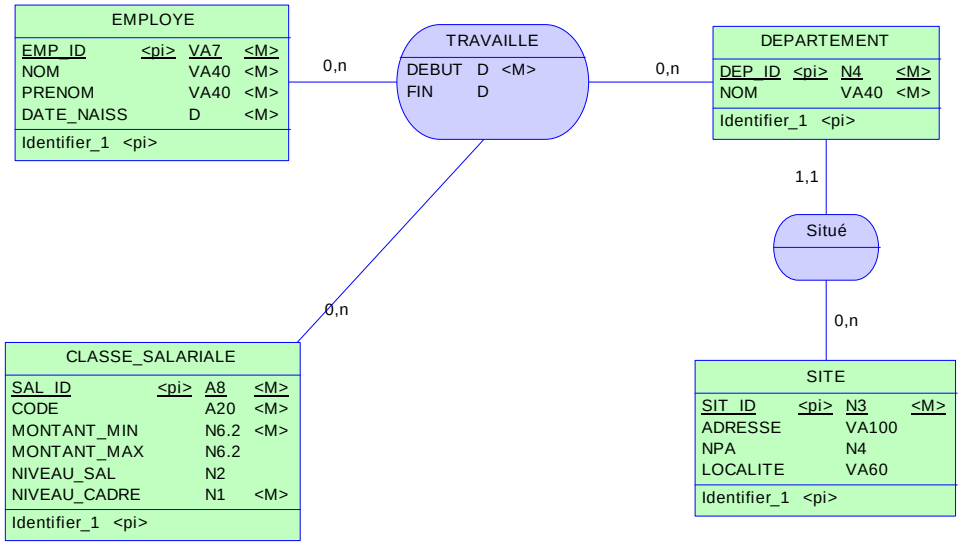
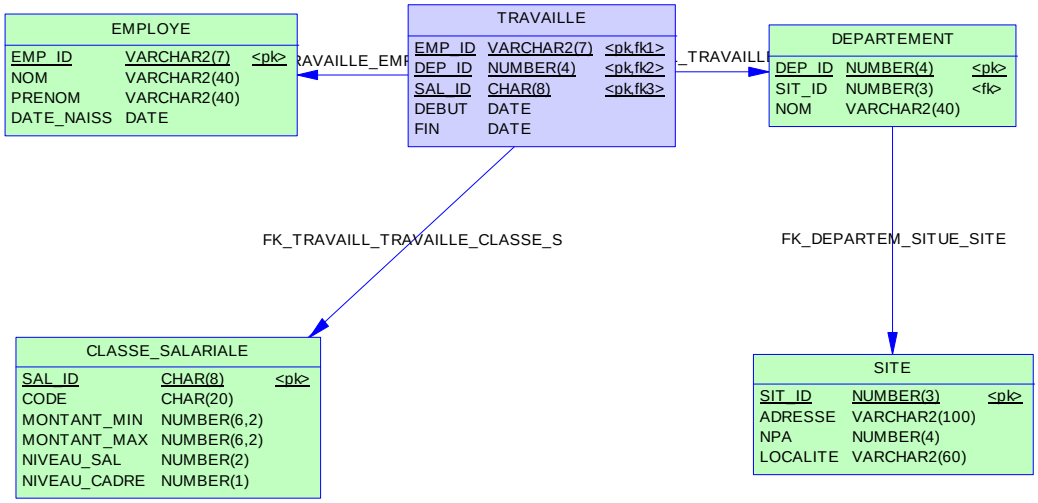
Examen blanc

Pour les exercices 1 et 2, considérez le modèle suivant (MCD et MLD) :



Modèle conceptuel de données



Modèle logique de données

Règles et informations supplémentaires :

- la clé étrangère entre employe et travaille est de type ‘on delete cascade’,
- Notez qu’un employé ne peut pas après la fin de son contrat (TRAVAILLE) être réengagé avec le même niveau de salaire dans le même département (clé de TRAVAILLE).
- le niveau de salaire (NIVEAU_SAL) doit se situer entre 1 et 5,
- la notion de cadre (NIVEAU_CADRE) est par défaut 0, c'est-à-dire non,
- les champs obligatoires sont notés M sur le MCD,
- la clé primaire de TRAVAILLE se nomme PK_TRAVAILLE,
- la clé étrangère entre TRAVAILLE et CLASSE_SALARIALE est « restrict ».

Exercice 1

On vous demande, pour des raisons d'historique, de stocker le nom des employés en cas de changement (mariage par exemple).
Vous pouvez insérer dans la table EMPLOYE_ HISTO, les mises à jour de nom. Cette table a déjà été créée pour vous.

Structure de la table EMPLOYE_ HISTO et extrait de la table :

EMP_HISTO_ID	EMP_ID	ANCIEN_NOM	DATE_MUTATION
1	EFR8920	Balmin	12.08.2007
2	IVF3920	Bigg	01.01.2007
3	EFR8920	Hugo	22.09.2007
4	…		

Veuillez mettre en place le mécanisme nécessaire pour répondre à ce besoin.
Vous devez aussi gérer la clé primaire de la table EMPLOYE_HISTO de façon automatique.

CREATE SEQUENCE seq_pk_emp_histo START WITH 1; (2 points)

```
CREATE OR REPLACE TRIGGER tr_au_employe
AFTER UPDATE
ON EMPLOYE
FOR EACH ROW
WHEN (old.nom <> new.nom)
BEGIN
  INSERT INTO EMPLOYE_HISTO VALUES (seq_pk_emp_histo.nextval, :old.emp_id, :old.nom, sysdate);
END;
```

Ou avec un IF dans le corps du trigger.

AFTER UPDATE 2 point

WHEN ou IF 2 points

INSERT 2 points

Syntaxe globale (yc replace) 2 points

Exercice 2

Automatisez à l'aide d'unité(s) de programmation PL/SQL l'augmentation salariale des employés ayant plus de 10 ans d'expérience pour un travail donné dans l'entreprise.

Une augmentation de salaire signifie passer d'un salaire de niveau n à un salaire de niveau n+1 (classe_salariale.niveau_sal). Les employés en niveau 5 (maximum de classe_salariale.niveau_sal) n'ont pas le droit à une augmentation de salaire.
Notez qu'il n'y a que 5 enregistrements dans la table CLASSE_SALARIALE (correspondant aux 5 niveaux de salaire).

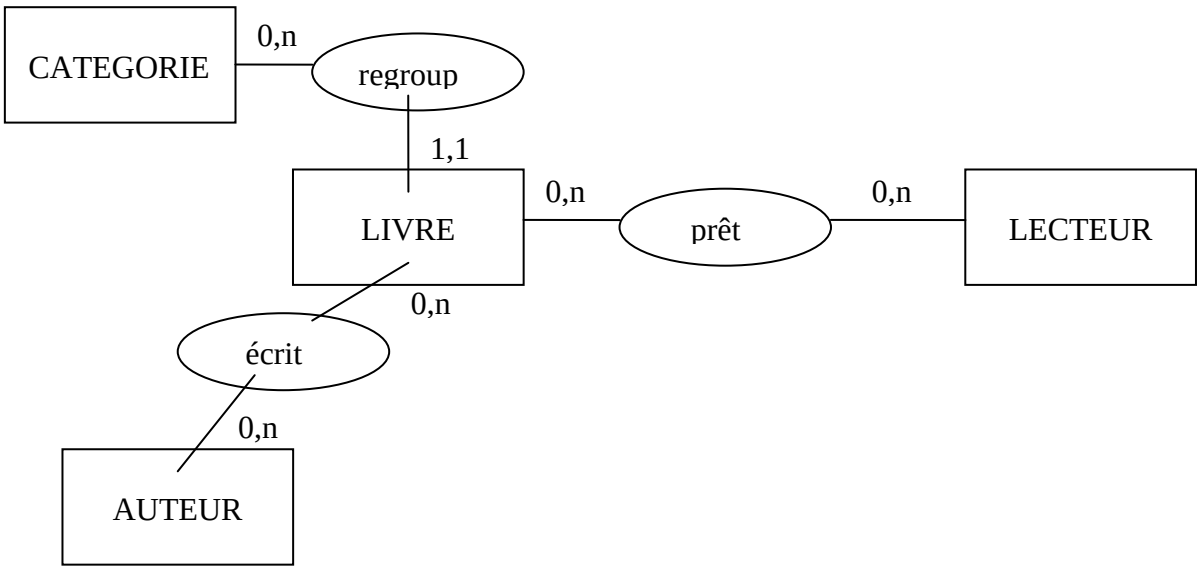
Lors d'une augmentation de salaire, vous devez mettre une date de fin au contrat de votre employé dans la table TRAVAILLE puis insérer une nouvelle ligne (nouveau contrat) avec la date du jour comme date de début et avec la nouvelle classe salariale (sal_id).

```
CREATE OR REPLACE PROCEDURE p_augmentation_salaire
IS
  CURSOR cur_employe_a_augmenter
  IS
  SELECT employe.emp_id, travaille.dep_id, travaille.sal_id, classe_salariale.niveau_sal
    FROM employe, travaille, classe_salariale
    WHERE employe.emp_id = travaille.emp_id AND
              travaille.sal_id = classe_salariale.sal_id AND
              classe_salariale.niveau_sal < 5 AND
              (to_number(to_char(sysdate,'yyyy'),'9999') - to_number(to_char(travaille.debut,'yyyy'),'9999')) >= 10
FOR UPDATE; (1 point)
(3 points)
var_nouveau_salaire classe_salariale.sal_id%type; (1 point)


BEGIN

  FOR emp_augmentation IN cur_employe_a_augmenter LOOP (2 point (1 pour emp_augmentation, 1 pour la boucle)
    -- Mettre une date de fin au contrat
    UPDATE travaille SET fin = sysdate
    WHERE emp_id = emp_augmentation.emp_id AND
              dep_id = emp_augmentation.dep_id AND
              sal_id = emp_augmentation.sal_id;
(1 point)
    -- Créer un nouveau contrat
    BEGIN
      SELECT sal_id INTO var_nouveau_salaire FROM classe_salariale
      WHERE niveau_sal = emp_augmentation.niveau_sal + 1 ; (1 point)
    EXCEPTION
      WHEN NO_DATA_FOUND THEN (1 point)
        rollback ;
        raise_application_errors(-20001,'Pas de salaire disponible…');
    END ;

    INSERT INTO travaille (emp_id, dep_id, debut, sal_id) values (emp_augmentation.emp_id,
emp_augmentation.dep_id, sysdate , var_nouveau_salaire) ; (1 point)
  END LOOP ;

COMMIT ; (1 point)

END ;
```

Donnée de base pour les exercices 3 et 4 :

On désire mettre en place une base de données pour gérer les prêts de livres d'une bibliothèque.

Le modèle de la base de données implantée est le suivant.



On considère la base de données existante et peuplée.

Les tables résultantes de ce modèle sont les suivantes :

categories	cat_numero	Number(9)	PK
	cat_libelle	Varchar(30)	
	cat_temp_pret	Number(3)	>0 (en nombre de jours)

auteurs	aut_numero	Number(9)	PK
	aut_nom	Varchar(30)	
	aut_prenom	Varchar(30)	
	aut_date_naiss	Date	

livres	liv_numero	Number(9)	PK
	liv_code_isbn	Varchar(20)	Unique
	liv_titre	Varchar(40)	
	liv_etat	Varchar(15)	['BON','MAUVAIS',INUTILISABLE']
	liv_cat_num	Number(9)	FK sur cat_numero

ecrit	ecr_liv_num	Number(9)	PK partielle + FK liv_num
	ecr_aut_num	Number(9)	PK partielle + FK aut_num
	ecr_date	Date	

lecteurs	lect_matricule	Number(7)	PK
	lect_nom	Varchar(30)	
	lect_prenom	Varchar(30)	

prets	pre_lect_matr	Number(7)	PK partielle + FK lect_matricule
	pre_liv_num	Number(9)	PK partielle + FK liv_numero
	pre_date_pret	Date	non null
	pre_date_retour	Date	

Exercice 3

Ajouter dans le schéma, les règles de gestions suivantes :

Si un lecteur est effacé, on effacera automatiquement tous ses prêts, à condition qu'il n'en possède pas actuellement (date de retour nulle !).
Si un lecteur a des prêts en cours, le système doit alors refuser l'effacement et produire un message d'erreur standard.

```
CREATE OR REPLACE TRIGGER ex3
BEFORE DELETE ON lecteur
FOR EACH ROW
DECLARE
nb_pret_en_cours Number(5) ;
pret_en_cours Exception ;

BEGIN
SELECT Count(*)
 INTO nb_prets_en_cours
 FROM prets
WHERE pre_lect_matr = :Old.lect_matricule
AND pre_date_retour IS Null ;

IF nb_prets_en_cours > 0 THEN
 RAISE pret_en_cours ;
ELSE
 DELETE FROM prets
  WHERE pret_lect_matr = :Old.lect_matricule ;
END IF ;
EXCEPTION
 WHEN pret_en_cours THEN
  Raise_Application_Error (-20001,'Pret en cours') ;
END ;
```

Exercice 4

Créer un outil permettant de générer des rappels pour les retardataires.

L'exécution de cette procédure va alimenter une table RAPPELS dont la structure est la suivante

Rappels	Rap_numero	Number(7)	PK , autoincrémenté par sequence
	Rap_niveau	Number(1)	[1,2,3]
	Rap_date	Date	
	Rap_lect_matr	Number(7)	FK lect_matricule
	Rap_liv_num	Number(9)	FK liv_numero

L'attribut rap_niveau signifie s'il s'agit du 1er rappel, du 2ième rappel ou du 3ième
rap_date contient la date à laquelle les rappels ont été détectés.

La procédure fonctionne de la manière suivante.
Elle va chercher les différents prêts en cours, et si la date limite (calculée à partir de la catégorie du livre) est dépassée, on créer un rappel.

S'il existe déjà un rappel pour ce prêt, on incrémente le niveau du rappel, que si le précédent a été fait depuis plus de 10 jours, et bien entendu, qu'il ne s'agit pas déjà du troisième. (S'il s'agit déjà du troisième, la procédure ne modifie pas son état)

On dispose des éléments suivants

La séquence existe (SEQ_RAPPELS)

L'implémentation des objets se trouveront dans le package dont voici la signature :

```
PACKAGE biblio IS

  PROCEDURE generer_rappels ;

END ;
```

Pour l'implémentation de cette procédure, il est conseillé d'écrire deux procédures,

1) nouveau_rappels qui insert les nouveaux rappels dans la table.
2)ancien_rappels qui met à jour le niveau du rappel des tuples existants.

Travail à effectuer : Créer la table rappels et package body en fonction des spécifications demandées.

```
CREATE TABLE rappels (  rap_numero Number(7) CONSTRAINT pk_rappels PRIMARY KEY ,
                        rap_niveau Number(1) CONSTRAINT ch_rap_niveau
                                CHECK (rap_niveau BETWEEN 1 AND 3 ),
                        rap_date   Date ,
                        rap_lect_matr Number(9) ,
                        rap_liv_num Number(9) ,
                        CONSTRAINT fk_rap_livre FOREIGN KEY rap_liv_num
                                REFERENCES livres ,
                        CONSTRAINT fk_rap_lecteur FOREIGN KEY rap_liv_num
                                REFERENCES livres ) ;
```

```
CREATE OR REPLACE PACKAGE BODY biblio IS
 PROCEDURE nouveau_rappels IS
  CURSOR cur_n IS SELECT pret_liv_num, pret_lect_matr
                  FROM livres, prets, categories
                 WHERE liv_numero = pre_liv_num
                  AND liv_cat_num = cat_numero
                  AND (Sysdate - pre_date_pret) > cat_temp_pret
                  AND pret_date_retour IS Null
                  AND liv_num NOT IN (SELECT rap_liv_num
                                      FROM rappels
                                      WHERE rap_lect_matr <> pret_lect_matr) ;
  cle rappels.rap_numero%Type ;
BEGIN
FOR emp_rec IN cur_n LOOP
  SELECT seq_rappels.Nextval
    INTO cle
   FROM dual ;

  INSERT INTO rappels ( rap_numero,
                        rap_niveau,
                              rap_date,
                                  rap_lect_matr,
                                  rap_liv_num)
                       VALUES( cle ,
                                  1,
                                  Sysdate,
                                  emp_rec.pret_lect_matr,
                                  emp_rec.pret_liv_num ) ;

END LOOP ;
END nouveau_rappels ;

PROCEDURE ancien_rappels
   CURSOR cur_a IS SELECT rap_niveau, rap_date
                   FROM prets, rappels
                  WHERE pre_liv_num = rap_liv_num
                   AND pret_date_matr = rap_lect_matr
                   AND pret_date_retour IS Null
                          AND rap_niveau < 3
                   AND (Sysdate - rap_date) > 10
                                  FOR UPDATE ;

BEGIN
 FOR emp_rec IN cur_a LOOP
  UPDATE rappels SET rap_niveau = rap_niveau + 1 ,
                     rap_date = Sysdate
            WHERE CURRENT OF cur_a ;

 END LOOP ;
END ancien_rappels ;

PROCEDURE generer_rappels IS
BEGIN
 nouveau_rappels ;
 ancien_rappels ;
 COMMIT ;
END generer_rappels ;

END ;
```