TP Noté - Langage SQL et SGBD Oracle

Gaël Roustan (Argonaultes)

2024

Abstract

Ce document contient les instructions du devoir noté.

Human Bot

Contexte

Votre société de conseil idBOT est appelée pour moderniser l'approche métier d'une entreprise familiale 'HUMAN BOT' spécialisée dans la création de robots humanoïdes.

Toute la société était gérée sur papier. Votre objectif est de remplacer cette gestion papier par une gestion entièrement numérique avec notamment une base de données Oracle accompagnée des procédures et fonctions adéquates pour la communication avec les futures applications nécessitant un échange d'information avec cette nouvelle base de données.

Consignes et évaluations

Il faudra donc respecter à la lettre les noms des fonctions, procédures et vues que l'on vous demande de créer.

Ces vues, procédures et fonctions sont nécessaires dans la mesure où les applications n'auront jamais connaissance de la structure des tables, qui pourra évoluer au fil du temps sans que cela ait un impact sur les applications tierces.

Livrable

A l'issue de votre session de travail, vous devez fournir plusieurs fichiers SQL à travers un repo GIT dont vous fournirez l'adresse via le formulaire Google. Ce repo GIT doit contenir les 3 fichiers suivants avec les noms exacts :

- `my_views.sql` : fichier SQL contenant uniquement la création des vues
- `my_funcs_procs.sql` : fichier SQL contenant les fonctions et procédures
- `my_triggers.sql` : fichier SQL contenant les triggers

Détails du fonctionnement (pour la création des tables)

Dans le cadre cette évaluation, un des collaborateurs de votre société a déjà réfléchi à la structure interne et vous fournit donc le script SQL qui permet de connaître les tables qui seront créées. Ce script est téléchargeable depuis le repo Github langage-sql.

```
DROP TABLE WORKERS_FACTORY_1;
DROP TABLE AUDIT_ROBOT;
DROP TABLE SUPPLIERS_BRING_TO_FACTORY_1;
DROP TABLE SUPPLIERS_BRING_TO_FACTORY_2;
DROP TABLE ROBOTS_HAS_SPARE_PARTS;
DROP TABLE ROBOTS_FROM_FACTORY;
DROP TABLE WORKERS_FACTORY_2 ;
DROP TABLE FACTORIES;
DROP TABLE SPARE_PARTS;
DROP TABLE SUPPLIERS;
DROP TABLE ROBOTS;



CREATE TABLE FACTORIES (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
     ↪  KEY,
    main_location VARCHAR2(255)
);

CREATE TABLE WORKERS_FACTORY_1 (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
     ↪  KEY,
    first_name VARCHAR2(100),
    last_name VARCHAR2(100),
    age NUMBER,
    first_day DATE,
    last_day DATE
);

CREATE TABLE WORKERS_FACTORY_2 (
    worker_id NUMBER GENERATED BY DEFAULT AS IDENTITY
 ↪  PRIMARY KEY,
    first_name VARCHAR2(100),
    last_name VARCHAR2(100),
    start_date DATE,
    end_date DATE
);

CREATE TABLE SUPPLIERS (
    supplier_id NUMBER GENERATED BY DEFAULT AS IDENTITY
 ↪  PRIMARY KEY,
    name VARCHAR2(100)
);
```

```
CREATE TABLE SPARE_PARTS (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
     ↪  KEY,
    color VARCHAR2(10) CHECK(color in ('red', 'gray',
 ↪  'black', 'blue', 'silver')),
    name VARCHAR2(100)
);

CREATE TABLE SUPPLIERS_BRING_TO_FACTORY_1 (
    supplier_id NUMBER REFERENCES suppliers(supplier_id),
    spare_part_id NUMBER REFERENCES spare_parts(id),
    delivery_date DATE,
    quantity NUMBER CHECK(quantity > 0)
);

CREATE TABLE SUPPLIERS_BRING_TO_FACTORY_2 (
    supplier_id NUMBER REFERENCES suppliers(supplier_id)
 ↪  NOT NULL,
    spare_part_id NUMBER REFERENCES spare_parts(id) NOT
 ↪  NULL,
    delivery_date DATE,
    quantity NUMBER CHECK(quantity > 0),
    recipient_worker_id NUMBER REFERENCES
 ↪  workers_factory_2(worker_id) NOT NULL
);

CREATE TABLE ROBOTS (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
     ↪  KEY,
    model VARCHAR2(50)
);

CREATE TABLE ROBOTS_HAS_SPARE_PARTS (
    robot_id NUMBER REFERENCES robots(id),
    spare_part_id NUMBER REFERENCES spare_parts(id)
);

CREATE TABLE ROBOTS_FROM_FACTORY (
    robot_id NUMBER REFERENCES robots(id),
    factory_id NUMBER REFERENCES factories(id)
);

CREATE TABLE AUDIT_ROBOT (
    robot_id NUMBER REFERENCES robots(id),
    created_at DATE
```

```
);
```

Vues (6 points)

La définition des vues demandées doit être enregistrée dans le fichier `my_views.sql` à la racine de votre repo GIT.

1. Donner la syntaxe SQL pour créer la vue `ALL_WORKERS` qui affiche en lecture seule pour l'ensemble des employés les propriétés suivantes :

- `lastname`
- `firstname`
- `age`
- `start_date`

Trier le résultat afin que le salarié arrivé le plus récemment soit affiché en premier. Si des valeurs sont manquantes, convservez les tout de même dans le résultat. N'affichez que les travailleurs toujours présents dans l'usine.

2. Créer une seconde vue `ALL_WORKERS_ELAPSED` qui donne le nombre de jours écoulés depuis l'arrivée de chaque employé au moment de la requête en vous basant sur la vue `ALL_WORKERS`.
3. Créer la vue `BEST_SUPPLIERS` qui affiche les fournisseurs ayant livré plus de 1000 pièces. Trier le résultat afin que le fournisseur ayant livré le plus grand nombre de pièces apparaisse en premier.
4. Créer une vue `ROBOTS_FACTORIES` qui permet de connaître l'usine ayant assemblé chaque robot enregistré.

Fonctions (5 points)

La définition des fonctions demandées doit être enregistrée dans le fichier `my_funcs_procs.sql` à la racine de votre repo GIT.

1. Créer la fonction `GET_NB_WORKERS(FACTOR NUMBER) RETURN NUMBER` qui renvoie le nombre de travailleurs dans une usine fournie en paramètre
2. Créer la fonction `GET_NB_BIG_ROBOTS() RETURN NUMBER` qui compte le nombre de robots assemblés avec plus de 3 pièces
3. Créer la fonction `GET_BEST_SUPPLIER() RETURN VARCHAR2(100)` qui renvoie le nom du meilleur fournisseur basée sur la vue `BEST_SUPPLIERS`

Procédures (5 points)

La définition des procédures demandées doit être enregistrée dans le fichier `my_funcs_procs.sql` à la racine de votre repo GIT.

1. Créer la procédure `SEED_DATA_WORKERS(NB_WORKERS NUMBER, FACTORY_ID NUMBER)` qui crée autant de workers que demandé. Les valeurs pour le nom seront `'worker_f_' || id` et `'worker_l_' || id` et la date de démarrage sera prise au hasard entre le 01/01/2065 et 01/01/2070. Pour générer une date au hasard, vous pouvez utiliser la requête SQL suivante

```
SELECT TO_DATE(TRUNC(DBMS_RANDOM.VALUE(TO_CHAR(DATE
↪  '2065-01-01','J'), TO_CHAR(DATE '2070-01-01','J'))),
↪  'J') FROM DUAL
```

1. Créer la procédure `ADD_NEW_ROBOT(MODEL_NAME VARCHAR2(50))` qui crée un nouveau modèle depuis la vue `ROBOTS_FACTORIES`.

Triggers (4 points)

La définition des triggers demandés doit être enregistrée dans le fichier `my_triggers.sql` à la racine de votre repo GIT.

1. En cas de tentative d'insertion d'un worker dans la vue `ALL_WORKERS_ELAPSED`, intercepter la demande d'insertion pour effectuer l'insertion dans la bonne table. Dans les autres cas, `UPDATING` et `DELETING`, lever une erreur.
2. A chaque nouveau robot créé, enregistrer la date d'ajout du robot dans la table `AUDIT_ROBOT`.
3. Si le nombre d'usines dans la table `FACTORIES` n'est pas égal au nombre de table respectant le format `WORKERS_FACTORY_<N>` alors empêcher toute modification de données via la vue `ROBOTS_FACTORIES`.