jacques', 'E6', 'Europa');

INSERT INTO Affectations
VALUES ('S10', 'Zeblouse', 'Agathe', 'E7', 'Galileo');

SELECT *
FROM EtreAffecte

CODESALARIE  CODEEQUIPE
------------ -----------
…            …
S10          E7
…            …
S5           E6
…            …
S9           E4
S9           E5
```