PROJET TAD GLPI

(Gestion du parc informatique de CY Tech)

Enseignant : BOUHAMOUM REDOUANE
Matière : Traitement Bases de données avancées
Année Universitaire : 2023-2024

Adam BOUHRARA - Clément CASSIET - Jordan GAUTIER - Joan LEGRAND
ING 2 Génie Système d'Information, Formation en Apprentissage

I. Introduction

1) Introduction Generale

Ce rapport présente notre projet visant à améliorer la performance du parc informatique de CY Tech en repensant la base de données GLPI, réalisée dans le cadre de notre cours de Bases de Données Avancées pour l'année 2023-2024.

Notre objectif est de répondre aux défis de gestion des tickets, à propos du matériels informatiques, des utilisateurs et des problèmes rencontrés, tout cela en mettant en place une nouvelle structure de base de données plus efficace. Nous décrirons notre démarche, de la conception à l'implémentation, en mettant l'accent sur les gains de performance obtenus.

2) Architecture et Exécution

Pour la gestion de versions et le partage de code, nous avons opté pour GitHub. Vous pouvez consulter l'intégralité de notre code en suivant ce lien.

Pour la structure des fichiers, nous avons opté pour des dossiers correspondant aux droits nécessaires à l'exécution des ces fichiers. Voici les associés des dossiers :
- CERGY : utilisateur GLPI_CERGY
- PAU : utilisateur GLPI_PAU
- BOTH : droits sur les 2 schémas ou supérieur (system)

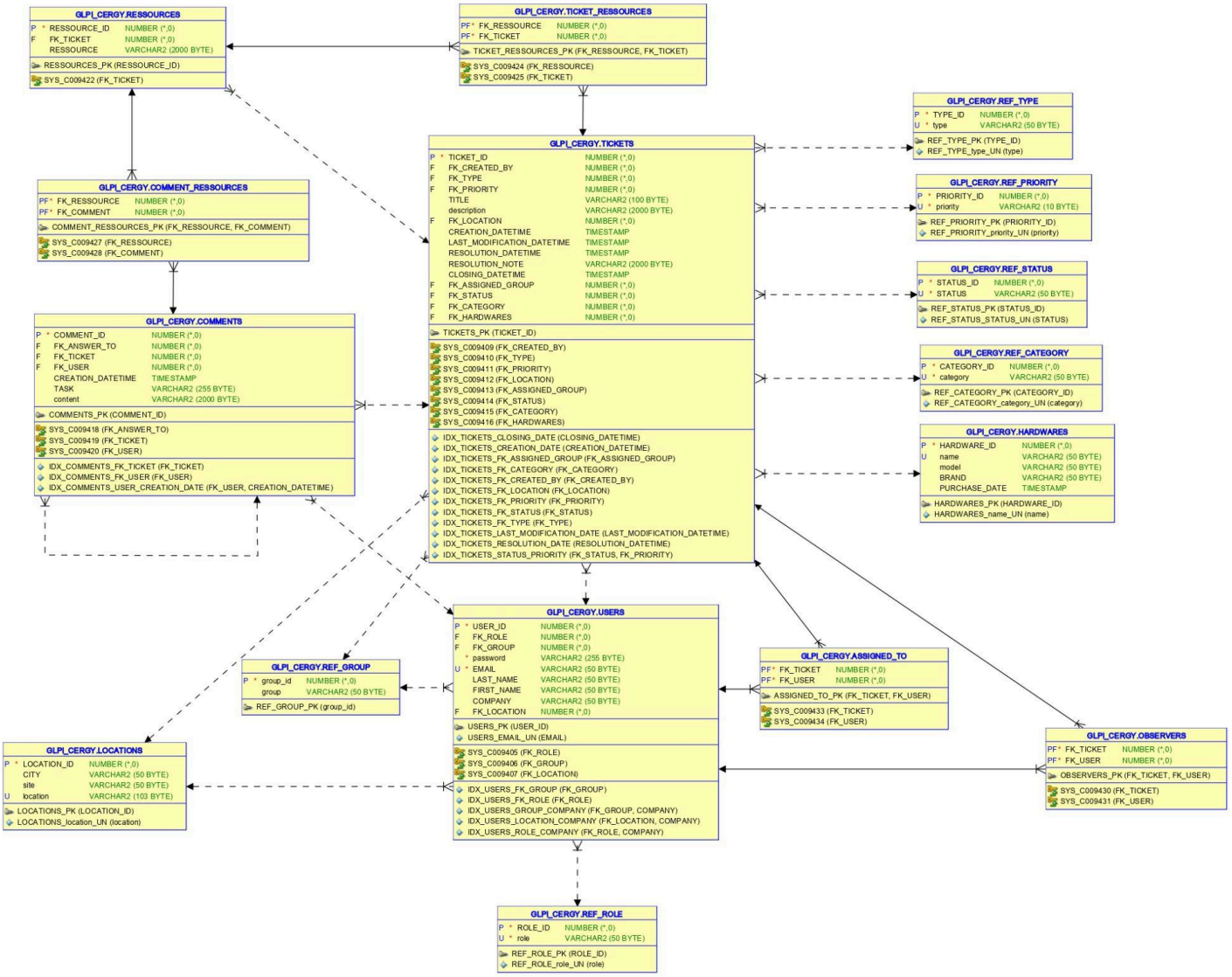
Pour automatiser la création et la suppression de nos bases de données, nous avons élaboré des scripts dédiés.

Le script de création de bases de données prend en charge la création de l'ensemble des composants nécessaires, incluant les tables, les déclencheurs (triggers), les index, les procédures stockées, les vues, ainsi que la gestion des rôles et des utilisateurs. Chaque fichier SQL est exécuté dans un ordre précis avec la connexion à l'utilisateur adéquate, tout en intégrant la génération de logs pour

faciliter le processus de débogage. Concernant la suppression de bases de données, un script PowerShell dédié a également été développé. Ce script assure une suppression complète de la base de données spécifiée, y compris tous ses composants associés.

II. Modélisation logique

1) Schéma de conception Générale



2) Création des tables

- Les tables

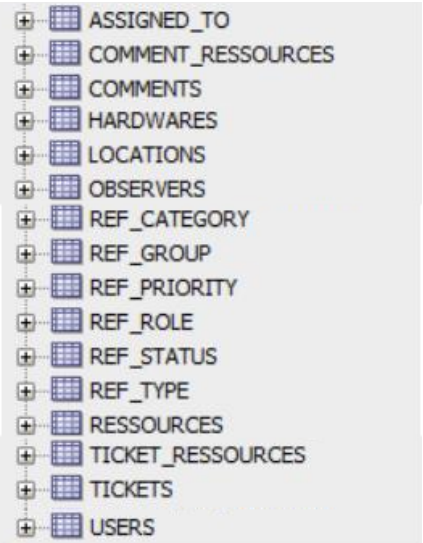
La conception de l'architecture de notre base de données tourne autour des tickets et des utilisateurs.

Dans un premier temps, pour la gestion des tickets, différentes tables de référence ont été mises en place pour faciliter la gestion des données en évitant la redondance et en assurant la cohérence des informations. Ces tables définissent une liste prédéfinie de valeurs pour les priorités, les statuts des tickets etc. Les tables REF_STATUS, REF_CATEGORY, REF_TYPE, REF_ROLE, REF_GROUP et REF_PRIORITY stockent les valeurs pour le statut, la catégorie, le type, le rôle et la priorité des tickets. Ces tables sont liées aux tables principales via des clés étrangères, permettant des vérifications sur ces champs. Cela garantit également la flexibilité et la facilité d'ajout de nouvelles valeurs, ce qui simplifie la gestion et assure l'évolutivité du système.

Par la suite, nous avons également pris en compte d'autres éléments partagés entre les tickets, tels que le matériel associé, les commentaires. Ces éléments permettent de fournir des informations supplémentaires et de mieux comprendre les problèmes soulevés par les utilisateurs. Une table de ressource a été créée à part, puisqu'elle est commune aux tickets et aux commentaires. En effet, les ressources étant des images ou des fichiers en pièce jointe, il est possible d'en ajouter à la création du ticket ou dans chaque commentaire.

En ce qui concerne les utilisateurs, chaque utilisateur peut se voir attribuer un rôle spécifique et être membre d'un groupe donné. Ensuite, des éléments sont communs entre les tickets et les utilisateurs, tels que la personne assignée au ticket, les observateurs et la localisation, ont été pris en compte dans la conception de la base de données. De plus, un lien direct entre les tickets et les utilisateurs est établi via l'information du créateur du ticket, permettant ainsi de suivre facilement l'origine de chaque ticket.

Les tables de liaison comme OBSERVERS et ASSIGNED_TO permettent de gérer les relations n-n entre les utilisateurs et les tickets, notamment pour les observateurs et les responsables des tickets. Ces tables de liaison contribuent à maintenir l'intégrité des données et à simplifier les opérations de gestion des tickets et des utilisateurs.



Ensemble des tables créées

De plus, dans l'objectif de réduire l'utilisation de l'espace disque et d'améliorer les performances de notre base de données, nous avons mis en place deux clusters sur chaque tablespace, regroupant les tables qui utilisent des informations des tickets et des utilisateurs. Des choix ont été faits pour les tables qui utilisent les deux, puisqu'il est impossible d'ajouter deux clusters pour une table en Oracle SQL. Les tables de références n'ont quant à elles pas de clusters puisqu'elles n'ont pas de colonne contenant l'ID du ticket ou de l'utilisateur.

- Le cluster user

Ce cluster user est créé autour de la colonne user_id. Les tables qui en bénéficient sont notamment : users, observers et assigned_to.
Cette configuration optimise les requêtes qui nécessitent des jointures sur les users, en regroupant physiquement les données liées autour de la colonne user_id.

- Le cluster ticket

Ce cluster ticket est créé autour de la colonne ticket_id. Les tables qui en bénéficient sont : ticket, ressources et comment.
Cette configuration optimise les requêtes qui nécessitent des jointures sur les tickets, en regroupant physiquement les données liées autour de la colonne ticket_id.

Et, nous avons créé des index sur ces clusters car c'est nécessaire pour pouvoir les utiliser.

```
CREATE CLUSTER GLPI_CERGY.user_cluster (user_id INT) TABLESPACE GLPI_CERGY_TABSPACE;
CREATE CLUSTER GLPI_CERGY.ticket_cluster (ticket_id INT) TABLESPACE GLPI_CERGY_TABSPACE;

CREATE INDEX GLPI_CERGY.idx_user_cluster ON CLUSTER GLPI_CERGY.user_cluster;
CREATE INDEX GLPI_CERGY.idx_ticket_cluster ON CLUSTER GLPI_CERGY.ticket_cluster;
```

3) Choix des contraintes et des types de données

Dans la conception de notre base de données, nous avons accordé une attention particulière au choix des contraintes et des types de données afin d'assurer la fiabilité, la cohérence et les performances de notre système.

Les contraintes sont des règles appliquées aux données pour garantir leur intégrité. Nous avons utilisé divers types de contraintes, notamment :

- Unique : Cette contrainte assure que la valeur d'une colonne est unique dans la table, évitant ainsi les doublons et assurant l'unicité des données.
- Null et Not Null : Les contraintes Null et Not Null permettent de spécifier si une colonne peut accepter des valeurs nulles ou non. Cela assure que les données essentielles ne sont pas omises.
- Valeurs par défaut : Nous avons également défini des valeurs par défaut pour certaines colonnes, comme un SYSTIMESTAMP par défaut pour la date de création d'un ticket ou une clé faisant référence à une table de référence, telle que le statut "À faire" par défaut pour un nouveau ticket.

De plus, nous avons appliqué des contraintes spécifiques pour garantir la sécurité et la validité des données. Par exemple, pour les mots de passe, nous avons imposé des contraintes telles que la longueur minimale de 14 caractères, la présence de chiffres, de lettres majuscules et minuscules, ainsi que la présence d'au moins un caractère spécial. De même, pour les adresses e-mail, nous avons appliqué une contrainte de structure pour garantir qu'elles suivent le format correct, avec une adresse e-mail valide contenant un "@".

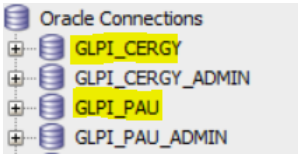
En ce qui concerne les types de données, nous avons soigneusement sélectionné ceux qui conviennent le mieux à chaque colonne pour optimiser les performances de la base de données. Nous avons utilisé des types de données appropriés pour les clés primaires, les clés étrangères et d'autres champs afin de garantir l'efficacité des opérations de recherche, de tri et de jointure.

```
-- C'est la table principale qui contient les details de chaque ticket de support.
CREATE TABLE GLPI_PAU.TICKETS (
    ticket_id INT PRIMARY KEY,  -- Identifiant unique du ticket
    fk_created_by INT NOT NULL,  -- Cle etrangere vers l'utilisateur qui a cree le ticket
    fk_type INT NOT NULL,  -- Cle etrangere vers le type de ticket
    fk_priority INT DEFAULT 3,  -- Cle etrangere vers la priorite du ticket
    title VARCHAR2(100) NOT NULL,  -- Titre du ticket
    "description" VARCHAR2(2000) NOT NULL,  -- Description du ticket
    fk_location INT NOT NULL,  -- Cle etrangere vers l'emplacement du ticket
    creation_datetime TIMESTAMP DEFAULT SYSTIMESTAMP,  -- Date et heure de creation du ticket
    last_modification_datetime TIMESTAMP DEFAULT SYSTIMESTAMP,  -- Date et heure de derniere modification du ticket
    resolution_datetime TIMESTAMP NULL,  -- Date et heure de resolution du ticket
    resolution_note VARCHAR2(2000) NULL,  -- Note de resolution du ticket
    closing_datetime TIMESTAMP NULL,  -- Date et heure de clôture du ticket
    fk_assigned_group INT NULL,  -- Clé étrangère vers le groupe assigné au ticket
    fk_status INT DEFAULT 1,  -- Clé étrangère vers le statut du ticket
    fk_category INT NOT NULL,  -- Clé étrangère vers la catégorie du ticket
    fk_hardwares INT NULL,  -- Clé étrangère vers le matériel associé au ticket (peut être NULL)
    FOREIGN KEY (fk_created_by) REFERENCES GLPI_PAU.USERS(user_id),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_type) REFERENCES GLPI_PAU.REF_TYPE(type_id),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_priority) REFERENCES GLPI_PAU.REF_PRIORITY(priority_id),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_location) REFERENCES GLPI_PAU.LOCATIONS(location_id),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_assigned_group) REFERENCES GLPI_PAU.REF_GROUP("group_id"),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_status) REFERENCES GLPI_PAU.REF_STATUS(status_id),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_category) REFERENCES GLPI_PAU.REF_CATEGORY(category_id),  -- Contrainte de clé étrangère
    FOREIGN KEY (fk_hardwares) REFERENCES GLPI_PAU.HARDWARES(hardware_id)  -- Contrainte de clé étrangère
)
CLUSTER GLPI_PAU.ticket_cluster(ticket_id);
```

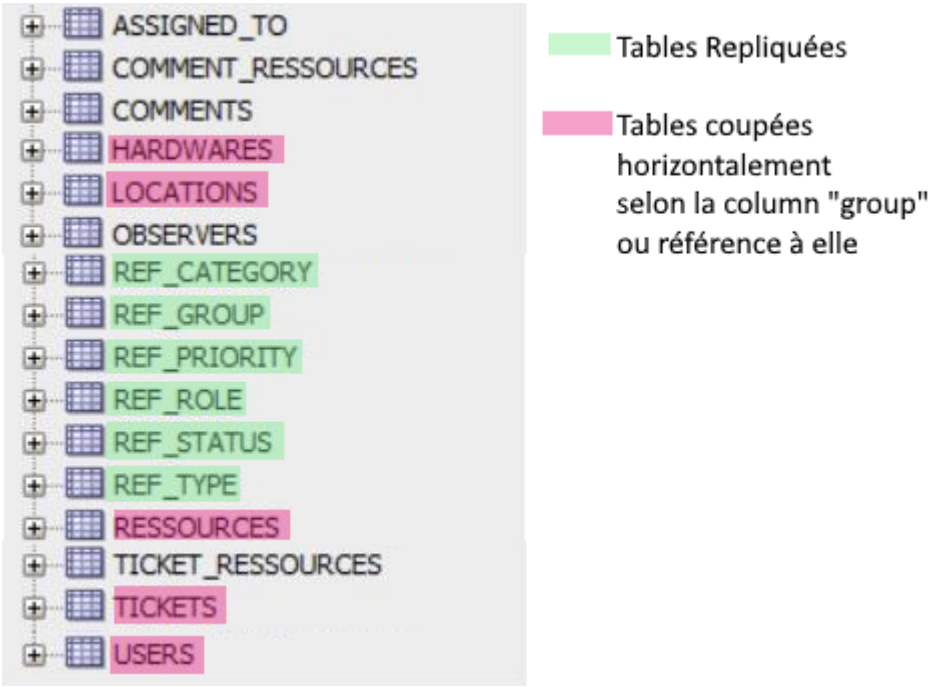
III. Modélisation physique

1) Fragmentation et liens entre les bases de données fractionnées

Cette modélisation et ces tables sont séparées en deux Databases.
Nous avons créé deux usagers qui servent de databases avec leur schéma : GLPI_CERGY et GLPI_PAU.

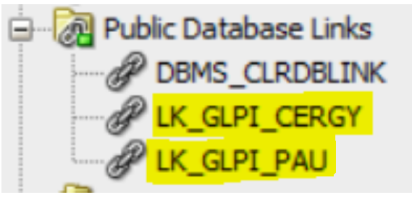


Certes, toutes les tables sont présentes de chaque côté, cependant nous avons effectué une coupure horizontale sur le champ ‘city’ (Cergy / Pau). Evidemment, certaines des tables ont nécessité d’être répliquées, les tables de ref notamment.
Ainsi, les tables comme Ticket, User, Hardwares et Ressources sont coupés horizontalement sur ce champ ‘city’ (de la table LOCATIONS) auxquelles elles font références. Et par extension toutes les tables qui font référence à ces tables (‘Comment’, etc.)

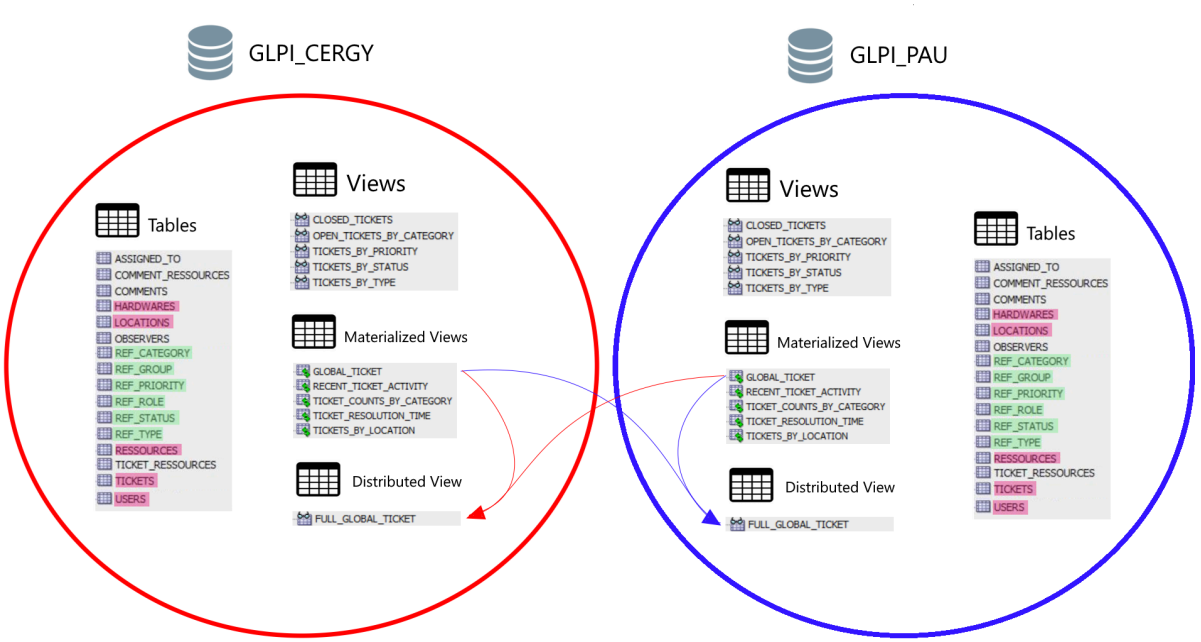


La colonne “group” à donc été retirée des tables Hardware et Ressources, mais elle a été gardé dans User, Ticket (et évidemment LOCATIONS) car elle donne une information intéressante, notamment dans la vue FULL_GLOBAL_TICKET qui rassemble les information des deux bases et qui est présente dans les deux bases.

Les deux databases peuvent communiquer entre elles grâce aux databases link public que nous avons créés, qui sont notamment utilisés dans la vue FULL_GLOBAL_TICKET. Chacune des deux pointe vers le schéma de l’autre base.



- Schéma récapitulatif de la fragmentation



2) Définition des index pour améliorer les performances des requêtes

Dans le cadre de l'optimisation des performances de notre base de données Oracle, nous avons mis en place des index visant à accélérer les requêtes courantes et à améliorer l'efficacité globale du système.
Nous avons utilisé différents types d'index pour répondre à divers besoins de requêtes.

- Les index B-tree

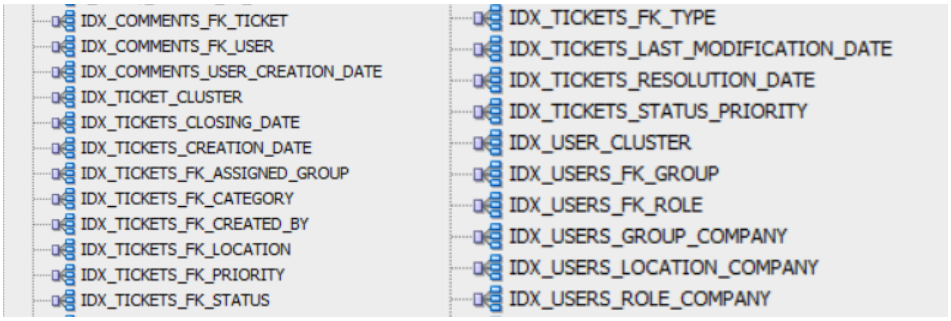
Ce sont des index à forte sélectivité, ils sont donc utilisés sur les colonnes présentant des valeurs presque uniques. Cette utilisation est idéale pour les opérations d'égalité. Par exemple, l'index sur la colonne fk_created_by dans la table TICKETS permet d'accélérer les requêtes filtrant par utilisateur, améliorant ainsi l'efficacité des recherches de tickets attribués à des utilisateurs spécifiques.

Les index peuvent s'appliquer sur plusieurs colonnes (index composite). Ils ont été créés pour les requêtes fréquentes. Par exemple, l'index composite sur les colonnes fk_status et fk_priority dans la table TICKETS permet d'accélérer les requêtes qui filtrent simultanément par état et priorité. Elles permettent donc des opérations selon les besoins spécifiques de notre application.

- Les index Bitmap

Ce sont des index à faible sélectivité, ils sont donc utilisés sur les colonnes ayant une faible cardinalité. Cette utilisation est idéale pour les opérations de filtrage. Par exemple, l'index bitmap sur la colonne fk_role dans la table USERS permet d'optimiser les requêtes qui filtrent ou agrègent les utilisateurs par rôle, comme la recherche de tous les utilisateurs appartenant à un certain rôle, comme "Support", "IT" ou "RH" par exemple.

Nous avons pris en compte les implications de performance et de stockage liées à la création d'index. Bien que les index permettent d'accélérer les requêtes, un excès d'index peut entraîner un surcoût de maintenance et de stockage. Par conséquent, nous avons limité le nombre d'index aux plus importants pour éviter tout impact négatif sur la base de données. Nous avons notamment ciblé les colonnes qui sont le plus susceptibles d'être filtrées (celles des table de ref et les foreign key notamment).



3) Création de vues pour simplifier l'accès aux données

Toutes les vues matérialisées et non matérialisées ont été créées sur les deux schémas de base de données GLPI CERGY et GLPI_PAU, respectivement pour les sites de CY TECH.

- Vues Matérialisées

Pour les vues matérialisées de notre projet, nous avons pris la décision stratégique d'en créer quatre spécifiques pour répondre à des besoins analytiques et opérationnels précis de notre base de données GLPI.

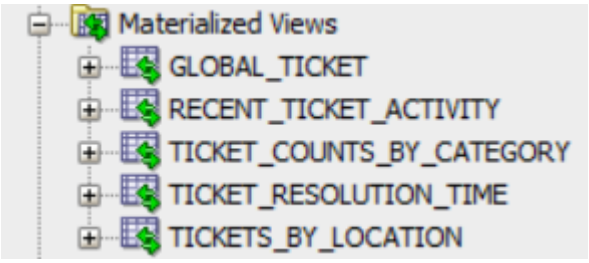
- **"Global_Ticket"**, a été conçue pour permettre une visualisation complète sur la table TICKETS, en y ajoutant les jointures sur les tables de références pour permettre une meilleure visibilité sur les tickets.

- **"Ticket_Counts_By_Category"**, a été conçue pour fournir un comptage précis et rapide du nombre de tickets par catégorie. Cette vue est cruciale pour évaluer la répartition des problèmes signalés par les utilisateurs et prioriser les efforts de résolution en conséquence.

- **"Tickets_By_Location"** offre une vue consolidée du volume de tickets par emplacement, facilitant ainsi l'identification des zones géographiques nécessitant une attention particulière.

- **"Ticket_Resolution_Time"** quant à elle, calcule le temps moyen nécessaire à la résolution des tickets, offrant ainsi un aperçu précieux des performances du support technique.

- **"Recent_Ticket_Activity"** permet un accès rapide aux cent derniers tickets ayant été modifiés, permettant de rester informé des derniers développements. Ces vues ont été matérialisées pour garantir des performances optimales lors de l'accès à ces informations cruciales.



- Vues Non Matérialisées

En ce qui concerne les vues non matérialisées, notre choix s'est porté sur six vues pour leur nature dynamique et leur nécessité de données en temps réel.

- **“Open_Tickets_By_Category”** offre une vision actualisée du nombre de tickets ouverts par catégorie, permettant de suivre de près la répartition des problèmes en cours de traitement et d'adapter les efforts en conséquence.

- **“Closed_Tickets”** permettant de suivre en temps réel les tickets fermés. Cette vue fournit une visibilité essentielle sur les résolutions de

problèmes et les achèvements de tâches, facilitant ainsi la gestion efficace du flux de travail et la prise de décision informée.

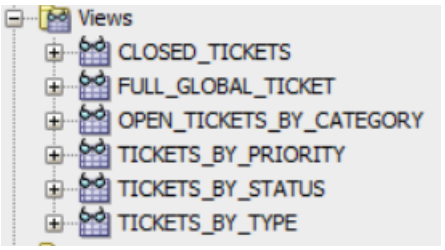
- **"Tickets_By_Priority"** permet une vision en temps réel du nombre de tickets classés par priorité, permettant une hiérarchisation efficace des tâches et une réponse rapide aux besoins critiques des utilisateurs.

- **“Tickets_By_Status”** propose une analyse dynamique du nombre de tickets par statut, cette vue aide les équipes à surveiller les flux de travail et à garantir une gestion fluide des tickets.

- **“Tickets_By_Type“** permet suivant la répartition des tickets par type, cette vue permet d'allouer efficacement les ressources, d'identifier les tendances et d'optimiser les processus de support.

- **“FULL_GLOBAL_TICKET”** est une vue consolidée qui combine les informations sur les tickets provenant des deux schémas de base de données, respectivement pour les sites de CY TECH. Elle utilise une jointure entre les vues "GLOBAL_TICKET" du schéma actuel et celle de l'autre schéma en utilisant le lien de base de données précédemment établi. Cette vue offre une vision complète et unifiée des tickets de tous les sites, permettant une gestion centralisée et une analyse globale des activités de support technique.

```
CREATE VIEW GLPI_CERGY.FULL_GLOBAL_TICKET
AS
SELECT * FROM GLPI_CERGY.GLOBAL_TICKET
UNION
SELECT * FROM GLPI_PAU.GLOBAL_TICKET@LK_GLPI_PAU;
```



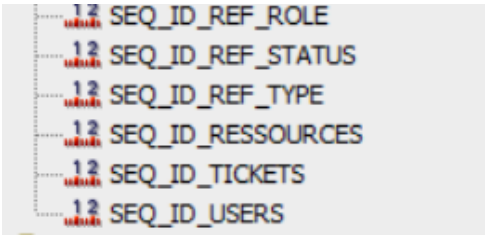
4) Utilisation de PL/SQL pour les triggers, procédures et fonctions

Pour faciliter l'utilisation, la maintenance et l'optimisation de la database, nous avons utilisé le PL/SQL et créé des triggers, des fonctions et des procédures. Leur utilisation s'inscrit dans une stratégie plus large visant à améliorer les performances et la fiabilité de notre système d'information.

• Les triggers

Nous avons créé un trigger par Table. Ces trigger assurés plusieurs choses :

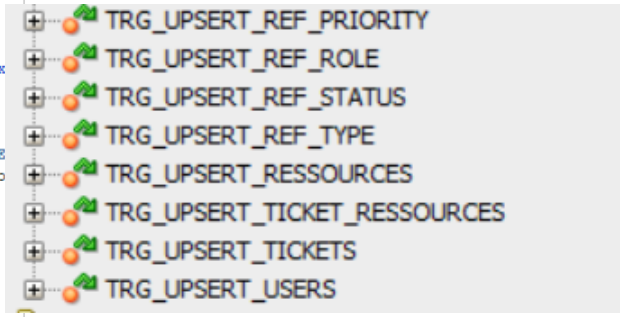
- À l'aide de séquence, ils assurent que les id des tables qui en possèdent sont correct lors des inserts en les remplaçant.



- Ils vérifient l'existence des références insérer/update (en supplément de la contrainte de FOREIGN Key qui le fait), et adaptent le message d'erreur.

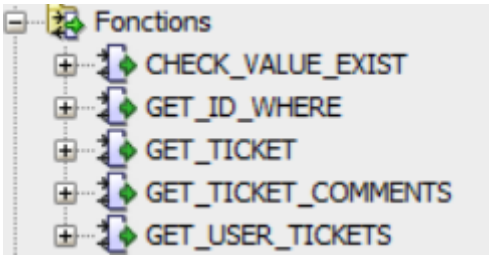
- Et évidemment, certaines colonnes nécessitent certains autres traitements et/ou vérifications réalisés par les triggers : majuscule au début des noms, concaténation de colonnes pour former une autre, vérification sur des dates…

```
-- Cri¿½ation du trigger pour USERS
CREATE OR REPLACE TRIGGER GLPI_CERGY.trg_upsert_users
BEFORE INSERT OR UPDATE ON GLPI_CERGY.USERS
FOR EACH ROW
BEGIN
    IF INSERTING THEN
      SELECT GLPI_CERGY.seq_id_users.nextval INTO :NEW.user_id FROM dual;
    END IF;

    IF (CHECK_VALUE_EXIST(:NEW.email, 'email', 'USERS')) THEN
        RAISE_APPLICATION_ERROR(-20009, 'Email ' || :NEW.email || ' already ex
    END IF;

     -- Vi¿½rifier si la valeur de fk_location existe dans la table LOCATIONS
    IF NOT CHECK_VALUE_EXIST(:NEW.fk_location, 'location_id', 'LOCATIONS') THE
        RAISE_APPLICATION_ERROR(-20010, 'Location with ID ' || :NEW.fk_locatio
    END IF;

    :NEW.last_name := INITCAP(:NEW.last_name);
    :NEW.first_name := INITCAP(:NEW.first_name);
    :NEW.company := INITCAP(:NEW.company);
EXCEPTION
```



- Les fonctions

Les fonctions peuvent accepter des paramètres en entrées, effectuer des opérations complexes, et retourner les valeurs que nous avons utilisées pour simplifier des opérations de vérification ou pour retourner des valeurs en entrant un simple id.
Par exemple, la fonction CHECK_VALUE_EXIST qui vérifie la présence d'une valeur dans une table. Cette fonction est beaucoup utilisée par nos procédures pour vérifier si une valeur spécifique existe dans une colonne d'une table donnée. Ainsi, on s'assure qu'il n'y ait pas de bug lors des appels sur des champs de la base de données qui pourraient être inexistants.
Nous avons également fait des fonctions pour encapsuler des Select statement filtrer, et qui retourne l'information dans une structure de donnée. Pour une éventuelle utilisation par un site web ou autre qui pourrait les appeler facilement, ce serait intéressant. Par exemple GET_TICKET qui renvoie toutes les informations d'un ticket (avec un id passer en paramètre) provenant de la vue GLOBAL_TICKET, ou GET_TICKET_COMMENTS qui retourne la liste des commentaires d'un ticket dont on a spécifier l'id en paramètre.

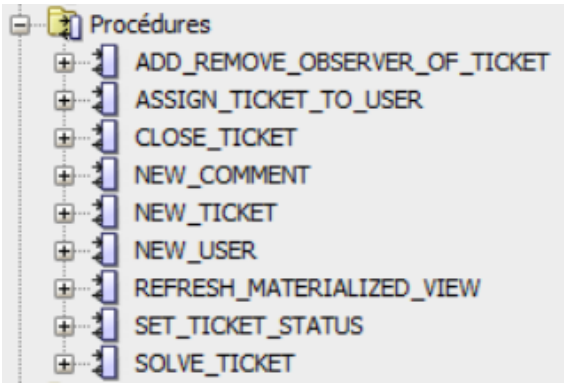


- Les procédures

Nous avons créé des procédures pour encapsuler des instructions permettant d'effectuer des opérations sur les données.
On a par exemple des procédures qui permettent d'agir directement sur la base de données. Elles ont pour intérêt d'encapsuler les actions et d'assurer de la fiabilité de l'action. Par exemple, lorsque l'on veut résoudre un ticket, il suffit d'appeler la procédure SOLVE_TICKET avec l'identifiant du ticket et une note de résolution.

Nous avons également produit des procédures pour simplifier la création de données.
Ces procédures comme NEW_TICKET, NEW_USER, NEW_COMMENT, ASSIGN_TICKET_TO_USER, etc. Elles prennent les paramètres nécessaires pour

créer un élément dans la database, en gérant les relations et les valeurs initiales notamment.



5) Gestion des autorisations et des rôles

Description des rôles mis en place sur la base de données

Dans le cadre de la gestion des autorisations sur la base de données, plusieurs rôles ont été définis pour contrôler l'accès aux données et garantir une sécurité adéquate. Voici les principaux rôles créés :

- GLPI_CERGY_OBSERVER : Ce rôle est attribué aux utilisateurs nécessitant uniquement des autorisations de lecture (SELECT) sur les vues de la base de données GLPI_CERGY.

- GLPI_CERGY_ANALYST : Ce rôle est attribué aux analystes qui ont besoin d'accéder et de mettre à jour certaines tables spécifiques de la base de données GLPI_CERGY, telles que les tables TICKETS et COMMENTS, en plus des autorisations de lecture sur l'ensemble des objets.

- GLPI_CERGY_DEV : Ce rôle est destiné aux développeurs qui nécessitent des autorisations étendues pour effectuer des opérations de lecture, écriture, mise à jour et suppression sur toutes les tables et vues de la base de données GLPI_CERGY, ainsi que sur les séquences associées.

- GLPI_CERGY_ADMIN : Ce rôle regroupe les privilèges de développement et d'administration sur la base de données GLPI_CERGY, incluant la création de liens de base de données, ainsi que les autorisations de connexion et de création de session.

Des rôles similaires ont également été créés pour la base de données GLPI_PAU, avec des noms correspondants (GLPI_PAU_OBSERVER, GLPI_PAU_ANALYST, GLPI_PAU_DEV, GLPI_PAU_ADMIN).

- GLPI_FULL_ADMIN : Le rôle "GLPI_FULL_ADMIN" a été créé pour les administrateurs système ou les super-utilisateurs qui nécessitent un accès complet aux deux schémas GLPI_CERGY et GLPI_PAU. Ce rôle a tous les privilèges des rôles administratifs spécifiques de chaque schéma, ainsi que le privilège global de connexion et de création de session. Il est destiné aux utilisateurs responsables de la gestion globale de l'ensemble de la base de données GLPI.

Justification des choix de sécurité et de gestion des accès

La mise en place de ces rôles permet de garantir une gestion sécurisée des accès à la base de données GLPI_CERGY et GLPI_PAU, en accordant des autorisations appropriées à chaque catégorie d'utilisateurs. Cette approche basée sur les rôles permet de limiter les risques de compromission des données en accordant uniquement les autorisations nécessaires à chaque utilisateur en fonction de ses responsabilités et de ses besoins opérationnels. De plus, en séparant les privilèges de développement, d'analyse et d'administration, nous assurons une gestion efficace et contrôlée des accès à la base de données, en conformité avec les principes de sécurité et de gouvernance des données.

IV. Conclusion

En résumé, notre projet pour améliorer la base de données GLPI de CY Tech a été une démarche méthodique et collaborative visant à résoudre des problèmes de conception de bases de données. Après avoir examiné en profondeur l'ancien système, nous avons identifié les parties que nous pourrions améliorer, et les inefficacités pour élaborer une nouvelle structure de base de données mieux adaptée aux besoins spécifiques de CY Tech, et de sa fragmentation.

En somme, nous avons développé une modélisation conceptuelle, logique et physique en tenant compte des exigences fonctionnelles et des contraintes techniques. Cela, en utilisant des bonnes pratiques de gestion de bases de données pour permettre, comme l’utilisation d'espace de stockage dédiés, des index bien optimisés et l'intégration de PL/SQL pour les procédures et fonctions, triggers, etc.. Pour atteindre une utilisation simplifiée, évoluable et maintenable de la base de données GLPI de CY Tech.

V. Annexes

MCD

MCD de notre base de donnée