Université de Montpellier
L3 Informatique

Création de la base de données

"SQUID GAME"

Rapport de projet
Projet informatique — HLIN405

Étudiants :
21802561 Enzo Goulesque
21805058 Mihai Poitelea
22008650 Heiarii Collenot
21605028 Margaux Renoir

Groupe B
Encadrant :
M. Michel Guillaume

Année : 2021 – 2022

Table des matières

Chapitre 1

Description de la base de données

Nous avons choisis de faire notre base de données sur le thème du "Squid game".

Squid game est une série dramatique de survie sud-coréenne qui est sortie en 2021. Elle raconte l'histoire d'un groupe de personnes, fortement endettées, voire ruinées, qui sont invitées par une mystérieuse organisation à jouer à six jeux pendant une durée de six jours afin d'obtenir une énorme somme d'argent. Toutefois, il y a un piège, si le participant perd, il meurt.
Voici différentes choses à connaître pour mieux comprendre notre base :

- dans le Squid Game, des investisseurs peuvent parier sur des joueurs lors des différents jeux.
- si les joueurs meurent tous, la somme d'argent qui était a gagner revient à leur famille, d'où l'intérêt d'avoir une table avec les informations des familles et leur iban.
- Les gardes n'ont pas de famille renseignée car ils ne participent pas aux jeux et n'ont donc pas d'argent à gagner.

Nous avons crée la base de données qui pourrait servir aux organisateurs du Squid Game pour gérer les différents jeux, retrouver facilement les paris, savoir combien de joueurs sont mort/encore en vie et lesquels mais aussi connaître les amitié(s) entre les joueurs, stocker les informations sur les familles des joueurs, etc

Chapitre 2

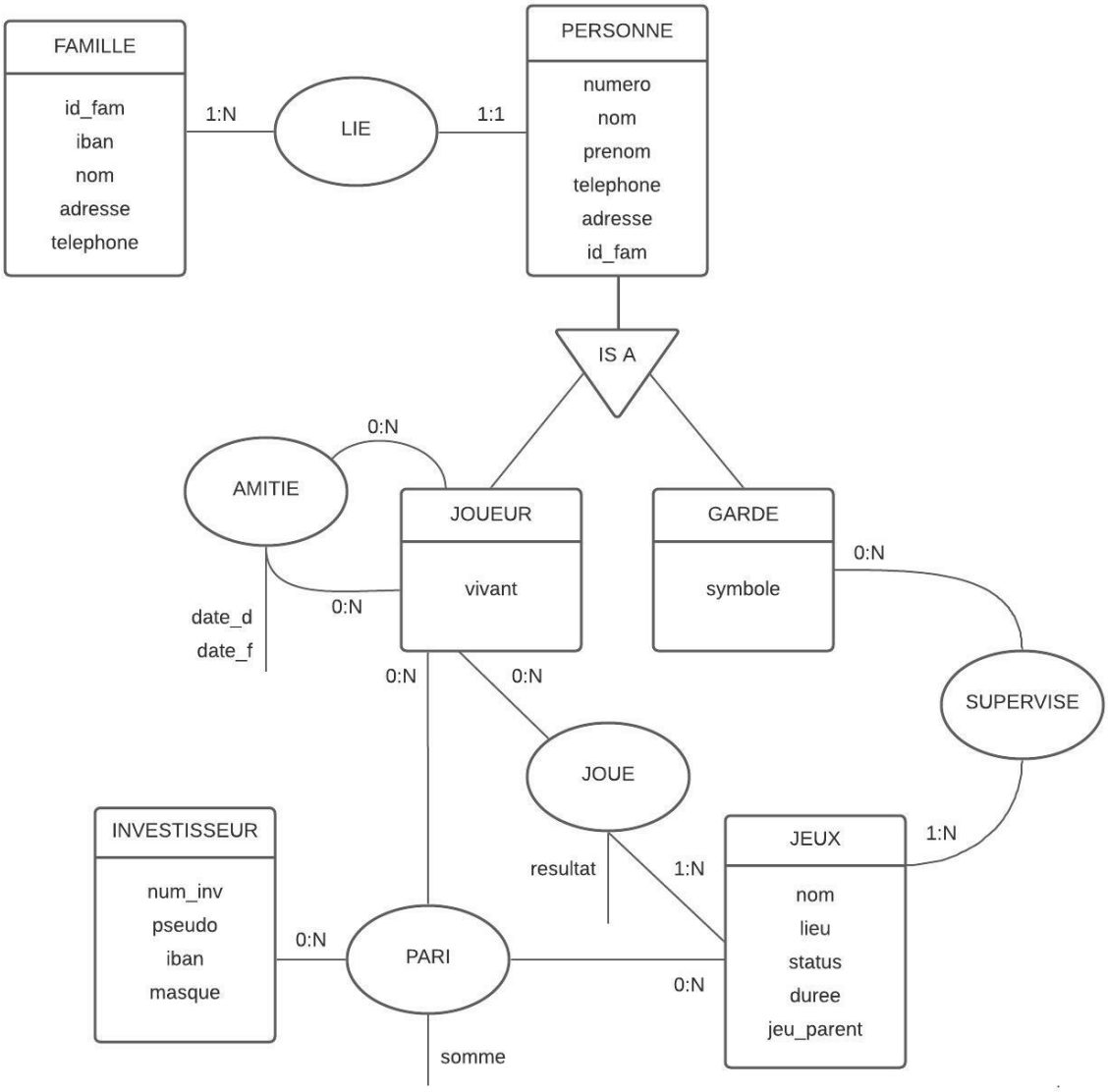
Dictionnaire des données

TABLE	NOM_SYMBOLIQUE	DESCRIPTION	DOMAINE	COMMENTAIRE	CONTRAINTES / REGLES DE CALCUL	CLE PRIMAIRE	CLE ETRANGERE
FAMILLE							
	IDFAM	ID de la famille	NUMERIC (6,0)	Format : 999999	OBLIGATOIRE	X	
	IBAN	Numero de l'iban	VARCHAR (30)		OBLIGATOIRE		
	NOM	Nom de la famille du joueur	VARCHAR (50)		OBLIGATOIRE		
	ADRESSE	Adresse de la famille du joueur	VARCHAR (50)	Format : X rue de / avenue X	FACULTATIF		
	TELEPHONE	Numero de la famille du joueur	VARCHAR (20)	Format : 9999999999	FACULTATIF		
PERSONNE							
	NUMERO	ID du joueur ou du garde	NUMERIC (6,0)	Format : 999999	OBLIGATOIRE	X	
	NOM	Nom du joueur ou du garde	VARCHAR(15)		OBLIGATOIRE		
	PRENOM	Prenom du joueur ou du garde	VARCHAR(20)		FACULTATIF		
	TELEPHONE	Numero de telephone du joueur ou du garde	VARCHAR(20)	Format : 9999999999	FACULTATIF		
	ADRESSE	Adresse du joueur ou du garde	VARCHAR(30)	Format : X rue de / avenue X	FACULTATIF		
	SYMBOLE	Symbole sur le masque du garde Il sert a savoir sa position dans la hierarchie (Carre > Triangle > Rond)	VARCHAR(10)	Format : ROND CARRE ou TRIANGLE	FACULTATIF PARMI : 'ROND', 'CARRE', 'TRIANGLE'		
	VIVANT	Indique si le joueur est vivant ou non (0 : mort ; 1 : vivant)	NUMBER (1)	Format : 0 ou 1	FACULTATIF PARMI : 0, 1		
	IDFAM	ID de la famille	NUMERIC (6,0)	Format : 999999	FACULTATIF car les gardes n'ont pas de famille		X
AMITIE							
	NUM1	ID du joueur1	NUMERIC (6,0)	Format : 999999	OBLIGATOIRE	X	X
	NUM2	ID du joueur2	NUMERIC (6,0)	Format : 999999	OBLIGATOIRE	X	X
	DATE_DEB	Date du début de leur amitié	DATE	Format : dd-mm-yyyy	OBLIGATOIRE		
	DATE_FIN	Date du fin de leur amitié	DATE	Format : dd-mm-yyyy	FACULTATIF		
JEUX							
	NOM	Nom du jeu	VARCHAR (25)		OBLIGATOIRE	X	
	LIEU	Lieu où se déroule le jeu	VARCHAR (25)		OBLIGATOIRE		
	DUREE	La durée en minutes du jeu	NUMERIC (4,0)	Format : 9999	FACULTATIF		
	STATUS	Indique si le jeu est fini, en cours ou à venir	VARCHAR (10)	Format : 'INI' 'EN_COURS' ou 'A_VENI	OBLIGATOIRE PARMI : 'FINI', 'EN_COURS', 'A_VENIR'		
	JEU_PARENT	Indique le jeu précédent pour savoir l'odre des jeux	VARCHAR(25)		FACULTATIF		X
JOUE							
	NUM_JOUEUR	ID du joueur qui joue / a joué	NUMERIC (4,0)	Format : 999999	OBLIGATOIRE	X	X
	NOM_JEU	Nom du jeu auquel il joue	VARCHAR (25)		OBLIGATOIRE	X	X
	RESULTAT	Permet de savoir si le joueur a survecut au jeu ou si il est mort	VARCHAR (15)	Format : 'A_SURVECU' ou 'MORT'	FACULTATIF PARMI : 'A_SURVECU', 'MORT'		
SUPERVISE							
	NUM_GARDE	ID du garde	NUMERIC (4,0)	Format : 999999	OBLIGATOIRE	X	X
	NOM_JEU	Nom du jeu qu'il supervise	VARCHAR (25)	Format :	OBLIGATOIRE	X	X
INVESTISSEUR							
	NUM_INV	ID de l'investisseur	NUMERIC (6,0)	Format : Format : 999999	OBLIGATOIRE	X	
	PSEUDO	Pseudo de l'investisseur	VARCHAR (15)		OBLIGATOIRE		
	IBAN	IBAN de l'investisseur	VARCHAR (30)		OBLIGATOIRE		
	MASQUE	Animal représenté par le masque Permet de différencier les investisseurs	VARCHAR (20)		FACULTATIF		
PARI							
	NUM_INV	ID de l'investisseur	NUMERIC (4,0)	Format : Format : 999999	OBLIGATOIRE	X	X
	NOM_JEU	Nom du jeu sur lequel il va parier	VARCHAR (30)		OBLIGATOIRE	X	X
	NUM_JOUEUR	Nom du joueur sur lequel il mise	NUMERIC (6,0)	Format : Format : 999999	OBLIGATOIRE		X
	SOMME	Le montant d'argent qu'il mise	NUMERIC (10,0)	Format : 9999999999	OBLIGATOIRE		

Chapitre 3

Les schémas

3.1 Schéma entité-association



Description du schéma entité-association :

— Une personne est soit un joueur soit un garde.

— Une personne ne peut avoir qu'une famille mais une famille peut contenir plusieurs personnes.

— Les joueurs peuvent être amis entre eux (relation réflexive).

— Un ou plusieurs joueurs peuvent jouer à un ou plusieurs jeux mais pas en même temps. Un jeu doit être joué au moins par un joueur. Le résultat du jeu est soit 'A_SURVECU' soit 'MORT'.

— Un ou plusieurs gardes peuvent surveiller des jeux et il doit y avoir au moins un garde par jeu.

— Enfin nous avons mis en place un système de pari, où des investisseurs peuvent parier sur la victoire d'un ou plusieurs joueurs dans un jeu.

3.2 Schéma relationnel

De ce schéma entité-association, nous avons tiré un schéma relationnel :

FAMILLE :(id_fam, iban, nom, adresse, telephone)
PERSONNE :(numero,nom, #id_fam ,prenom, telephone, adresse, symbole, vivant)
AMITIE :(#num1,#num2, date_deb, date_fin)
JEUX :(nom, lieu, duree, status)
JOUE :(#num_joueur, #nom_jeu, resultat)
SUPERVISE :(#num_garde, #nom_jeu)
INVESTISSEUR :(num_inv,Pseudo, iban, masque)
PARI :(#num_inv, #num_jeu, #num_joueur, somme)

3.3 Schéma physique

Le schéma physique associé (les tables, attributs)

```
CREATE TABLE FAMILLE (
    ID_FAM NUMERIC(6,0),
    IBAN VARCHAR(30),
    NOM VARCHAR(50),
    ADRESSE VARCHAR(50),
    TELEPHONE VARCHAR(20),
    CONSTRAINT PK_FAMILLE PRIMARY KEY (IDFAM)
);

CREATE TABLE PERSONNE (
    NUMERO  NUMERIC(6,0),
    NOM VARCHAR(50),
    PRENOM VARCHAR(20),
    TELEPHONE VARCHAR(20),
    ADRESSE VARCHAR(30),
    SYMBOLE VARCHAR(10),CHECK (SYMBOLE IN ('ROND','CARRE','TRIANGLE')),
    VIVANT NUMBER(1), CHECK (VIVANT IN (0,1)),
    ID_FAM NUMERIC(6,0),
    CONSTRAINT PK_PERSONNE PRIMARY KEY (NUMERO),
    CONSTRAINT FK_IDFAM FOREIGN KEY (IDFAM) REFERENCES FAMILLE(IDFAM)
);

CREATE TABLE AMITIE (
    NUM1 NUMERIC(6,0),
    NUM2 NUMERIC(6,0),
    DATE_DEB DATE NOT NULL,
    DATE_FIN DATE,
    CONSTRAINT PK_AMITIE PRIMARY KEY (NUM1,NUM2),
    CONSTRAINT FK_NUMERO1 FOREIGN KEY (NUM1) REFERENCES PERSONNE(NUMERO),
    CONSTRAINT FK_NUMERO2 FOREIGN KEY (NUM2) REFERENCES PERSONNE(NUMERO)
);

CREATE TABLE JEUX (
    NOM VARCHAR(25),
    LIEU VARCHAR(25) NOT NULL,
    DUREE NUMERIC(4,0),
    STATUS VARCHAR(10), CHECK (STATUS IN ('FINI','EN_COURS','A_VENIR')),
    JEU_PARENT VARCHAR(25),
    CONSTRAINT PK_JEUX PRIMARY KEY (NOM),
    CONSTRAINT FK_NOMPRE FOREIGN KEY (JEU_PARENT) REFERENCES JEUX(NOM)
);
```

```sql
CREATE TABLE JOUE (
    NUM_JOUEUR NUMERIC(4,0),
    NOM_JEU VARCHAR(25),
    RESULTAT  VARCHAR(15), CHECK(RESULTAT IN('A_SURVECU', 'MORT')),
    CONSTRAINT PK_JOUE PRIMARY KEY (NUM_JOUEUR,NOM_JEU),
    CONSTRAINT FK_NUMERO_JOUE FOREIGN KEY (NUM_JOUEUR) REFERENCES PERSONNE(NUMERO),
    CONSTRAINT FK_NOM_JEU_JOUE FOREIGN KEY (NOM_JEU) REFERENCES JEUX(NOM)
);

CREATE TABLE SUPERVISE (
    NUM_GARDE NUMERIC(4,0),
    NOM_JEU VARCHAR(25),
    CONSTRAINT PK_SUPERVISE PRIMARY KEY (NUM_GARDE,NOM_JEU),
    CONSTRAINT FK_NUM_GARDE FOREIGN KEY (NUM_GARDE) REFERENCES PERSONNE(NUMERO),
    CONSTRAINT FK_NOM_JEU_SUPERVISE FOREIGN KEY (NOM_JEU) REFERENCES JEUX(NOM)
);

CREATE TABLE INVESTISSEUR (
    NUM_INV NUMERIC(6,0),
    PSEUDO VARCHAR(15) NOT NULL,
    IBAN VARCHAR(30) NOT NULL,
    MASQUE VARCHAR(20),
    CONSTRAINT PK_INVESTISSEUR PRIMARY KEY (NUM_INV)
);

CREATE TABLE PARI (
    NUM_INV NUMERIC(4,0),
    NOM_JEU VARCHAR(30),
    NUM_JOUEUR NUMERIC(6,0) NOT NULL,
    SOMME NUMERIC(10,0) NOT NULL,
    CONSTRAINT PK_PARI PRIMARY KEY (NUM_INV,NOM_JEU),
    CONSTRAINT FK_NUM_INV FOREIGN KEY (NUM_INV) REFERENCES INVESTISSEUR(NUM_INV),
    CONSTRAINT FK_NOMJEU_PARI FOREIGN KEY (NOM_JEU) REFERENCES JEUX(NOM),
    CONSTRAINT FK_NUM_JOUEUR_PARI FOREIGN KEY (NUM_JOUEUR) REFERENCES PERSONNE(NUMERO)
);
```

Chapitre 4

Les requêtes

4.1 Requête 1

Nombre de pari sur les jeux passés ou en cours

```
SELECT COUNT(NUM_JOUEUR),NOM_JEU FROM PARI GROUP BY NOM_JEU;
```

4.2 Requête 2

Joueur(s) dont le nom de famille commence par la lettre A

```
SELECT PRENOM,NOM,NUMERO FROM PERSONNE WHERE NOM LIKE 'A%';
```

4.3 Requête 3

Gardes qui ont supervisé le deuxième jeu

```
SELECT PRENOM,NOM,NUMERO,SYMBOLE FROM PERSONNE
WHERE NUMERO IN
(SELECT NUM_GARDE FROM SUPERVISE WHERE NOM_JEU = 'BISCUIT');
```

4.4 Requête 4

Pseudo de la personne qui a parié la plus grande somme

```
SELECT PSEUDO FROM INVESTISSEUR
WHERE NUM_INV =
    (SELECT NUM_INV FROM PARI
     WHERE SOMME >=
        (SELECT MAX(SOMME) FROM PARI));
```

4.5 Requête 5

Joueurs vivants sur lesquels personne n'a parié

```
SELECT PRENOM, NOM, NUMERO FROM PERSONNE
WHERE NOT EXISTS
    (SELECT NUMERO FROM PARI
     WHERE NUMERO = NUM_JOUEUR)
AND VIVANT = '1';
```

Chapitre 5

Les procédures

5.1 La fonction "nbrAmisVivants"

A Cette fonction prend en paramètre le numéro d'un joueur (ne marche pas sur les gardes) et renvoie le nombre d'amis vivants qu'une personne possède.

5.2 La procédure "emettrePari"

B Cette procèdure prend en paramètre le numéro de l'investisseur qui va réaliser le pari, le joueur sur lequel il va parier, le nom du jeu (il doit être en cours pour que le pari puisse être réalisé) et enfin la somme d'argent (qui est au maximum de 9999999999euro).
Une fois les tests validés sur les valeurs entrées en paramètre, on les insère dans la table "PARI"

5.3 Les triggers

5.3.1 Explication du trigger "TRIGG_MORT"

C Notre trigger se déclanche lors de l'insertion ou la mise a jour de la table "JOUE", il nous sert à la fois à empêcher l'utilisateur de faire jouer un joueur qui est mort mais aussi à changer le status de l'attribut "vivant" dans la table "PERSONNE" pour que lorsqu'un joueur décède dans un jeu, la valeur de son attribut qui était à 1 pour indiquer qu'il était vivant, passe à 0 pour indiquer qu'il est mort.

5.3.2 Explication du trigger "TRIGG_JOUEUR_GARDE"

D Notre trigger se déclenche lors de l'insertion ou la mise a jour de la table "PERSONNE" pour éviter a l'utilisateur de rentrer une personne qui est à la fois joueur et garde ou aucun des deux. En effet, dans notre base de données, les personnes doivent être soit des joueurs soit des gardes.

Chapitre 6

Les tests

Voici les différents tests que vous pouvez effectuer pour tester notre travail. Vous pourrez effectuer les tests de la fonction et de la procédure dans le fichier "test_Groupe3.sql"

6.1 Test de la procédure "emettrePari"

Plusieurs informations vous seront demandées en entrée :

— Un numéro d'un investisseur qui n'a pas encore parié (2,3,4,5).

— Un numéro de joueur vivant (2 par exemple)

— Le nom du jeu en cours ('EPREUVE A LA CORDE')

— Une somme

Pour tester si la procédure s'est correctement réalisée, exécuter la commande "SELECT * FROM PARI ;"

Le résultat (avec l'investisseur numéro 2 et une somme de 9999999999euro) :

```
SQL> SELECT * FROM PARI;

   NUM_INV NOM_JEU                        NUM_JOUEUR      SOMME
---------- ------------------------------ ---------- ----------
         1 RED LIGHT GREEN LIGHT                   1     100000
         2 RED LIGHT GREEN LIGHT                  16     150000
         3 RED LIGHT GREEN LIGHT                  15       5000
         4 RED LIGHT GREEN LIGHT                   5      18000
         5 RED LIGHT GREEN LIGHT                  18      50500
         1 BISCUIT                                11      30000
         2 BISCUIT                                 1     200000
         3 BISCUIT                                18      10500
         4 BISCUIT                                18      40500
         5 BISCUIT                                17     125000
         1 EPREUVE A LA CORDE                      2     120000
         2 EPREUVE A LA CORDE                      2 9999999999

12 lignes selectionnees.
```

6.2 Test de la fonction "nbr_amis"

Entrer le joueur numéro 1, la fonction devrait vous renvoyer que le joueur numéro 1 à trois amis vivants.
Résultat :

Pour tester nos triggers, voici quelques insertions que vous pouvez faire.

6.3 Test du trigger "TRIGG_MORT"

Pour tester ce Trigger, vous pouvez essayer ces insertions :
- INSERT INTO JOUE VALUES (000001,'BILLES','MORT') ;
Le Joueur 1 va passer de vivant à mort sur la table Personnage

- INSERT INTO JOUE VALUES (000011,'BILLES','MORT') ;
Le Joueur 11 est déjà mort donc exception

6.4 Test du trigger "TRIGG_JOUEUR_GARDE"

Pour tester ce Trigger, vous pouvez essayer ces insertions :
- INSERT INTO PERSONNE VALUES
(000031,'TEST','Personnage',33699874684,'111 rue Saint Germain','ROND',1,0000019) ;
Le personnage a des attributs de Joueur et Garde donc erreur :
"-20000 :'Un personnage ne peut pas être un Joueur et un Garde'"

-INSERT INTO PERSONNE VALUES
(000032,'TEST2','Personnage',33699874684,'111 rue Saint Germain',NULL,1,0000019) ;

-INSERT INTO PERSONNE VALUES
(000032,'TEST2','Personnage',33699874684,'111 rue Saint Germain','ROND',NULL,NULL) ;

-INSERT INTO PERSONNE VALUES
(000032,'TEST2','Personnage',33699874684,'111 rue Saint Germain',NULL,NULL,NULL) ;
Le personnage n'a aucun attribut de joueur/garde donc erreur :
"-20002 :'Un personnage doit être un Joueur ou un Garde'"

Annexes

Annexe A

Fonction nbrAmisVivants

```
CREATE OR REPLACE FUNCTION nbrAmisVivants(num_joueur PERSONNE.NUMERO%TYPE)
RETURN NUMERIC IS


    -- variables locales et exceptions

    joueur_mort_exception EXCEPTION;
    nbr_amis NUMERIC(3, 0);
    est_vivant NUMBER(1);

BEGIN

    DBMS_OUTPUT.ENABLE();

    -- pour verifier si le joueur existe
    -- renvoie une exception NO_DATA_FOUND si joueur inexistant
    SELECT vivant INTO est_vivant
    FROM PERSONNE WHERE PERSONNE.NUMERO = num_joueur;

--on verifie si il est vivant, sinon on renvoie une exception

    IF est_vivant = 0
        THEN RAISE joueur_mort_exception;
    END IF;

    SELECT COUNT(*) INTO nbr_amis
    FROM PERSONNE
    JOIN AMITIE ON PERSONNE.NUMERO = AMITIE.NUM1
    WHERE NUMERO = num_joueur
    AND DATE_FIN IS NULL
    AND AMITIE.NUM2 IN (SELECT NUMERO FROM PERSONNE WHERE VIVANT = 1);

    RETURN nbr_amis;

EXCEPTION

    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20010, 'ERREUR NBR_AMIS_VIVANTS: Le joueur ' || num_joueur || ' n''existe pas!');
    WHEN joueur_mort_exception THEN
        RAISE_APPLICATION_ERROR(-20011, 'ERREUR NBR_AMIS_VIVANTS: Le joueur ' || num_joueur || ' est mort, il n''a donc plus d''amis...');

END;
/
```

Annexe B

Procédure emettrePari

```
prompt ----------------------------------------;
prompt ---  Création PROCEDURE emettrePari  -----;
prompt ----------------------------------------;

CREATE OR REPLACE PROCEDURE emettrePari
        (num_investisseur_P IN INVESTISSEUR.NUM_INV%TYPE,
        num_joueur_P IN PERSONNE.NUMERO%TYPE,
        nom_jeu_P IN JEUX.NOM%TYPE,
        somme_P IN PARI.SOMME%TYPE) IS


        -- variables locales et exceptions

        investisseur_inexistant_exception EXCEPTION;
        joueur_inexistant_exception EXCEPTION;
        jeu_inexistant_exception EXCEPTION;

        joueur_mort_exception EXCEPTION;
        jeu_indisponible_exception EXCEPTION;

        somme_incorrect EXCEPTION;
        pari_deja_existant EXCEPTION;

        num_investisseur_liste INVESTISSEUR.NUM_INV%TYPE;
        num_investisseur_pari PARI.NUM_INV%TYPE;

        existe_joueur NUMERIC(1, 0);
        etat_joueur PERSONNE.VIVANT%TYPE;
        nom_jeu JEUX.NOM%TYPE;


BEGIN

        DBMS_OUTPUT.ENABLE();

-- TEST qu'on selectionne un investisseur qui existe ---------------------------------------

        SELECT COUNT(*) INTO num_investisseur_liste FROM INVESTISSEUR
        WHERE NUM_INV = num_investisseur_P;

        IF num_investisseur_liste = 0

                THEN RAISE investisseur_inexistant_exception;
        END IF;
```

```
-- TEST qu'on selectionne une personne qui existe ET qui n'est pas un garde ET qui vivante

        SELECT COUNT(*) INTO existe_joueur FROM PERSONNE
        WHERE NUMERO = num_joueur_P AND SYMBOLE IS NULL;

        IF existe_joueur = 0

                THEN RAISE joueur_inexistant_exception;
        END IF;


        SELECT VIVANT INTO etat_joueur FROM PERSONNE
        WHERE NUMERO = num_joueur_P AND SYMBOLE IS NULL;

        IF etat_joueur = 0

                THEN RAISE joueur_mort_exception;
        END IF;



-- TEST qu'on selectionne un jeu qui existe et qui est disponible ------------------------

        SELECT COUNT(*) INTO nom_jeu FROM JEUX
        WHERE NOM = nom_jeu_P;

        IF nom_jeu = 0

                THEN RAISE jeu_inexistant_exception;
        END IF;


        SELECT STATUS INTO nom_jeu FROM JEUX
        WHERE NOM = nom_jeu_P;

        IF nom_jeu = 'FINI' OR nom_jeu = 'A_VENIR'

                THEN RAISE jeu_indisponible_exception;
        END IF;
```

```
-- TEST qu'on a une somme correcte ------------------------------------------------------

        IF somme_P IS NULL OR somme_P <= 0

                THEN RAISE somme_incorrect;
        END IF;


-- TEST si l'investisseur selectionné a deja parié pour le jeu ----------------------------

        SELECT COUNT(*) INTO num_investisseur_pari FROM PARI
        WHERE NUM_INV = num_investisseur_P AND NOM_JEU = nom_jeu_P;


        IF num_investisseur_pari > 0

                THEN RAISE pari_deja_existant;
        END IF;


-- SI tout est bon, on insert les valeurs dans la table PARI


        INSERT INTO PARI VALUES(num_investisseur_P, nom_jeu_P, num_joueur_P, somme_P);

EXCEPTION


        WHEN investisseur_inexistant_exception
        THEN RAISE_APPLICATION_ERROR(-20013, 'ERREUR EMETTRE_PARI: L''investisseur specifie n''existe pas!');
        WHEN joueur_inexistant_exception
        THEN RAISE_APPLICATION_ERROR(-20014, 'ERREUR EMETTRE_PARI: Le joueur specifie n''existe pas!');
        WHEN jeu_inexistant_exception
        THEN RAISE_APPLICATION_ERROR(-20015, 'ERREUR EMETTRE_PARI: Le jeu specifie n''existe pas!');

        WHEN joueur_mort_exception
        THEN RAISE_APPLICATION_ERROR(-20016, 'ERREUR EMETTRE_PARI: Le joueur ' || num_joueur_P || 'est mort, impossible de parier sur lui...');
        WHEN jeu_indisponible_exception
        THEN RAISE_APPLICATION_ERROR(-20017, 'ERREUR EMETTRE_PARI: Le jeu n''est pas disponible! (fini ou a venir)');

        WHEN somme_incorrect
        THEN RAISE_APPLICATION_ERROR(-20018, 'ERREUR EMETTRE_PARI: La somme specifiee n''est pas valide!');
        WHEN pari_deja_existant
        THEN RAISE_APPLICATION_ERROR(-20019, 'ERREUR EMETTRE_PARI: L''investisseur specifie a deja effectue un pari pour ce jeu!');

END;
/
```

Annexe C

Trigger TRIGG_MORT

```
CREATE OR REPLACE TRIGGER TRIGG_MORT
BEFORE INSERT OR UPDATE ON JOUE
FOR EACH ROW
DECLARE
    MESSAGE EXCEPTION;
    numeroJ PERSONNE.NUMERO%type;
    estVivant PERSONNE.VIVANT%type;
BEGIN
    SELECT VIVANT INTO estVivant FROM PERSONNE WHERE NUMERO = :new.NUM_JOUEUR;
    IF estVivant = 0
        THEN RAISE MESSAGE;
    END IF;

    SELECT NUMERO INTO numeroJ FROM PERSONNE WHERE NUMERO = :new.NUM_JOUEUR;
        IF (:new.RESULTAT = 'MORT') THEN
            UPDATE PERSONNE
            SET VIVANT = 0
            WHERE PERSONNE.NUMERO = numeroJ;
        END IF;

EXCEPTION
    WHEN MESSAGE THEN RAISE_APPLICATION_ERROR(-20001,'Un personnage mort ne peut pas jouer');

END;
/
```

Annexe D

Trigger TRIGG_JOUEUR_GARDE

```
CREATE OR REPLACE TRIGGER TRIGG_JOUEUR_GARDE
BEFORE INSERT OR UPDATE ON PERSONNE
FOR EACH ROW
DECLARE
    MESSAGE1 EXCEPTION;
    MESSAGE2 EXCEPTION;

BEGIN
    IF :new.SYMBOLE IS NOT NULL AND (:new.VIVANT IS NOT NULL OR :new.IDFAM IS NOT NULL)
        THEN RAISE MESSAGE1;
    END IF;
    IF :new.SYMBOLE IS NULL AND (:new.VIVANT IS NULL OR :new.IDFAM IS NULL)
        THEN RAISE MESSAGE2;
    END IF;

EXCEPTION
    WHEN MESSAGE1 THEN RAISE_APPLICATION_ERROR(-20000,'Un personnage ne peut pas etre un Joueur et un Garde');
    WHEN MESSAGE2 THEN RAISE_APPLICATION_ERROR(-20002,'Un personnage doit etre un Joueur ou un Garde');

END;
/
```