Modélisation d'une base de données pour les caractéristiques et les interactions des vaisseaux dans un jeu vidéo

Groupe 3
Bernardon Vincent [22009116]
Duban Mathis [2210226]
Hurel Jérémy [21907809]

Pour les besoins d'un jeu vidéo spatial, nous avons besoin d'une base de données stockant toutes les informations relatives aux entités présentes dans notre jeu afin de les représenter. Cette base de données permet une gestion complète et structurée des éléments interactifs présents dans le jeu, assurant ainsi une expérience immersive et cohérente pour les utilisateurs.

Avant de définir la modélisation du projet voici quelques éléments à prendre en compte pour une meilleure compréhension du texte :

- on notera les initiales AUEC signifiant ALPHA UNITED EARTH CREDITS qui représente la monnaie du jeu
- on ne fera aucune distinction entre les personnages joueurs et les personnages non-joueurs

Pour mener à bien ce projet, nous allons modéliser son contexte de la manière suivante :

Chaque **vaisseau** est défini par un identifiant unique qui est utilisé pour l'identifier. Toutes les caractéristiques du vaisseau seront spécifiées: sa masse en kg, sa longueur en m, sa largeur en m, et sa hauteur en m ainsi que son nom.
Un prix d'achat sera associé à chaque vaisseau en jeu (qui a pour unité des auec), pour plus de réalisme ce prix pourra prendre une très grande valeur (on parle en million d'AUEC).
Chaque vaisseau possède à son bord une quantité définie d'objets.
La masse totale de la masse de tous les objets contenus dans un vaisseau doit être inférieure à la moitié de la masse du vaisseau à vide.

Les modèles d'objets sont définis par un nom,un id et d'un statut (légal ou illégal), un prix en AUEC et une masse en g.

Chaque vaisseau peut posséder un **équipage** composé de plusieurs **personnes**,
Cet équipage est défini par un nom, identifiant unique permettant de l'identifier, une date de création, et d'un chef d'équipe (qui est une personne).
On souhaite avoir la possibilité d'afficher dans un tableau toutes les informations des membres d'équipage d'un vaisseau en particulier. Mais également d'afficher toutes les informations relatives à tous les vaisseaux du jeu et leurs équipages à la fois.

Chaque personne est représentée par un nom, un prénom, un numéro unique qui est issu d'un registre de **propriétaire** (celui-ci est précisé plus loin), une date de naissance, et d'un potentiel poste. On souhaite déterminer l'âge de la personne via sa date de naissance.
On souhaite avoir un moyen de lister les identifiants libres de propriétaire afin de faciliter le dénombrement d'entités dans nos données.

Un vaisseau a un unique propriétaire (une **personne** ou bien une **entreprise**) symbolisé par un numéro d'identification.

On souhaite garder un **historique des propriétaires**, avec la date à laquelle ils ont acheté le vaisseau, celle à laquelle ils ont cessé d'en être propriétaire (dans le cas où il ne le sont plus).

Pour simuler les changements de propriétaire, on aura également une manière de transférer un vaisseau d'un propriétaire à un autre.

Une entreprise est définie par un numéro d'identification unique issue du registre des propriétaires, une date de création. Une entreprise est forcément dirigée par un chef d'entreprise qui est nécessairement une personne majeure.
On souhaitera garder l'historique des chefs que l'entreprise a pu avoir tout au long de sa vie.

On désire avoir un moyen d'afficher le nombre d'objets illégaux d'une entreprise en particulier.

L'entreprise a des employés qui sont des personnes, celles-ci occupent un poste dans l'entreprise et ont un salaire, une personne ne peut être employée qu'une fois dans une même entreprise.

On distingue 2 types d'entreprises:

- Les **entreprises de fabrication de vaisseau** qui proposent une gamme de vaisseaux à vendre. Les entreprises de fabrication de vaisseaux gardent un **historique de leur vente** qui comporte un identifiant du vaisseau vendu à un propriétaire, l'identifiant du client ainsi que la date de vente.
- Les entreprises de fabrication de vaisseau sont aussi caractérisées par une des spécialités suivantes : Combat,Transport, Exploration,Industrial,Support,Competition,Ground,Mult.

On souhaite aussi avoir la possibilité de récupérer l'écart type de l'âge des employés dans une entreprise sur demande.

Il existe aussi des **entreprises de fabrication d'objets** qui proposent une gamme d'objets et qui sont définis par une catégorie d'entreprise.

Afin de garantir une continuité dans notre base, on souhaiterait avoir un moyen de garantir qu'un identifiant de propriétaire ne soit utilisé qu'une fois que ce soit dans les entreprises, ou bien les personnes.
On souhaite également faire la même vérification pour assurer que l'identifiant d'une entreprise soit utilisé une seule fois pour les entreprises vaisseaux ou les entreprises objets.

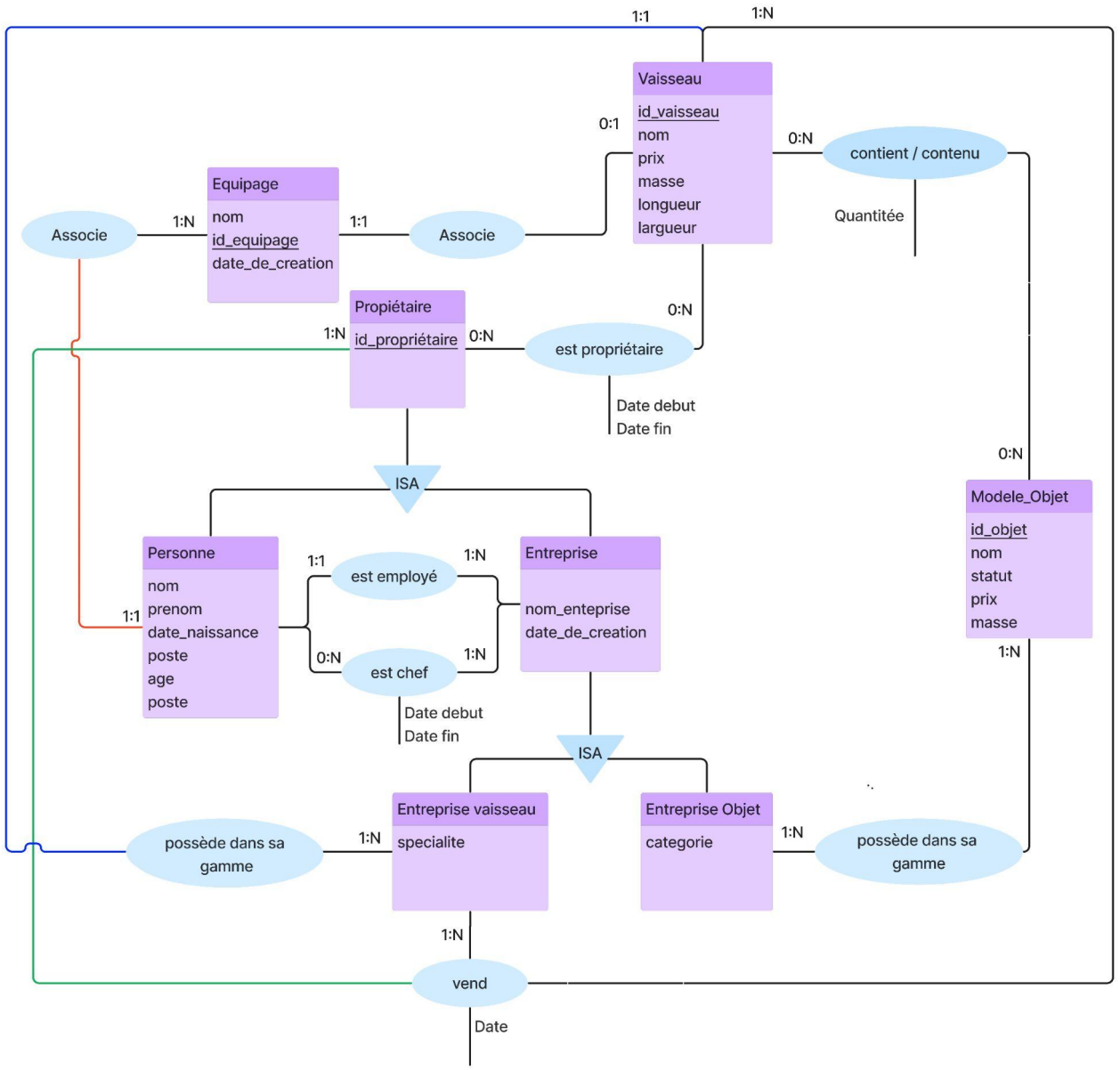
Dictionnaire de Données :

Code Rubrique/ Nom	Nom Rubrique/ Sémantique	Type	Règle d'intégrité/ Contraintes	Calculé (Oui/Non)
id_proprietaire	identifiant du propriétaire	INT	identifiant/unique	Non
id_equipage	identifiant de l'équipage	INT	identifiant/unique	Non
nom	nom de l'équipage	VARCHAR(50)	NOT NULL	Non
date_creation	date de la création de l'équipage	DATE	NOT NULL	Non
id_vaisseau	identifiant du vaisseau	INT	identifiant/unique	Non
nom_vaisseau	nom du vaisseau	VARCHAR(50)	NOT NULL	Non
prix	prix du vaisseau en auec	INT	NOT NULL	Non
masse	masse du vaisseau en g	INT	NOT NULL	Non
longueur	Longueur du vaisseau en m	INT	NOT NULL	Non
largeur	largeur du vaisseau en m	INT	NOT NULL	Non
nom	Nom de la personne	VARCHAR(50)	NOT NULL	Non
prenom	Prénom de la personne	VARCHAR(50)	NOT NULL	Non
date_naissance	Date de naissance de la personne	DATE	NOT NULL	Non
poste	poste de la personne	VARCHAR(50)	facultatif	Non
âge	Âge de la personne	INT	NOT NULL	Oui
nom_entreprise	nom de l'entreprise	VARCHAR(50)	NOT NULL	Non

date_creation	date de création de l'entreprise	DATE	NOT NULL	Non
Quantité	le nombre d'occurrence de l'objet dans l'inventaire	INT	NOT NULL	Non
Nom	nom de l'objet	VARCHAR(50)	NOT NULL	Non
id_objet	identifiant de l'objet	INT	NOT NULL	Non
statut	statut légal de l'objet	VARCHAR(7)	Appartient à {'légal', 'illégal'}	Non
masse	masse de l'objet en g	INT	NOT NULL	Non
prix	prix de l'objet	INT	NOT NULL	Non
date_debut	date de la prise de position du rôle de propriétaire	DATE	NOT NULL	Non
date_fin	date de la fin de la prise de position du rôle de propriétaire	DATE	facultatif	Non
date_debut	date de la prise de position du rôle de chef d'entreprise	DATE	NOT NULL	Non
date_fin	date de la fin de la prise de position du rôle de chef d'entreprise	DATE	facultatif	Non
date	date de vente du vaisseau par l'entreprise propriétaire	DATE	NOT NULL	Non
categorie	catégorie de l'entreprise fabrication objet	VARCHAR(50)	facultatif	Non

specialite	spécialité de l'entreprise fabrication vaisseau	VARCHAR(50)	Appartient à { 'Combat','Transport','Exploration','Industrial','Support','Competition','Ground','Multi'}	Non

Schéma entité-relation



Dans ce schéma, on distingue les entitées avec les rectangles violet et les associations avec les ellipses bleu claires.

Schéma relationnel

Proprietaire (id_proprietaire);

Equipage (id_equipage, nom , date_creation, #id_vaisseau);

Vaisseau(id_vaisseau, prix, masse, longueur, largeur, #id_fabriquant);

Personne(#id_personne, nom, prenom, poste, âge, date_naissance, #id_entreprise, # id_equipage);

Entreprise(#id_entreprise, nom_entreprise, date_creation);

Entreprise_Vaisseau(#id_entreprise, spécialité);

Entreprise_Objet(#id_entreprise, catégorie);

Modele_Objet(id_objet, nom, statut, prix, masse);

Gamme_Vente_Objet(#id_fabriquant, #id_objet);

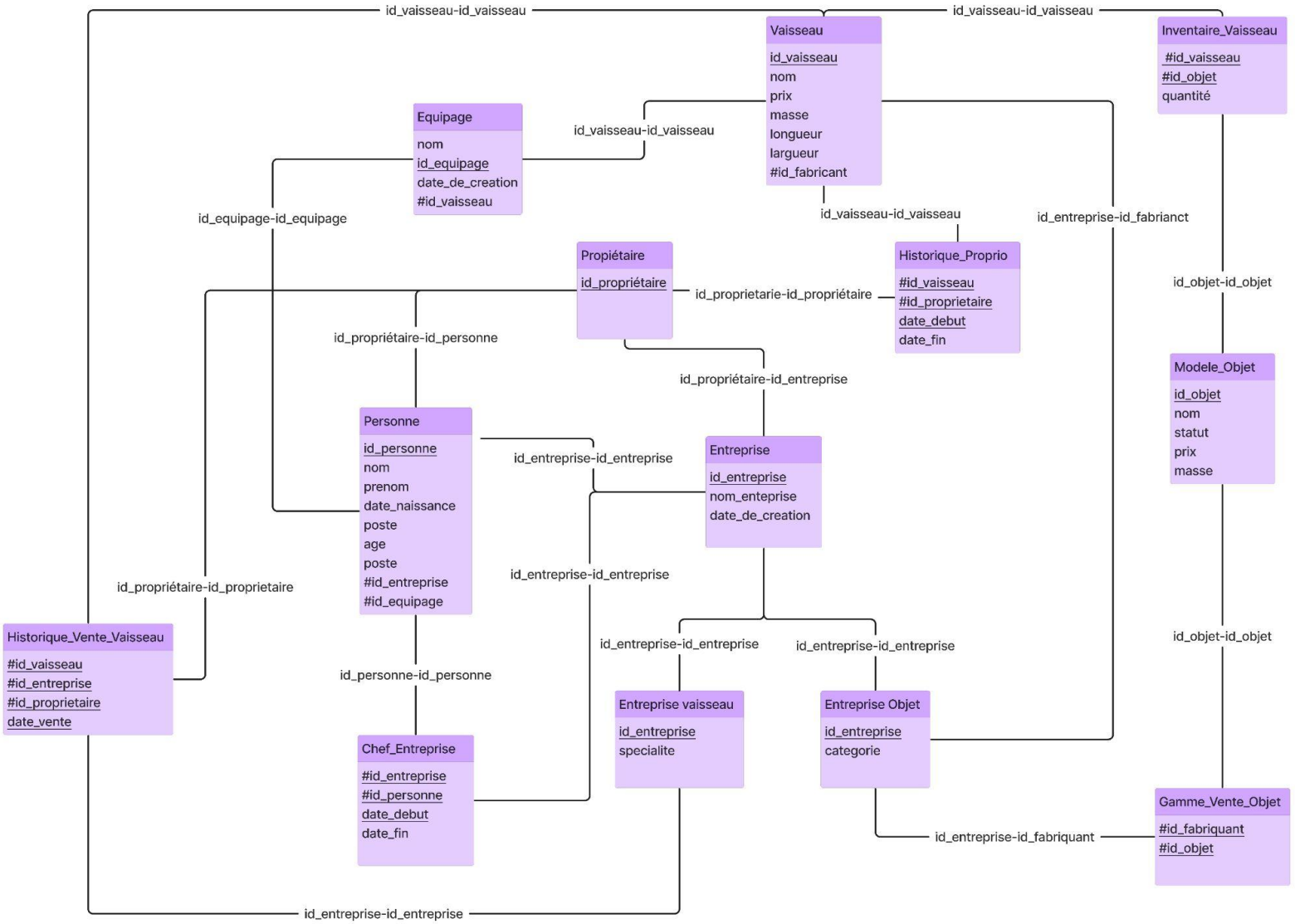
Inventaire_Vaisseau(#id_vaisseau, #id_objet, quantité);

Chef_Entreprise(#id_entreprise, #id_personne, date_debut, date_fin);

Historique_Proprio(#id_vaisseau, #id_proprietaire, date_debut, date_fin);

Historique_Vente_Vaisseau(#id_vaisseau, #id_entreprise, #id_proprietaire, date_vente);

Schéma physique :



Note : Nous avons réalisé le travail en utilisant POSTGRESQL. Voici la commande pour exécuter un fichier :

```
\i path/dossierProjet/NomFichier.sql
```

Procédure :

Notre première procédure nommée **afficher_informations_equipage** , permet d'afficher dans un tableau toutes les informations des personnes de l'équipage d'un vaisseau en particulier.
Cette procédure semble tout à fait triviale mais prend tout son sens lors de son utilisation dans notre deuxième procédure.

```
CREATE OR REPLACE PROCEDURE afficher_informations_equipage(IN vaisseau_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    equipage_id INT;
    personne_info RECORD;
BEGIN

    SELECT id_equipage INTO equipage_id
    FROM Equipage
    WHERE id_vaisseau = vaisseau_id;


    IF equipage_id IS NOT NULL THEN

        RAISE NOTICE '+----------------------------Equipage---------------------------------+';
        RAISE NOTICE '+------+-------------+----------+-----------+------+-----------------+';
        RAISE NOTICE '| ID   | Nom         | Prénom   | Poste     | Âge  | Date de naissance|';
        RAISE NOTICE '+------+-------------+----------+-----------+------+-----------------+';
        FOR personne_info IN
            SELECT id_personne, nom, prenom, poste, age, date_naissance
            FROM Personne
            WHERE id_equipage = equipage_id
        LOOP
            RAISE NOTICE '| % | % | % | % | % | % |',
                lpad(personne_info.id_personne::text, 4, ' '),
                lpad(personne_info.nom, 12, ' '),
                lpad(personne_info.prenom, 9, ' '),
                lpad(personne_info.poste, 10, ' '),
                lpad(personne_info.age::text, 4, ' '),
                personne_info.date_naissance;
        END LOOP;
        RAISE NOTICE '+------+-------------+----------+-----------+------+-----------------+';
        RAISE NOTICE ' ';
        RAISE NOTICE ' ';
        RAISE NOTICE ' ';

    ELSE
        RAISE NOTICE 'Ce vaisseau n''est associé à aucun équipage.';
        RAISE NOTICE ' ';
        RAISE NOTICE ' ';
        RAISE NOTICE ' ';

    END IF;
END;
$$;
```

Notre deuxième procédure nommée **afficher_informations_equipage_de_tous_les_vaisseaux** va afficher les informations de tous les vaisseaux y compris les membres d'équipage associés à chaque vaisseau grâce à l'appel de notre procédure réalisée auparavant. Cette procédure permet d'afficher de manière lisible et structurée dans un tableau toutes les informations pertinentes relatives aux vaisseaux disponibles.

```
CREATE OR REPLACE PROCEDURE afficher_informations_equipage_de_tous_les_vaisseaux()
LANGUAGE plpgsql
AS $$
DECLARE
    vaisseaux_info RECORD;
    equipage_bool BOOLEAN := FALSE;
BEGIN
    FOR vaisseaux_info IN
        SELECT id_vaisseau, nom, prix, masse, longueur, largeur, id_fabriquant
        FROM Vaisseau
    LOOP
        SELECT TRUE INTO equipage_bool
        FROM equipage
        WHERE id_vaisseau=vaisseaux_info.id_vaisseau;

        IF equipage_bool THEN
            RAISE NOTICE '+----------------------------Vaisseau--------------------------------+';
            RAISE NOTICE '+-------------------------%-------------------------------+',
                lpad(vaisseaux_info.nom, 10, ' ');
            RAISE NOTICE '+------+-------------+----------+-----------+---------+---------------+';
            RAISE NOTICE '| ID   | Prix        | Masse    | Longueur  | Largeur | ID_fabriquant |';
            RAISE NOTICE '+------+-------------+----------+-----------+---------+---------------+';
            RAISE NOTICE '| % | % | % | % | % | % |',
                lpad(vaisseaux_info.id_vaisseau::text, 4, ' '),
                lpad(vaisseaux_info.prix::text, 10, ' '),
                lpad(vaisseaux_info.masse::text, 7, ' '),
                lpad(vaisseaux_info.longueur::text, 8, ' '),
                lpad(vaisseaux_info.largeur::text, 6, ' '),
                lpad(vaisseaux_info.id_fabriquant::text, 12, ' ');
            CALL afficher_informations_equipage(vaisseaux_info.id_vaisseau);
        ELSE
            RAISE NOTICE '|Aucun équipage pour ce vaisseau|';
        END IF;
    END LOOP;
    RAISE NOTICE '+------+-------------+----------+-----------+--------+---------------+';
END;
$$;
```

Notre troisième procédure **TransfertVaisseau** est conçue pour effectuer le transfert d'un vaisseau d'un propriétaire source vers un autre propriétaire tout en mettant à jour l'historique de propriété du vaisseau. Cette procédure est pertinente dans un contexte de suivi de l'historique des propriétaires des vaisseaux, car elle permet de garder une trace claire des transferts de propriété tout en maintenant l'intégrité des données dans les tables d'historique.

```
CREATE OR REPLACE PROCEDURE TransfertVaisseau(IN vaisseau_idD INT,IN source_id INT,IN destination_id INT,IN date_transfer DATE)
LANGUAGE plpgsql
AS $$
DECLARE
    checkIfProprio RECORD;
    ok BOOLEAN := TRUE;
BEGIN

    FOR checkIfProprio IN SELECT * FROM Historique_Proprio WHERE id_vaisseau=vaisseau_idD AND id_proprietaire=source_id AND date_fin IS NULL
    LOOP
        IF ok THEN
            UPDATE Historique_Proprio SET date_fin = date_transfer WHERE id_vaisseau = vaisseau_idD AND date_fin IS NULL;
            INSERT
            INTO
            Historique_Proprio (id_vaisseau, id_proprietaire, date_debut)
            VALUES (vaisseau_idD, destination_id, date_transfer);
            ok := FALSE;
        ELSE
            RETURN;
        END IF;
    END LOOP;
    IF ok THEN
        RAISE EXCEPTION 'Pas de vaisseau correspondant transferable';
    END IF;
END;
$$;
```

Fonctions:

La fonction **ecart_type_age_entreprise** calcule l'écart type de l'âge des employés d'une entreprise quand celle-ci a bien des employés.
On récupère la moyenne d'âge des employés par une requête puis on parcours la liste des employés d'une entreprise pour calculer la variance $Var(X) = \frac{1}{nombre\ d'élément}\sum (x_i - moyenne)^2$, avec X la liste des employés de l'entreprise cible. L'écart type étant égal la racine de la variance c'est de là que vient le return sqrt(sum/count).

```
-- Retourne l'écart type de l'age des employé d'une entreprise.
CREATE OR REPLACE FUNCTION ecart_type_age_entreprise(IN entreprise_id INT)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    age_employer INT;
    count INT := 0;
    mean_age INT;
    sum INT := 0;
BEGIN
    SELECT AVG(age) INTO mean_age FROM Personne WHERE id_entreprise= entreprise_id GROUP BY id_entreprise;

    FOR age_employer IN
        SELECT age  FROM Personne WHERE id_entreprise= entreprise_id
    LOOP
        count := count+1;
        sum := sum + (mean_age - age_employer)*(mean_age - age_employer);

    END LOOP;

    IF count=0 THEN
        RAISE EXCEPTION 'pas d employé dans cette entreprise';

    ELSE
        RETURN SQRT(sum/count);
    END IF;
END;
$$;
```

La fonction **listFreeId** permet d'avoir la liste des id de propriétaire disponible entre l'id le plus petit et l'id le plus grand, cela permet d'avoir un ensemble d'id plus dense.

```
-- Retourne la liste de tout les id libre pour identifié un propriétaire entre l'id min et l'id max
CREATE OR REPLACE FUNCTION listFreeId()
RETURNS SETOF INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    current_id INT := 0;
    pres_id INT := 0;
    id_proprietaire INT;
BEGIN
    FOR id_proprietaire IN
        SELECT *
        FROM Proprietaire
        ORDER BY id_proprietaire
    LOOP
        current_id := id_proprietaire;

        IF NOT EXISTS (SELECT id_personne FROM Personne WHERE id_personne=current_id)
            AND
            NOT EXISTS (SELECT id_entreprise FROM Entreprise WHERE id_entreprise=current_id)
        THEN
            RETURN NEXT current_id;
        END IF;

        WHILE (current_id - pres_id) > 1 LOOP
            pres_id := pres_id +1;
            RETURN NEXT pres_id;
        END LOOP;
        pres_id := current_id;
    END LOOP;
    RETURN;
END;
$$;
```

La fonction **NombreObjetsIllegauxEntreprise** nous permet de savoir le nombre de modèle d'objets illégaux vendus par une entreprise.

```
-- Donne le nombre de model d'objet illégaux vendu par une entreprise
CREATE OR REPLACE FUNCTION NombreObjetsIllegauxEntreprise (entreprise_id INT)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    nb_objet INT := 0;
BEGIN
    SELECT COUNT(*) INTO nb_objet
    FROM Modele_Objet mo
    JOIN Gamme_Vente_Objet gvo ON mo.id_objet = gvo.id_objet
    WHERE mo.statut = 'ILLEGAL' AND gvo.id_fabriquant = entreprise_id;
    /*RAISE NOTICE '| % |',
        lpad(nb_objet::text, 4, ' ');*/
    RETURN nb_objet;
END;
$$;
```

Triggers:

Notre premier trigger **check_masse** va vérifier que la masse totale de la masse de tous les objets contenus dans un vaisseau doit être inférieure à la moitié de la masse du vaisseau à vide. Ce trigger est important afin de vérifier qu'à chaque insertion ou modification de la table inventaire_vaisseau, notre condition de masse soit respectée par la suite, si ce n'est pas le cas la quantité d'objet ne sera pas insérée.

```
CREATE OR REPLACE FUNCTION check_masse()
RETURNS TRIGGER AS $$
DECLARE
    masse_objet integer := 0;
    masse_inventaire integer := 0;
    masse_vaisseau integer := 0;
    total_masse_inventaire integer := 0;
    val_tuple RECORD;
BEGIN

    FOR val_tuple IN
        SELECT iv.quantite, md.masse
        FROM Inventaire_Vaisseau iv
        JOIN Modele_Objet md ON md.id_objet = iv.id_objet
        WHERE iv.id_vaisseau = NEW.id_vaisseau AND iv.id_objet <> NEW.id_objet
    LOOP
        total_masse_inventaire := total_masse_inventaire + (val_tuple.quantite * val_tuple.masse);
    END LOOP;


    SELECT masse INTO masse_objet
    FROM Modele_Objet
    WHERE id_objet = NEW.id_objet;

    SELECT masse INTO masse_vaisseau
    FROM Vaisseau
    WHERE id_vaisseau = NEW.id_vaisseau;


    masse_inventaire := total_masse_inventaire;
    --l'unité de la masse des objets étant en g et la masse des vaisseaux étant en Kg , nous devons multiplier par 1000
    --attention la masse des vaisseaux est en kg alors que les objets sont en g
    IF ((masse_inventaire + (NEW.quantite * masse_objet)) >= (masse_vaisseau / 2) * 1000) THEN
        RAISE NOTICE 'masse inventaire overcharge';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Créer le déclencheur
CREATE TRIGGER trigger_vaisseau_check_masse_objet
AFTER INSERT OR UPDATE ON Inventaire_Vaisseau
FOR EACH ROW
EXECUTE FUNCTION check_masse();
```

Notre deuxième trigger **update_age** est très important pour notre base de données, car il assure automatiquement la mise à jour de l'âge des individus en fonction de leur date de naissance. Cette fonctionnalité permet d'avoir des données actualisées en permanence, sans nécessiter de manipulations manuelles à chaque modification.

Note : Afin de garder les valeurs à jour, une cronTab (planificateur de tâches) pourra être éventuellement mise en place que ça soit par le biais d'une librairie spécialisée ou d'un processus externe afin d'automatiser le calcul de l'âge.

```sql
CREATE OR REPLACE FUNCTION update_age()
RETURNS TRIGGER AS $$
DECLARE
    age_result INTEGER;

    annee_current INTEGER;
    mois_current INTEGER;
    jour_current INTEGER;

BEGIN
    SELECT EXTRACT(YEAR FROM CURRENT_DATE),EXTRACT(MONTH FROM CURRENT_DATE),EXTRACT(DAY FROM CURRENT_DATE) INTO annee_current,mois_current,jour_current;

    age_result := annee_current- EXTRACT(YEAR FROM NEW.date_naissance);
    IF mois_current> EXTRACT(MONTH FROM NEW.date_naissance) THEN
        NEW.age := age_result -1;
        RETURN NEW;
    END IF;

    IF mois_current= EXTRACT(MONTH FROM NEW.date_naissance)  AND jour_current> EXTRACT(DAY FROM NEW.date_naissance) THEN
        NEW.age := age_result -1;
        RETURN NEW;
    END IF;

    RETURN NEW;

END;
$$ LANGUAGE plpgsql;

-- Créer le déclencheur
CREATE OR REPLACE TRIGGER trigger_update_age
BEFORE INSERT OR UPDATE ON Personne
FOR EACH ROW
EXECUTE FUNCTION update_age();
```

Le troisième trigger **check_chef_entreprise_majeur** va vérifier que lors d'une insertion ou la modification d'un tuple dans la table chef entreprise que la personne en question soit majeure car il est impossible d'avoir un chef d'entreprise mineur.

```sql
CREATE OR REPLACE FUNCTION check_chef_entreprise_majeur()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.date_debut IS NOT NULL THEN
        IF (SELECT age FROM Personne WHERE id_personne = NEW.id_personne) < 18 THEN
            RAISE EXCEPTION 'Le chef d''entreprise doit être une personne majeure';
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Création du trigger pour INSERT
CREATE TRIGGER trigger_check_chef_entreprise_majeur_insert
BEFORE INSERT OR UPDATE ON Chef_Entreprise
FOR EACH ROW
EXECUTE FUNCTION check_chef_entreprise_majeur();
```

Ce quatrième trigger **is_not_used_by_personne**, nous permet de garantir qu'un id_proprietaire n'est pas déjà utilisé par une personne afin de l'utiliser pour une entreprise. Dans le cas où l'identifiant de la requête n'est pas déjà présent dans la liste des identifiants des propriétaires il y sera alors ajouté.

```sql
CREATE OR REPLACE FUNCTION is_not_used_by_personne()
RETURNS TRIGGER AS $$

BEGIN
    IF NOT EXISTS (SELECT id_proprietaire FROM Proprietaire WHERE id_proprietaire=NEW.id_entreprise) THEN
        INSERT INTO Proprietaire (id_proprietaire) VALUES (NEW.id_entreprise);
        RETURN NEW;
    END IF;

    IF EXISTS (SELECT id_personne FROM Personne WHERE id_personne=NEW.id_entreprise) THEN
        RAISE EXCEPTION 'deja une personne';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_is_not_used_by_personne
BEFORE INSERT OR UPDATE ON Entreprise
FOR EACH ROW
EXECUTE FUNCTION is_not_used_by_personne();
```

Ce cinquième trigger **is_not_used_by_entreprise** nous permet de faire l'inverse du quatrième, c'est-à-dire. De vérifier que id_proprietaire n'est pas déjà utilisé dans la table entreprise afin de pouvoir créer une personne. Il ajoute aussi l'id dans Propriétaire en cas de besoin.

```
CREATE OR REPLACE FUNCTION is_not_used_by_entreprise()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (SELECT id_proprietaire FROM Proprietaire WHERE id_proprietaire=NEW.id_personne) THEN
        INSERT INTO Proprietaire (id_proprietaire) VALUES (NEW.id_personne);
        RETURN NEW;
    END IF;

    IF EXISTS (SELECT id_entreprise FROM Entreprise WHERE id_entreprise=NEW.id_personne) THEN
        RAISE NOTICE 'deja une enterprise';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_is_not_used_by_entreprise
BEFORE INSERT OR UPDATE ON Personne
FOR EACH ROW
EXECUTE FUNCTION is_not_used_by_Entreprise();
```

Ce sixième trigger **is_not_used_by_entreprise_objet**, nous permet de faire comme le cinquième. De vérifier que l'id_entrerpise n'est pas utilisé par une entreprise objet, afin de créer une entreprise vaisseau.

```
CREATE OR REPLACE FUNCTION is_not_used_by_entrepise_objet()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT id_entreprise FROM Entreprise_Objet WHERE id_entreprise=NEW.id_entreprise) THEN
        RAISE EXCEPTION 'deja une Entreprise Objet';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_is_not_used_by_entreprise_objet
BEFORE INSERT OR UPDATE ON Entreprise_Vaisseau
FOR EACH ROW
EXECUTE FUNCTION is_not_used_by_entrepise_objet();
```

Enfin ce dernier trigger, nous permet de faire l'inverse du cinquième. Autrement dit, il vérifie que l'id_entreprise n'est pas déjà utilisé par une entreprise vaisseau, si ce n'est pas le cas alors on autorise l'insertion d'une entreprise objet.

```
CREATE OR REPLACE FUNCTION is_not_used_by_entreprise_vaisseau()
RETURNS TRIGGER AS $$

BEGIN

    IF EXISTS (SELECT id_entreprise FROM Entreprise_Vaisseau WHERE id_entreprise=NEW.id_entreprise) THEN
        RAISE EXCEPTION 'deja une entreprise vaisseau';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_is_not_used_by_entreprise_vaisseau
BEFORE INSERT OR UPDATE ON Entreprise_Objet
FOR EACH ROW
EXECUTE FUNCTION is_not_used_by_entreprise_vaisseau();
```

Fichier de test :

Voici les différents tests que nous avons réalisé:

Triggers

Pour le trigger **check_chef_entreprise_majeur,** le but ici est de vérifier que lorsque nous insérons une personne mineure dans la table Chef_Entreprise, une exception soit levée en indiquant que la personne est mineure, ainsi la personne ne sera pas insérée.

Tout d'abord nous créons toutes les informations relative de la personne de test puis nous insérons notre nouvelle personne mineure:

```
INSERT INTO Personne (id_personne, nom, prenom, poste, date_naissance, id_entreprise, id_equipage)
VALUES (120,
        'Duban',
        'Mathis',
        'Ecolier',
        '2023-05-20',
        121,
        6600);
```

Ici, notre personne, à seulement quelques mois et par conséquent ne peut être chef d'entreprise, une erreur sera soulevée.

```
psql:C:/Users/mrvin/Desktop/test/projetBDD/final/test_Groupe3.sql:50: ERREUR:  Le chef d'entreprise doit Ãªtre une personne majeure
CONTEXTE : fonction PL/pgSQL check_chef_entreprise_majeur(), ligne 5 à RAISE
```

Pour le trigger **trigger_vaisseau_check_masse_objet ,** on va vérifier que lors d'une insertion sur Inventaire_Vaisseau, la condition :

$$MTotal \ + \ MNouvelObjet \leqslant MTotal \ / \ 2$$

Pour ce scénario de test , on créé le vaisseau de Test qui a une masse de 500kg .

```
INSERT INTO Vaisseau (id_vaisseau, nom, prix, masse, longueur, largeur, id_fabriquant)
VALUES (42642,
        'CORSAIR',
        1000,
        500,
        30,
        20,
        14); --500 de masse kg
```

Puis on insère 15000 objets de 25 g, ce qui nous donne 375000 g ou 375 kg. Le vaisseau ayant une masse de 500 kg, la masse des objets dépasse la moitié de la masse du vaisseau soit 250 kg. Une exception est donc levée.

```
INSERT INTO Inventaire_Vaisseau (id_vaisseau, id_objet, quantite)
VALUES(42642,
       13,
       15000); --25 de masse g, donc 25 * 15 = 375 g, donc * 1000 = 375 kg
```

```
psql:C:/Users/mrvin/Desktop/test/projetBDD/final/test_Groupe3.sql:82: ERREUR:  masse inventaire overcharge
CONTEXTE : fonction PL/pgSQL check_masse(), ligne 32 à RAISE
INSERT 0 1
```

Pour le trigger **trigger_update_age,** on va juste vérifier que l'attribut calculé âge se calcule correctement quand on insère sa date de naissance. Pour cela on va créer une personne et vérifier que son âge soit bien calculé.

```
INSERT INTO Personne(id_personne,nom,prenom,date_naissance) VALUES  (131, 'Smith', 'Howard','1993-05-20');
SELECT id_personne, age
FROM Personne
WHERE id_personne=131;
```

```
INSERT 0 1
 id_personne | age
-------------+-----
         131 |  29
(1 ligne)
```

On va également vérifier que lors d'une modification de tuple, la valeur soit bien modifiée, on va alors modifier l'âge de cette personne avec un UPDATE et vérifier que l'âge soit bien calculé :

```
INSERT INTO Personne(id_personne,nom,prenom,age,date_naissance) VALUES  (104, 'Smith', 'Ethan',1120,'1993-05-20');
SELECT id_personne, age
FROM Personne
WHERE id_personne=104;
UPDATE Personne SET date_naissance='1990-01-01' WHERE id_personne=104;
SELECT id_personne, age
FROM Personne
WHERE id_personne=104;
```

```
INSERT 0 1
 id_personne | age
-------------+-----
         104 |  29
(1 ligne)


UPDATE 1
 id_personne | age
-------------+-----
         104 |  32
(1 ligne)
```

Pour le trigger **is_not_used_by_personne** on va prouver que l'insertion est réalisée, c'est-à-dire que l'id_propriétaire n'est pas déjà utilisée par une personne.

```
INSERT INTO Entreprise (id_entreprise, nom_entreprise, data_creation)
VALUES (101,
        'Aegis Dynamics',
        '2023-01-01');
```

```
INSERT 0 1
```

Maintenant on va montrer que si l'id_propriétaire est déjà utilisé par une personne, alors le trigger soulève l'erreur.

```
INSERT INTO Personne (id_personne, nom, prenom, poste, date_naissance) VALUES
    (111, 'Smith', 'John', 'Engineer', '1993-05-20');

INSERT INTO Entreprise (id_entreprise, nom_entreprise, data_creation) VALUES
    (111, 'Aegis Dynamics', '2023-01-01');
```

```
INSERT 0 1
psql:C:/Users/mrvin/Desktop/test/projetBDD/final/test_Groupe3.sql:137: ERREUR:  deja une personne
CONTEXTE : fonction PL/pgSQL is_not_used_by_personne(), ligne 10 à RAISE
```

Afin de tester le trigger **is_not_used_by_entreprise**, on commencera pas prouver que l'insertion fonctionne si l'id_propriétaire n'est pas déjà utilisé par une entreprise.

```
INSERT INTO Personne (id_personne, nom, prenom, poste, date_naissance, id_equipage)
VALUES (110,
        'Smith',
        'John',
        'Engineer',
        '1993-05-20',
        1);
```

Maintenant on va tester que le trigger se déclenche si l'id_propriétaire est déjà utilisé par une entreprise.

```sql
INSERT INTO Entreprise (id_entreprise, nom_entreprise, data_creation)
VALUES (150,
        'Aegis Dynamics',
        '2023-01-01');


INSERT INTO Personne (id_personne, nom, prenom, poste, date_naissance)
VALUES (150,
        'Smith',
        'John',
        'Engineer',
        '1993-05-20');
```

```
INSERT 0 1
psql:C:/Users/mrvin/Desktop/test/projetBDD/final/test_Groupe3.sql:130: NOTICE:  deja une enterprise
```

Pour tester **is_not_used_by_entreprise_vaisseau** on regarde uniquement le cas ou il soulève l'exception. Puisque l'id_entreprise est déjà utilisé par une entreprise vaisseau.

```sql
INSERT INTO Entreprise_Vaisseau (id_entreprise,specialite)
VALUES (150,'Combat');

INSERT INTO Entreprise_Objet (id_entreprise)
VALUES (150);
```

```
INSERT 0 1
psql:C:/Users/mrvin/Desktop/test/projetBDD/final/test_Groupe3.sql:146: ERREUR:  deja une entreprise vaisseau
CONTEXTE : fonction PL/pgSQL is_not_used_by_entreprise_vaisseau(), ligne 6 à RAISE
```

Afin de tester **is_not_used_by_entreprise_objet**, nous allons uniquement faire le cas où le trigger soulève l'erreur, c'est-à-dire quand l'id_entreprise est déjà renseigné dans une entreprise objet déjà existante.

```sql
INSERT INTO Entreprise (id_entreprise, nom_entreprise, data_creation) VALUES
    (151, 'La Farfouille', '2023-01-01');
INSERT INTO Entreprise_Objet (id_entreprise)
VALUES (151);
INSERT INTO Entreprise_Vaisseau (id_entreprise,specialite)
VALUES (151,'Combat');
```

```
psql:C:/Users/mrvin/Desktop/test/projetBDD/final/test_Groupe3.sql:155: ERREUR:  deja une Entreprise Objet
CONTEXTE : fonction PL/pgSQL is_not_used_by_entrepise_objet(), ligne 4 à RAISE
```

Procédures:

Pour tester les différentes procédures, nous les avons simplement appelé dans le fichier de test et vérifier que leurs résultats étaient en concordance avec nos données. Voici comment nous les avons appelé :

Pour **afficher_information_equipage,** nous vérifions que les informations des équipages sont bien affichées dans notre tableau.

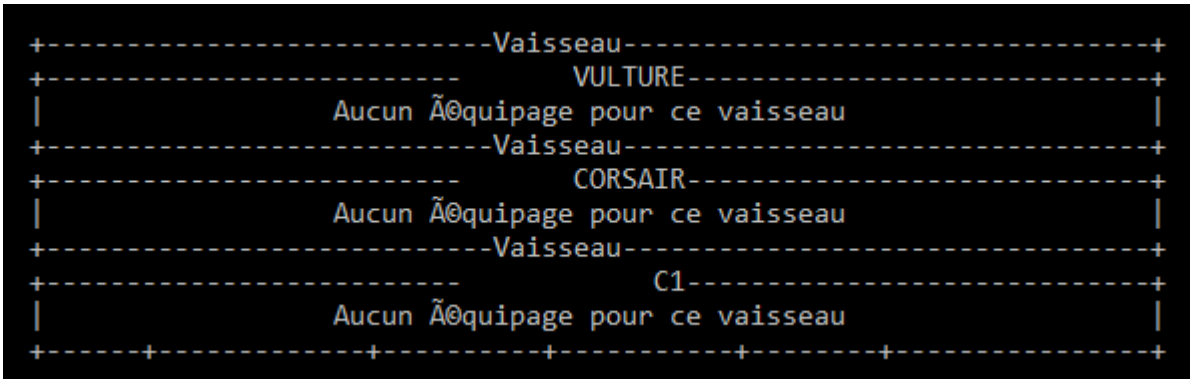
```
CALL afficher_informations_equipage(1);
CALL afficher_informations_equipage(100);
```

```
+-----------------------------Equipage-----------------------------------+
+------+-------------+----------+-----------+------+------------------+
| ID   | Nom         | PrÃ©nom   | Poste     | Ã'ge  | Date de naissance|
+------+-------------+----------+-----------+------+------------------+
|    1 |       Smith |     John |   Engineer |   29 | 1993-05-20 |
|  110 |       Smith |     John |   Engineer |   29 | 1993-05-20 |
+------+-------------+----------+-----------+------+------------------+



Ce vaisseau n'est associÃ© Ã  aucun Ã©quipage.
```

Puis nous appelons **afficher_informations_equipage_de_tous_les_vaisseaux** pour afficher toutes les informations des vaisseaux avec leurs équipages de manière structurée dans un tableau.

```
call afficher_informations_equipage_de_tous_les_vaisseaux();
```

```
+-----------------------------Vaisseau-----------------------------------+
+--------------------------       Pioneer-------------------------------+
+------+-------------+----------+-----------+---------+----------------+
| ID   | Prix        | Masse    | Longueur  | Largeur | ID_fabriquant |
+------+-------------+----------+-----------+---------+----------------+
|   10 |       3500 |    1400 |       75 |     65 |           20 |
+-----------------------------Equipage-----------------------------------+
+------+-------------+----------+-----------+------+------------------+
| ID   | Nom         | PrÃ©nom   | Poste     | Ã'ge  | Date de naissance|
+------+-------------+----------+-----------+------+------------------+
|   10 |       Taylor |    Olivia |   Engineer |   28 | 1994-11-28 |
+------+-------------+----------+-----------+------+------------------+
```



Ce n'est qu'une partie du résultat de la procédure qui renvoie les informations des vassaux et leur équipage quand il en ont un.

Pour la procédure **transfert_vaisseau,** on va vérifier que notre transfert de propriétaire pour notre vaisseau est bien fonctionnel. Pour cela on va créer les entitées nécessaire pour le transfert, effectuer le transfert puis vérifier avec un SELECT que le propriétaire et la date de changement de propriétaire sont bien changés.

```sql
INSERT INTO Proprietaire (id_proprietaire) VALUES (75),(76);
INSERT INTO Entreprise (id_entreprise, nom_entreprise, data_creation) VALUES (75, 'torp cher','2023-04-12'),(76, 'merci','2023-07-04');
INSERT INTO Entreprise_Vaisseau (id_entreprise, specialite) VALUES (75,'Combat'),(76,'Industrial');
INSERT INTO Vaisseau (id_vaisseau, nom, prix, masse, longueur, largeur, id_fabriquant) VALUES (42942, 'M2', 1000, 500, 30, 20, 75);
INSERT INTO Historique_Proprio (id_vaisseau, id_proprietaire, date_debut, date_fin) VALUES (42942, 75, '2023-01-01', NULL);

select *
from Historique_Proprio
where id_vaisseau = 42942 AND id_proprietaire=75;

call TransfertVaisseau(42942,75,76,'2023-12-19');

select *
from Historique_Proprio
where id_vaisseau = 42942 AND (id_proprietaire=76 OR id_proprietaire=75);
```

```
INSERT 0 2
INSERT 0 2
INSERT 0 2
INSERT 0 1
INSERT 0 1
 id_vaisseau | id_proprietaire | date_debut | date_fin
-------------+-----------------+------------+----------
       42942 |              75 | 2023-01-01 |
(1 ligne)


CALL
 id_vaisseau | id_proprietaire | date_debut |  date_fin
-------------+-----------------+------------+------------
       42942 |              75 | 2023-01-01 | 2023-12-19
       42942 |              76 | 2023-12-19 |
(2 lignes)
```

Fonction :

Pour la fonction **ecart_type_age_entreprise,** on appelle la fonction et on vérifie que le résultat retourné est bien correct.

```
SELECT ecart_type_age_entreprise(11);
```

```
 ecart_type_age_entreprise
---------------------------
                         5
(1 ligne)
```

Pour la fonction **listFreeId,** on va regarder que la liste des id libres retournés (entre id min et id max) sont bien correct :

```
SELECT listFreeId();
```

```
listfreeid
-----------
         31
         32
         33
         34
         35
         36
         37
         38
         39
         40
         51
         52
         53
         54
         55
         56
         57
         58
         59
```

Pour la fonction **NombreOjetsIllegauxEntreprise**, on va simplement l'appeler et vérifier que les résultats retournés sont bien corrects.

```
--test fonction NombreObjetsIllegauxEntreprise, entreprise avec des objets illégaux
select NombreObjetsIllegauxEntreprise(22);

--test fonction NombreObjetsIllegauxEntreprise, entreprise sans objets illégaux
select NombreObjetsIllegauxEntreprise(21);
```

```
 nombreobjetsillegauxentreprise
--------------------------------
                              5
(1 ligne)


 nombreobjetsillegauxentreprise
--------------------------------
                              0
(1 ligne)
```

Requêtes :

Une fois que la base de données, nos fonctions et nos triggers sont implémentés, nous pouvons réaliser certaines requêtes afin de récupérer diverses informations.

Nous avons à notre disposition beaucoup de personnes qui travaillent dans différentes entreprises, on peut donc se demander :

Combien compte t-on d'employés pour chaque entreprise?

Cette requête est très pertinente car elle permet de dénombrer toutes les entreprises ainsi que leur nombre d'employés. Cette requête utilisera le mot clé GROUP BY afin de regrouper chaque entreprise ainsi que le mot clé COUNT(*) pour compter le nombre d'employés associé à chaque groupe.

On répond à cette question de cette manière :

```
SELECT id_entreprise,Count(*) as Nombre_Employes
FROM Personne
GROUP BY id_entreprise;
```

Dans notre base de données, nous avons recensé bon nombre d'équipage travaillant dans différents vaisseaux, chaque équipage étant relié à un seul vaisseau, on peut alors se demander :

Quels sont les vaisseaux n'ayant pas d'équipage ?

Cette requête est très pertinente car elle permet de récupérer les informations des vaisseaux n'ayant pas d'équipage. Pour cela, on va devoir réaliser une sous-requête qui va récupérer les id des vaisseaux ayant un équipage, puis on va chercher tous les id de vaisseaux qui ne sont pas présents dans cet ensemble avec NOT IN. Il suffit alors d'afficher toutes les informations des vaisseaux récupérés avec SELECT * sur les id présent dans notre ensemble trouvé auparavant.
On répond à cette question de cette manière:

```
SELECT * FROM Vaisseau
WHERE id_vaisseau NOT IN
    (
    SELECT id_vaisseau FROM Equipage GROUP BY id_vaisseau
    );
```

Chaque vaisseau à son propre prix (en auec) de vente, ayant cette information on peut alors se demander comment se comporte le marché de vente de vaisseau dans notre jeu, y a t'il des vaisseaux "supérieurs" définis par leur prix. On peut alors se demander :

Quels sont les vaisseaux ayant un prix supérieur à la moyenne des prix des autres vaisseaux?

Pour cette requête on va devoir utiliser l'opérateur AVG qui va récupérer la moyenne de la valeur de l'attribut passé en paramètre. En utilisant AVG sur l'attribut prix de la table vaisseau, on va récupérer la moyenne des prix de tous les autres vaisseaux. Pour finir il suffit de regarder les prix des vaisseaux qui sont supérieurs à cette valeur.

On répond à cette question de cette manière :

```
SELECT *
FROM Vaisseau
WHERE prix > (
    SELECT AVG(prix)
    FROM Vaisseau
);
```

Nous avons intégré dans notre base de données une multitude d'objets divers et variés allant d'un kit de soin à un laser mortel, tous ces objets ne sont pas légaux car ils sont plus ou moins dangereux et les vaisseaux possédant de tels objets dans leur inventaires sont considérés comme dangereux. Les pires criminels spatiaux n'ont généralement que des objets illégaux dans leur inventaire. On peut alors se poser comme question :

Quels sont les vaisseaux qui ne possèdent que des objets illégaux dans leur inventaire?

Pour traiter cette problématique, nous utiliserons l'approche d'une opération de division. Notre objectif est de déterminer quels vaisseaux ont exclusivement des objets catégorisés comme "illégaux" dans leur inventaire. Cette démarche implique la création d'une requête spécifique visant à isoler les vaisseaux répondant à cette condition précise. En subdivisant les données selon le critère "objets légaux uniquement", nous identifions les vaisseaux qui satisfont cette particularité au sein de notre ensemble de données, il ne nous reste plus qu'à récupérer les vaisseaux

qui ne correspondent pas à cet ensemble pour avoir ceux ayant uniquement des objets illégaux dans leur inventaire.

Voici comment on répond à notre question:

```sql
SELECT id_vaisseau, nom
FROM Vaisseau
WHERE id_vaisseau NOT IN (
    SELECT id_vaisseau
    FROM Inventaire_Vaisseau
    GROUP BY id_vaisseau
    HAVING COUNT(*) > 0
    AND COUNT(*) = (
        SELECT COUNT(*)
        FROM Inventaire_Vaisseau iv
        JOIN Modele_Objet mo ON iv.id_objet = mo.id_objet
        WHERE mo.statut = 'LEGAL'
        AND iv.id_vaisseau = Vaisseau.id_vaisseau
    )
);
```

Nous avons de nombreuses entreprises référencées dans notre base de données. Parmi les entreprises proposant des objets à la vente, nous souhaitons filtrer celles ayant une activité soutenue. Pour répondre à ce besoin on peut se poser la question suivante :

Quelles sont les entreprises dont le nombre de gammes d'objets proposés à la vente dépasse la moyenne du nombre de gammes d'objets proposés à la vente par les autres entreprises ?

La logique de cette requête réside dans son utilisation du principe des sous-requêtes pour comparer les performances des fabricants d’objets. Dans un premier temps, la première sous-requête interne calcule le nombre total de gamme de vente proposé par chaque entreprise. Puis, elle compare ce total à la moyenne générale des gammes proposées et on retourne les entreprises associées à nos résultats..

On répond à cette question de cette manière :

```sql
SELECT id_entreprise, nom_entreprise
FROM Entreprise E1
WHERE (
    SELECT COUNT(*)
    FROM Gamme_Vente_Objet GVO
    WHERE GVO.id_fabriquant = E1.id_entreprise
) > (
    SELECT AVG(nombre_objets)
    FROM (
        SELECT COUNT(*) AS nombre_objets
        FROM Gamme_Vente_Objet
        GROUP BY id_fabriquant
    )
);
```