Système d'information, Architecture et Base de données

Gaël Roustan (Argonaultes)

2024-05-23

Abstract

Ce document contient les étapes pour mener à bien le TP afin de valier les acquis relatifs aux types d'architecture ainsi que la manipulation des bases de données.

Human Bot

Contexte

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

Avant votre intervention, 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 accompagnée des procédures et fonctions adéquates pour la communication avec les futures applications nécessitant un échange d'information avec cette nouvelle base de données.

Vous intervenez alors que la société vient de faire l'acquisition d'une seconde usine qui avait son propre mode de fonctionnement.

Modalités

Groupes

Ce TP peut être réalisé en groupe de 4 ou 5.

Durée

Pour réaliser ce TP, vous avez 7 heures.

Utilisation de GIT

L'historique de vos commits GIT seront utilisés pour l'évaluation.

Description détaillée

La gestion d'une usine s'appuie sur différents aspects :

- la gestion des ressources humaines
- la gestion du stock
- la gestion des fournisseurs

Chacune des usines a sa propre vision et l'objectif n'est pas de forcer l'usine rachetée à adopter la vision de la société mère mais plutôt de transformer et fusionner les modes de gestion des 2 usines.

Usine de la société mère

Pour chaque employé, la société mère conserve le nom, le prénom, l'age, le 1er jour d'arrivée dans l'usine en tant que travailleur et le jour du départ.

Pour chaque fournisseur, nous conservons le nom ainsi que les différentes livraisons de pièces détachées. Pour chaque livraison, nous avons la date de livraison ainsi que la quantité de pièces livrées.

Pour terminer, l'usine a pour objectif de fabriquer des robots. Chaque robot est composé d'un certain nombre de pièces. Pour suivre les variations de stock, nous notons la quantité de robots qui est produite chaque jour.

Usine rachetée

Pour chaque employé, nous connaissons le prénom, le nom, le jour de démarrage dans l'usine ainsi que sa présence ou non dans les effectifs.

Pour chaque fournisseur, nous conservons le nom ainsi que les différentes livraisons de pièces détachées. Pour chaque livraison, nous avons la date de livraison, la quantité de pièces livrées ainsi que l'ouvrier qui a réceptionné la commande. Cette information permet de s'assurer qu'un responsable s'est occupé de vérifier le contenu de la livraison.

Pour terminer, l'usine a pour objectif de fabriquer des robots. Chaque robot est composé d'un certain nombre de pièces. Pour suivre les variations de stock, nous notons la quantité de pièces détachées utilisées chaque jour.

Stratégie

La société HUMAN BOT ne souhaite pas dépenser un ou plusieurs loyers pour des solutions de gestion de stock, suivi de fournisseur et de suivi RH. HUMAN BOT souhaite plutôt investir dans une solution fabriquée sur mesure sur laquelle elle aura la main en s'appuyant si nécessaire sur des briques Open Source.

Vous devez fournir un dictionnaire de données permettant de recouvrir le fonctionnement des 2 usines.

Ce dictionnaire vous servira à réaliser le MCD permettant de modéliser le fonctionnement des 2 usines.

Il est important de bien séparer le fonctionnement des 2 usines tout au long du processus car nous ne souhaitons modifier le fonctionnement

des 2 usines. En revanche, il faut prévoir un moyen de regrouper les visions différentes de fonctionnement des 2 usines pour faciliter le suivi au niveau de la société mère.

Solution technique

Dans votre solution technique, nous attendons bien entendu une base de données, si possible PostgreSQL.

Pour l'affichage des pages web, vous devez utiliser Flask, le framework d'application Web Python.

Pour l'interface avec la base de données, vous pouvez utiliser un ORM tel que SQLAlchemy.

Interface utilisateur RH

Créer une page web qui affiche les salariés de toutes les usines.

Interface utilisateur Fournisseur

Créer une page web qui affiche les livraisons des 10 derniers jours.

API Fournisseur

Pour faciliter la gestion du stock, vous devez fournir un point d'accès qui permet d'afficher le nombre de pièces restantes par usine.

Ce sera à la charge du fournisseur de consulter ce point d'accès pour programmer des livraisons permettant de maintenir les stocks à un niveau suffisant.

Interface utilisateur Stock

Créer une page web qui affiche les pièces restantes de chaque usine.

Sécurité

Ne vous préoccupez pas pour le moment de l'aspect sécurité. Nous l'implémenterons plus tard. Cela signifie qu'il n'est pas nécessaire d'implémenter une page de connexion.

Livrable

Consignes et évaluations

Il faut 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.

Référentiel GIT

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

- `hb_overview.png` : schéma logique du futur SI
- `hb_dico.md` : dictionnaire de données au format markdown
- `hb_mcd.png` : schéma MCD au format image
- `hb_mld.md` : MLD au format markdown
- `hb_schema.sql` : fichier SQL contenant uniquement la création des tables
- `hb_views.sql` : fichier SQL contenant uniquement la création des vues
- `hb_funcs_procs.sql` : fichier SQL contenant les fonctions et procédures
- `hb_triggers.sql` : fichier SQL contenant les triggers
- `ws_rh` : dossier du projet Python Flask pour le service RH
- `ws_stock` : dossier du projet Python Flask pour le service Stock
- `api_fournisseur` : dossier du projet Python Flask API pour le service
- `ws_fournisseur` : dossier du projet Python Flask pour le service Fournisseur

Schéma logique

Le schéma logique suit le formalisme :

- rectangle pour représenter un service métier interne à Human Bot
- cercle pour représenter un service externe à Human Bot
- une flèche pour montrer l'initiateur de la demande d'accès à la donnée

Sur ce schéma, indiquer le pattern client-serveur à différentes échelles. Lister dans un fichier séparé les couples client-serveur.

Dictionnaire de données

Sous la forme d'un simple tableau, lister les signifiants et les signifiés.

- code
- description
- nature (nombre, chaine de caractères)
- contraintes
- règle de calcul

Attention aux manifestations de données suivantes :

- synonyme : plusieurs noms qui font référence à la même donnée, il ne faut conserver qu'un seul.
- polysème : un mot fait référence à différentes données, il faut préciser le mot en question. Par exemple, nom peut être le nom de la société et le nom d'un ouvrier. Il faut alors créer 2 signifiants différents : nom_ouvrier et nom_société

Vues

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, conservez 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

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

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

Triggers

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.