Création d'un tableau de bord pour l'application GLPI via Microsoft Access

Rapport de Stage

Salvo Luka

Université de Lorraine

Institut Universitaire de Technologie Nancy-Charlemagne

Département Informatique

2Ter Bd Charlemagne, 54000 Nancy

Année universitaire 2024-2025

Création d'un tableau de bord pour l'application GLPI via Microsoft Access

Salvo Luka

Remerciements

Tout d'abord, je tiens à remercier l'ensemble du personnel pour son accompagnement durant ce stage et pour avoir partagé son savoir-faire avec moi.
Je remercie également mon tuteur de stage, **Monsieur GUENEGO**, pour sa disponibilité et son aide précieuse tout au long de ce stage.
Je tiens aussi à exprimer ma gratitude envers mon maître de stage, **Monsieur Christophe DAHLEM**, pour m'avoir accueilli au sein de son entreprise et m'avoir donné l'opportunité de réaliser ce stage dans un cadre aussi enrichissant, je remercie aussi mes collègues : Charles, Eric, Valentin, Lyassine, Yannick, Alan, Sébastien, Nicolas et Philippe pour leur accueil et leur aide.
Enfin, je remercie mes enseignants de l'**IUT Nancy-Charlemagne** pour leur accompagnement tout au long de ma formation et pour les compétences qu'ils m'ont transmises, me permettant ainsi d'effectuer ce stage dans de bonnes conditions.
Je souhaite également exprimer ma reconnaissance à mes parents et grands-parents pour leur soutien indéfectible tout au long de cette aventure.
Un grand merci à **Monsieur Joël Lanno**, chef du département informatique, ainsi qu'à **Monsieur Roland Roth**, président de la Communauté d'Agglomération Sarreguemines Confluences, pour leur soutien et leur confiance.

Table des matières

1 Introduction

Ce stage a été effectué dans le cadre de la deuxième année de mon BUT (Bachelor Universitaire de Technologie) Informatique. Il s'est déroulé du 17 février au 11 avril 2025, pour une durée de 8 semaines, au sein de l'équipe informatique de la **Communauté d'Agglomération Sarreguemines Confluences (CASC)**. Cette expérience m'a permis de découvrir le domaine du traitement des données à travers la gestion des tickets informatiques via l'application **GLPI**.

L'objectif principal de mon stage était de créer un tableau de bord pour l'application **GLPI** en utilisant **Microsoft Access**. Ce tableau de bord vise à améliorer la gestion et l'analyse des tickets en fournissant des outils interactifs, des visualisations claires des données, ainsi qu'une fonctionnalité d'exportation vers un fichier Excel regroupant les informations essentielles pour un éventuel rendu. Le but de la création de ce tableau de bord était de répondre à un besoin de l'équipe informatique de la CASC, notamment a besoin de Monsieur Christophe Dahlem, responsable du pôle support, qui doit produire des statistiques mensuelles sur les tickets informatiques traités. Ces statistiques sont essentielles pour évaluer la performance du service informatique et justifier les performances aux élus de la Communauté d'agglomération de Sarreguemines.

Microsoft Access est une application de gestion de bases de données permettant de stocker, manipuler et organiser des informations. Elle est souvent utilisée pour créer des formulaires, des requêtes, des rapports et des tableaux de bord. **Microsoft Access** offre également la possibilité de gérer des actions via des macros ou du code **VBA**, un langage sur lequel je me suis appuyé pour développer les fonctionnalités du tableau de bord, comme détaillé plus loin dans ce rapport.

GLPI est une application open-source de gestion des services informatiques et des actifs. Elle permet de gérer les tickets, les demandes, les incidents, les problèmes, les changements, les actifs, les contrats, les fournisseurs, les licences, les documents, les réservations de ressources, les annuaires, les utilisateurs et les groupes.

Ce rapport est structuré comme suit : après une présentation de l'entreprise et du service informatique, nous expliquerons le contexte et les raisons de la création du tableau de bord, puis détaillerons les missions confiées, les fonctionnalités attendues, l'approche technique adoptée et le développement du projet. Enfin, une conclusion présentera un bilan des apports de ce stage et des perspectives d'avenir.

1.1 Présentation de l'entreprise

La **Communauté d'Agglomération Sarreguemines Confluences (CASC)** est une collectivité et un Établissement public de coopération intercommunale (EPCI) regroupant 38 communes et comptant environ 67 000 habitants. Située dans le département de la Moselle, en région Grand Est, elle est présidée par **Monsieur Roland Roth**.

La CASC a pour mission de renforcer la qualité de vie sur son territoire en mutualisant les compétences des communes et en mettant en œuvre des projets de développement local. Elle joue un rôle clé dans la gestion de services publics tels que les transports, la gestion des déchets, et l'assainissement, tout en mettant un accent particulier sur l'attractivité économique et touristique de la région.

La CASC est composée de 10 grands services, parmi lesquels la direction générale des services, la direction de la communication et du protocole, la direction des ressources humaines, ou encore la direction des services techniques. Cette dernière intègre 8 services techniques différents, dont le service informatique, qui sera détaillé dans la section suivante.

FIGURE 1 – Les locaux de la CASC à Sarreguemines. Source : Créé par l'auteur.

1.2 Présentation du service informatique

Le service informatique de la **Communauté d'Agglomération Sarreguemines Confluences (CASC)** est composé de 11 membres, sous la direction de **Monsieur Joël Lanno**. Ce service est en charge de la gestion du parc informatique, de la maintenance des équipements, de l'administration des logiciels, et de la mise en œuvre de solutions numériques pour les différents services de l'agglomération et de la ville de Sarreguemines.

Le service est structuré en plusieurs pôles :

— **Support informatique**, dirigé par **Monsieur Christophe Dahlem**, compte quatre collaborateurs. Ce pôle gère l'application métier **GLPI**, un outil open-source de gestion des tickets informatiques, pour lequel j'ai développé un tableau de bord via **Microsoft Access**.

— **Gestion de projets informatiques**, sous la responsabilité de **Monsieur Alan Massia**, regroupe trois collaborateurs. Ce pôle supervise le déploiement de solutions numériques.

En collaboration avec le service informatique de la ville de Sarreguemines, le service de la CASC propose des solutions adaptées aux besoins des deux entités. Parmi les outils gérés, on trouve :

— **Digitech**, une solution de gestion des cartes de bus pour le transport urbain, en collaboration avec la ville de Sarreguemines.

— **AIRS Delib**, un logiciel dédié à la gestion des rapports et délibérations communautaires.

— Des applications pour la gestion des stations d'épuration, soutenant les services techniques de l'agglomération.

Ces outils illustrent l'engagement du service informatique à fournir des solutions adaptées aux besoins variés de la CASC, tout en soutenant les élus (président, vice-présidents, conseillers communautaires) dans leurs missions.

FIGURE 2 – Bâtiment du service informatique de la CASC à Sarreguemines. Source : Créé par l'auteur.

1.3 Contexte et justification du tableau de bord

Le tableau de bord a été développé pour répondre à un besoin récurrent du service informatique de la **CASC**. Chaque mois, **Monsieur Christophe Dahlem**, responsable du pôle support, doit produire des statistiques détaillées sur les tickets informatiques traités. Ces statistiques permettent d'obtenir une vision globale de l'efficacité du service informatique, en mesurant des indicateurs tels que le nombre de tickets ouverts et fermés, les délais de résolution, ou encore la répartition des incidents et des demandes par catégorie.

Ces données sont ensuite présentées aux élus de la CASC, notamment le président, les vice-présidents, et les conseillers communautaires, lors de réunions stratégiques. Elles servent à évaluer la performance du service informatique, à identifier les axes d'amélioration, et à justifier les besoins en ressources ou en outils supplémentaires. Avant la création de ce tableau de bord, la production de ces rapports était manuelle et chronophage, nécessitant des extractions et des analyses complexes. Mon projet visait donc à automatiser et simplifier ce processus, tout en offrant une interface intuitive pour visualiser et manipuler les données.

1.4 Missions confiées

Lors de mon arrivée au service informatique de la **CASC**, ma mission principale était de concevoir et développer un tableau de bord pour l'application **GLPI** via **Microsoft Access**. Ce tableau de bord devait permettre une visualisation claire et interactive des données des tickets, tout en offrant la possibilité de modifier certains champs directement dans la base de données pour faciliter les analyses statistiques ultérieures. Ce tableau de bord devait contenir des statistiques sur les tickets GLPI mais aussi sur l'anti virus du service informatique **WithSecure**. Ce tableau de bord a pour but de servir mensuellement à la production de statistiques sur les tickets informatiques traités par le service informatique de la **CASC**, il est destiné à être utilisé par le chef de l'équipe support, **Monsieur Christophe Dahlem**, et a été mise en place pour facilité la sortie de statistques en le moins de temps possible.

Un défi majeur au début du stage a été de m'adapter aux outils et technologies utilisés. Cela incluait l'apprentissage du code **VBA**, la maîtrise du fonctionnement de **Microsoft Access**, et la compréhension de la structure des données de **GLPI**, qui étaient complexes et dispersées dans plusieurs tables. Ces obstacles initiaux m'ont permis de développer mes compétences techniques et ma capacité à travailler dans un environnement professionnel avec des contraintes de temps et des attentes précises.

En parallèle, j'ai également conçu un cahier des charges pour m'assurer que mes propositions étaient en adéquation avec les besoins de l'équipe informatique. Ce document a été essentiel pour clarifier les attentes et les fonctionnalités du tableau de bord.

Lors de la mise en production de l'application **Access**, j'ai été amené à rédiger un guide d'utilisation pour faciliter la prise en main de l'outil par les utilisateurs. Ce guide, conçu pour être clair et accessible, présente les différentes fonctionnalités du tableau de bord, les étapes pour effectuer des analyses, et des conseils pour résoudre ou éviter les problèmes courants.

1.5 Fonctionnalités attendues du tableau de bord

L'objectif principal du tableau de bord était de fournir des outils pour extraire et analyser les données de **GLPI** afin de produire des rapports mensuels sur des métriques clés. Parmi les données à extraire et à analyser, les plus importantes étaient :

- Le nombre de tickets ouverts et fermés par mois, année et catégorie.
- Le pourcentage de tickets ouverts et fermés par mois, année et catégorie.
- Le pourcentage de tickets clos en moins de 4 heures, 24 heures, 7 jours, et plus de 7 jours, par mois, année et catégorie.
- Le pourcentage de demandes ouvertes par mois, année et catégorie.
- Le pourcentage d'incidents ouverts par mois, année et catégorie.
- Le nombre d'incidents ouverts par mois, année et catégorie.
- Le nombre de demandes ouvertes par mois, année et catégorie.

Pour répondre à ces besoins, j'ai intégré des fonctionnalités d'exportation des données vers Excel. Par exemple, une table regroupant les informations essentielles peut être exportée dans un fichier Excel grâce à un bouton et du code **VBA**. De plus, j'ai rendu cet export personnalisable, permettant à l'utilisateur de filtrer les données avant de les exporter, afin de répondre à des besoins spécifiques d'analyse.

Il était convenu, au fil des semaines, de permettre des modifications sur certains champs des tickets, notamment les entités, les types, et les catégories. Pour cela, le tableau de bord devait intégrer des combo boxes dans le formulaire, synchronisées via un bouton et du code **VBA**, qui met à jour directement les données dans l'application web **GLPI** à l'aide d'un driver **ODBC**.

Enfin, une des dernières fonctionnalités attendues était l'extraction de statistiques depuis l'application **WithSecure**. Le tableau de bord devait être capable de lire un fichier `.xlsx`, d'en extraire les données, de les insérer dans une table, et de traiter les champs essentiels pour générer des statistiques.

1.6 Approche technique et outils

Pour répondre aux attentes de l'équipe informatique, il a été décidé d'utiliser des requêtes **SQL** pour extraire et manipuler les données, plutôt que de s'appuyer uniquement sur l'interface graphique de **Microsoft Access**, notamment les macros. Cette approche offrait plusieurs avantages : une meilleure lisibilité des requêtes pour l'équipe, la possibilité de les modifier ou d'en ajouter facilement, et une compréhension plus claire de leur objectif. Les requêtes **SQL** ont été combinées avec du code **VBA** pour automatiser les tâches et gérer les interactions utilisateur dans le tableau de bord.

Le choix de **Microsoft Access** comme outil principal a été motivé par sa simplicité d'utilisation et sa capacité à créer des formulaires interactifs, tout en permettant une maintenance aisée par l'équipe informatique après mon départ. Cependant, cela a également nécessité de surmonter certaines limitations, notamment pour les requêtes complexes, ce qui m'a conduit à approfondir mes compétences en **VBA** et **SQL**.

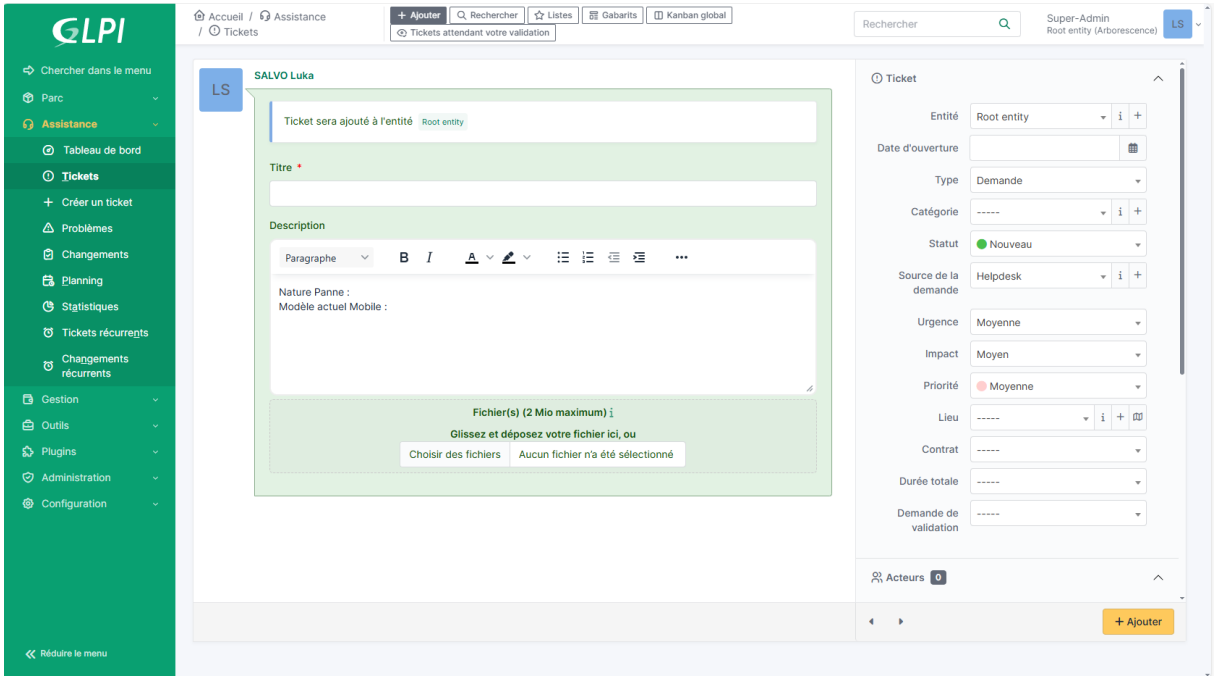


FIGURE 3 – Interface de l'application GLPI lors de la création d'un ticket.

2 Développement

Le développement a constitué la partie centrale de mon stage, axée sur la création d'un tableau de bord interactif pour visualiser, analyser et modifier les données issues de l'application **GLPI** via **Microsoft Access**. Ce projet a nécessité une approche structurée, combinant la conception de bases de données, le développement de formulaires interactifs, l'écriture de requêtes **SQL**, et l'automatisation via du code **VBA**. Voici les principales étapes et composants développés, ainsi que les défis rencontrés et les solutions apportées.

2.1 Analyse des besoins

L'analyse des besoins a été la première étape cruciale de mon stage, réalisée dès la première semaine. Elle a permis de comprendre les attentes du service informatique de la **Communauté d'Agglomération Sarreguemines Confluences (CASC)** concernant l'analyse et la visualisation des données issues de l'application **GLPI**. Cette phase a été essentielle pour poser des bases solides avant de passer à la conception et au développement du tableau de bord.

2.1.1 Recueillir les besoins fonctionnels

Pour recueillir les besoins, j'ai organisé des séances de mise au point avec mon maître de stage, **Monsieur Dahlem**, dès les premiers jours du stage. Ces réunions ont été cruciales pour identifier les attentes fonctionnelles du tableau de bord. Le service informatique souhaitait un outil capable de fournir des statistiques précises sur les tickets, telles que :

- Le nombre de tickets ouverts et fermés par mois, année et catégorie.
- Les pourcentages de tickets ouverts et fermés par mois, année et catégorie.
- Les pourcentages de tickets clos en moins de 4 heures, 24 heures, 7 jours, ou plus, par mois, année et catégorie.
- Les pourcentages et nombres d'incidents et de demandes, par mois, année et catégorie.

Un autre besoin clé était la possibilité de modifier directement certaines données dans la base, comme le type de ticket (Incident ou Demande), la catégorie ou l'entité associée. Cela permettait de corriger des erreurs ou d'adapter les données pour des analyses statistiques plus précises. Enfin, le service informatique a exprimé le besoin d'exporter ces données sous forme de tableaux Excel pour faciliter le partage et l'analyse externe. Actuellement, une seule personne est désignée pour utiliser ce tableau de bord, mais les fonctionnalités devaient être suffisamment

flexibles pour une utilisation future par d'autres membres de l'équipe.

2.1.2 Comprendre la structure des données GLPI

Un défi important a été de comprendre la structure des données de **GLPI**, qui repose sur une base de données relationnelle (MariaDB). Pour interroger ces données, j'ai utilisé un driver **ODBC** MySQL configuré entre **Microsoft Access** et la base MariaDB de **GLPI**. Cela m'a permis d'explorer une copie des tables (afin de préserver la base de données actuelle) et leurs relations, identifiant ainsi les champs essentiels à retranscrire dans le tableau de bord. Ces données, une fois extraites, ont été centralisées dans la table `DataWarehouse_Tickets` pour faciliter les analyses.

Les données des tickets sont réparties dans plusieurs tables interconnectées, telles que `glpi_tickets` (contenant les informations brutes des tickets), `glpi_entities` (pour les entités), `glpi_itilcategories` (pour les catégories), et d'autres tables comme `glpi_users` ou `glpi_ticketscosts`. Par exemple, pour calculer les pourcentages de tickets clos dans différents délais, il fallait extraire les dates de création et de clôture des tickets (`date` et `date_close`) ainsi que leur statut (`status`). Pour différencier les incidents des demandes, le champ `type` (1 pour Incident, 2 pour Demande) était nécessaire ; pour le comprendre, j'ai analysé la base de données via le driver **ODBC** MySQL en fonction de l'id du ticket, déterminant quel chiffre correspondait à quel type. Ces données, accessibles directement via la connexion **ODBC**, nécessitaient un traitement pour être exploitables dans le data warehouse.

La complexité de cette structure a nécessité une analyse approfondie pour identifier les relations entre les tables et déterminer comment extraire les informations pertinentes. Par exemple, j'ai dû comprendre que le champ `entities_id` dans `glpi_tickets` était une clé étrangère liée à la table `glpi_entities`, et que le champ `itilcategories_id` était lié à `glpi_itilcategories`. Cette compréhension a été essentielle pour concevoir le data warehouse dans la phase suivante.

2.1.3 Formalisation des besoins

Pour formaliser les attentes, j'ai créé un **diagramme de cas d'utilisation** (voir Figure 4) ainsi qu'un **cahier des charges** dès la première semaine. Le diagramme a permis de valider que les besoins étaient bien compris et de clarifier les fonctionnalités principales du tableau de bord : visualisation des données, modification des tickets, et exportation vers Excel. Il a également

mis en évidence les différents acteurs impliqués, notamment les techniciens qui utiliseraient le tableau de bord pour analyser les tickets, et les administrateurs qui auraient besoin de modifier les données.

Le cahier des charges a détaillé les fonctionnalités attendues, les métriques à calculer, et les contraintes du projet. Il a servi de document de référence tout au long du stage, permettant à l'équipe informatique et à moi-même de rester alignés sur les objectifs.

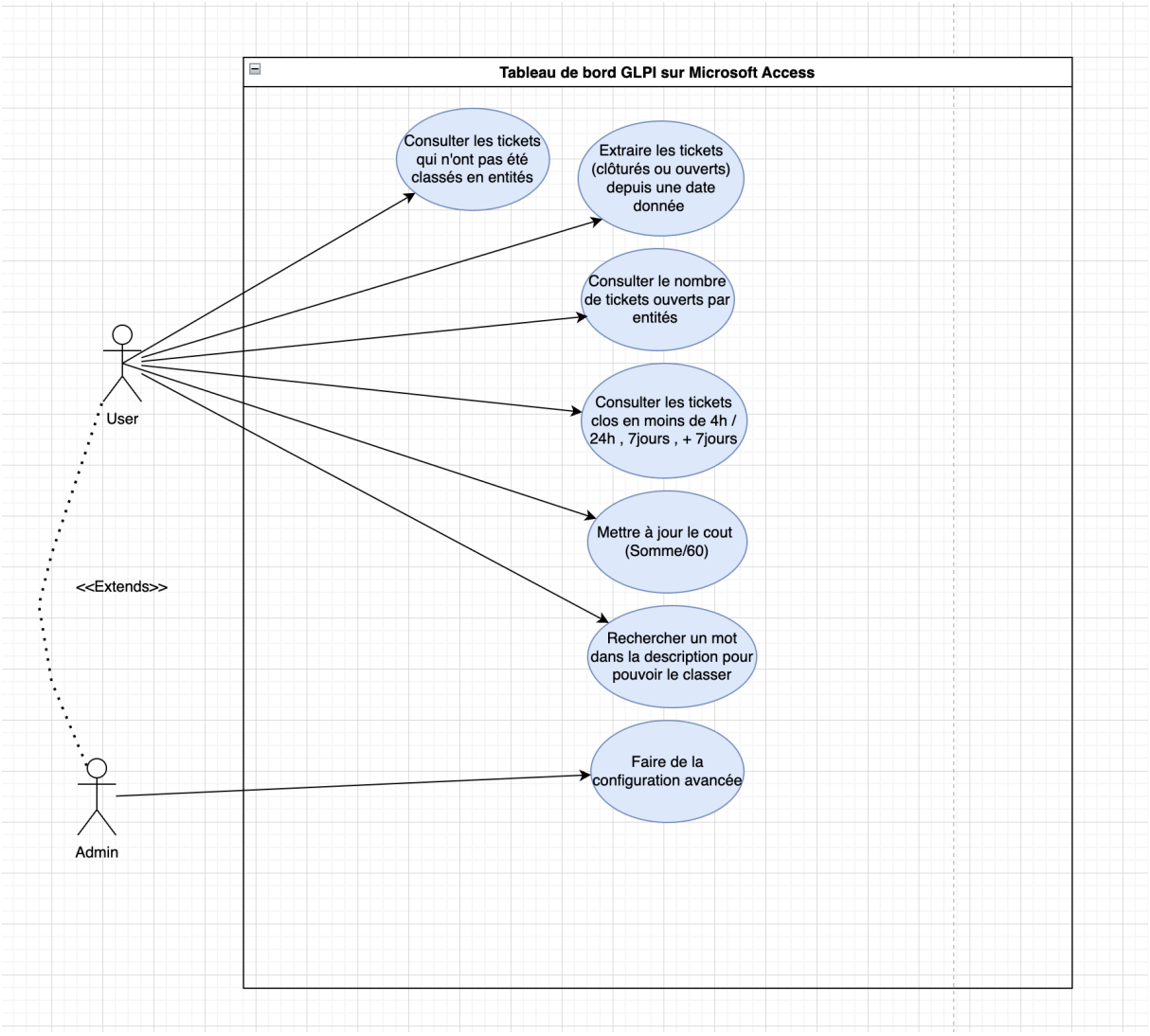


FIGURE 4 – Diagramme de cas d'utilisation pour le tableau de bord.

2.1.4 Contraintes et priorisation

Une contrainte majeure était le temps limité du stage (8 semaines), ce qui a nécessité une priorisation des fonctionnalités. Les visualisations et les modifications de données ont été considérées comme prioritaires, car elles répondaient directement aux besoins les plus urgents du service informatique. En revanche, des fonctionnalités avancées, comme des filtres dynamiques plus complexes ou l'intégration de nouvelles métriques, ont été envisagées comme des évolutions futures.

Une autre contrainte était la nécessité d'adapter le tableau de bord aux compétences de l'équipe informatique, qui était plus familière avec **Microsoft Access** qu'avec d'autres technologies comme PHP ou des frameworks web. Cela a influencé les choix techniques, notamment l'utilisation de **VBA** et de requêtes **SQL** pour automatiser les tâches et gérer les interactions utilisateur.

Pour remédier à la contraite de temps, chaque semaine je me fixais un objectif de développement, en me concentrant sur une fonctionnalité ou un ensemble de fonctionnalités spécifiques. Par exemple, Voici mon diagramme de Gantt qui illustre la planification des tâches et des fonctionnalités à développer au cours du stage. Ce diagramme a été mis à jour régulièrement pour refléter l'avancement du projet et les ajustements nécessaires.

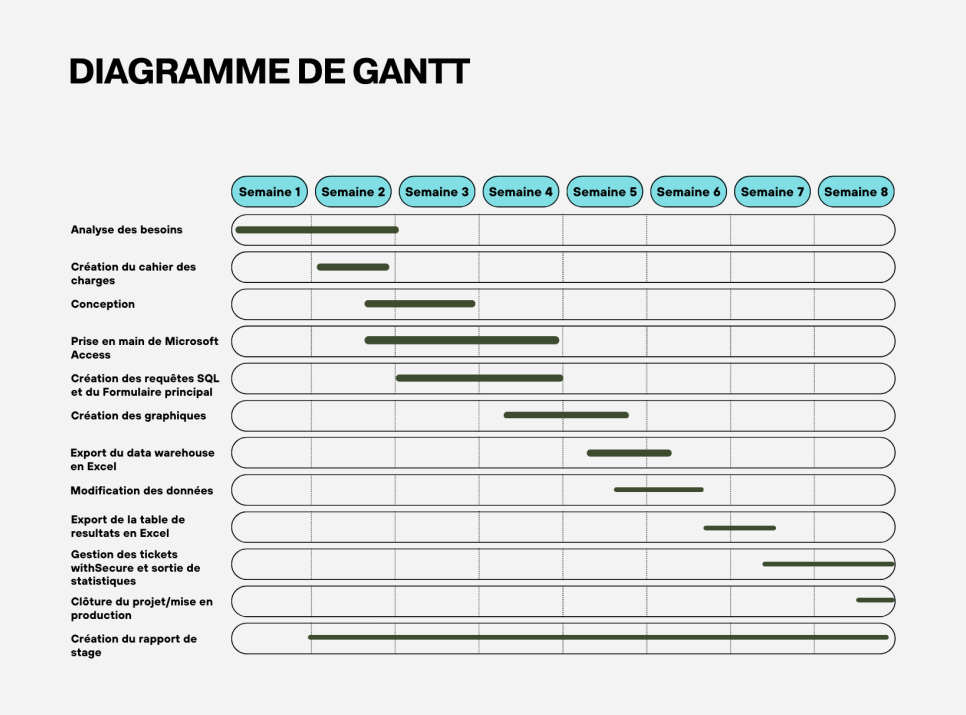


FIGURE 5 – Diagramme de Gantt représentant les tâches tout au long du stage.

2.2 Conception

La phase de conception a été une étape essentielle pour structurer le projet et garantir un développement efficace. Elle a suivi l'analyse des besoins et s'est appuyée sur une compréhension approfondie des données de **GLPI** et des attentes du service informatique de la **Communauté d'Agglomération Sarreguemines Confluences (CASC)**. L'objectif principal de cette phase était de concevoir un système capable de centraliser, structurer et manipuler les données de **GLPI** pour répondre aux besoins d'analyse et de modification exprimés.

2.2.1 Algorithmique

Avant de commencer à coder une fonction, j'ai adopté une approche systématique en concevant à la main, sous forme d'algorithmes, les différentes étapes nécessaires à son fonctionnement. Cette méthode m'a permis de structurer ma pensée, de mieux comprendre les besoins de chaque fonction, et d'anticiper les problèmes potentiels avant de passer à l'implémentation en **VBA**. En écrivant les algorithmes sur papier, j'ai pu décomposer les tâches complexes en étapes simples, identifier les variables nécessaires, et prévoir les cas particuliers ou les erreurs possibles.

Par exemple, pour la fonction `CommandeModification_Click`, qui gère les modifications des champs d'un ticket via le formulaire interactif, j'ai conçu un algorithme détaillant les étapes suivantes : vérification de la sélection d'un ticket, récupération des valeurs des combo boxes, comparaison avec les données actuelles, et exécution d'une requête `UPDATE` si nécessaire. Cet algorithme, illustré ci-dessous, m'a aidé à organiser le code de manière modulaire et à éviter des erreurs lors de la mise à jour des données dans `DataWarehouse_Tickets`.

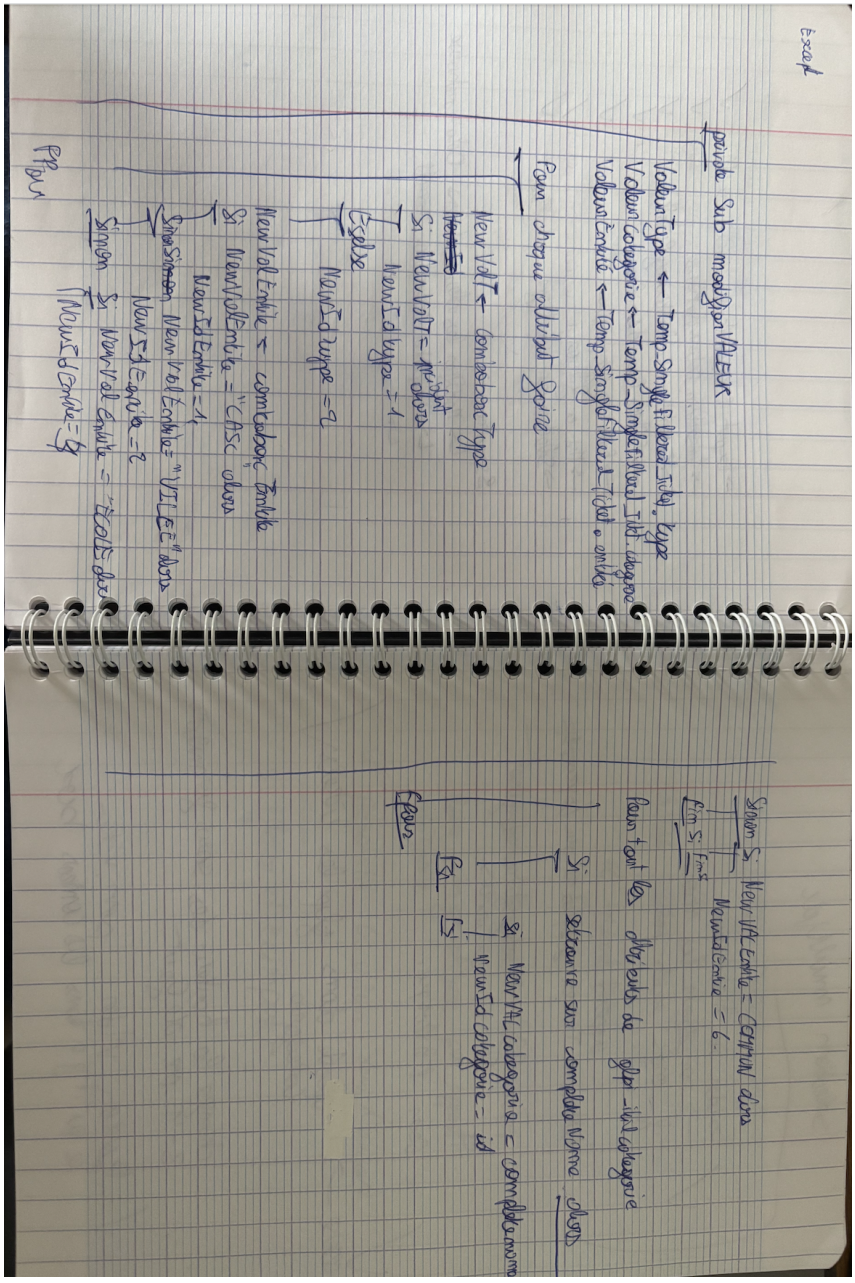


FIGURE 6 – Première version de l'algorithme de la fonction `CommandeModification_Click`, conçu pour gérer les modifications des tickets.

Cette approche n'a pas été limitée à une seule fonction, mais a été appliquée de manière systématique à d'autres parties du projet. Par exemple, pour la fonction `Remplacer_caracteres_specials` , utilisée pour supprimer les caracteres spéciaux des descriptions, titres, demandeurs, techniciens des tickets, j'ai également conçu un algorithme préalable. Cet algorithme décrivait les étapes suivantes : vérifier si la chaîne est vide, parcourir la chaîne pour identifier les caractères spéciaux, remplacer ces caractères par des espaces, et nettoyer les espaces excédentaires. Cette planification m'a permis d'optimiser la fonction en évitant des boucles inutiles et en gérant efficacement les cas particuliers, comme les descriptions vides ou les balises imbriquées.

L'utilisation d'algorithmes écrits à la main a eu plusieurs bénéfices concrets dans le développement du tableau de bord. Tout d'abord, elle m'a permis de réduire le temps de débogage en identifiant les erreurs potentielles dès la phase de conception. Par exemple, dans `CommandeModification_Click`, l'algorithme m'a aidé à prévoir la nécessité de vérifier si un ticket était sélectionné avant de tenter une modification, évitant ainsi des erreurs d'exécution. Ensuite, cette approche a facilité la communication avec mon tuteur de stage, **Monsieur Dahlem**, en lui présentant des algorithmes clairs pour valider la logique avant le codage. Enfin, elle a contribué à produire un code plus structuré et maintenable, car chaque fonction était basée sur une logique préalablement définie.

Cependant, cette méthode manuelle présentait certaines limites. Écrire des algorithmes à la main pouvait être chronophage, surtout pour des fonctions complexes nécessitant plusieurs itérations. De plus, sans outils de modélisation formels, il était parfois difficile de visualiser les interactions entre différentes fonctions. Malgré ces limites, cette approche artisanale s'est révélée particulièrement adaptée au contexte du stage, où la simplicité et la clarté étaient prioritaires.

En conclusion, la conception algorithmique a joué un rôle clé dans la réussite du projet en garantissant une implémentation robuste et efficace des fonctionnalités du tableau de bord. Cette méthode m'a non seulement aidé à produire un code de qualité, mais m'a également appris à adopter une démarche rigoureuse et réfléchie dans le développement logiciel, une compétence que je continuerai à appliquer dans mes projets futurs.

2.2.2 Conception du data warehouse

Pour faciliter l'extraction et l'analyse des données issues de **GLPI**, j'ai créé un **data warehouse** dans **Microsoft Access**. **GLPI** repose sur une base de données relationnelle (MariaDB), que j'ai interrogée via un driver **ODBC** MySQL pour extraire les données essentielles. Ces données, provenant de nombreuses tables interconnectées telles que `glpi_tickets`, `glpi_entities`, `glpi_itilcategories`, `glpi_users`, `glpi_locations`, et `glpi_tickettypes`, étaient dispersées et complexes à analyser directement. Le data warehouse a donc été conçu pour retranscrire ces données essentielles dans une table principale, `DataWarehouse_Tickets`, tout en conservant les tables de référence nécessaires pour maintenir les relations.

Le data warehouse est composé de six tables principales :

— `glpi_tickets` : Table contenant les informations brutes des tickets, incluant des champs comme `id` (clé primaire), `date` (date de création), `date_close` (date de clô-

ture), `status` (statut du ticket), `entities_id` (clé étrangère vers `glpi_entities`), et `itilcategories_id` (clé étrangère vers `glpi_itilcategories`).

— `glpi_entities` : Table de référence contenant les entités de **GLPI**, avec des champs comme `id` (clé primaire) et `name` (nom de l'entité). Elle est utilisée pour associer chaque ticket à une entité via le champ `entities_id`.

— `glpi_itilcategories` : Table de référence pour les catégories des tickets, avec des champs comme `id` (clé primaire) et `completename` (nom complet de la catégorie). Elle permet de lier chaque ticket à une catégorie via le champ `itilcategories_id`.

— `glpi_users` : Table contenant les informations sur les utilisateurs, comme les techniciens assignés aux tickets, avec des champs comme `id` (clé primaire) et `name` (nom de l'utilisateur). Elle est utilisée pour des analyses futures.

— `glpi_tickets_users` : Table de liaison entre les tickets et les utilisateurs. Elle permet de visualiser quels tickets ont été attribués ou créés par les utilisateurs.

— `glpi_ticketscosts` : Table contenant les informations sur le coût des tickets. Elle est utilisée uniquement pour les tickets dont l'entité est "Écoles".

Pour remplir `DataWarehouse_Tickets`, j'ai utilisé du code **VBA** pour écrire des requêtes **SQL** complexes, car l'interface graphique de **Microsoft Access** ne permet pas facilement de réaliser des requêtes impliquant des jointures multiples et des transformations de données. Par exemple, j'ai utilisé des jointures `LEFT JOIN` pour associer les tickets à leurs entités et catégories, et j'ai ajouté des champs calculés comme `nom_type` pour convertir le champ `type` (1 ou 2) en un libellé lisible ("Incident" ou "Demande").

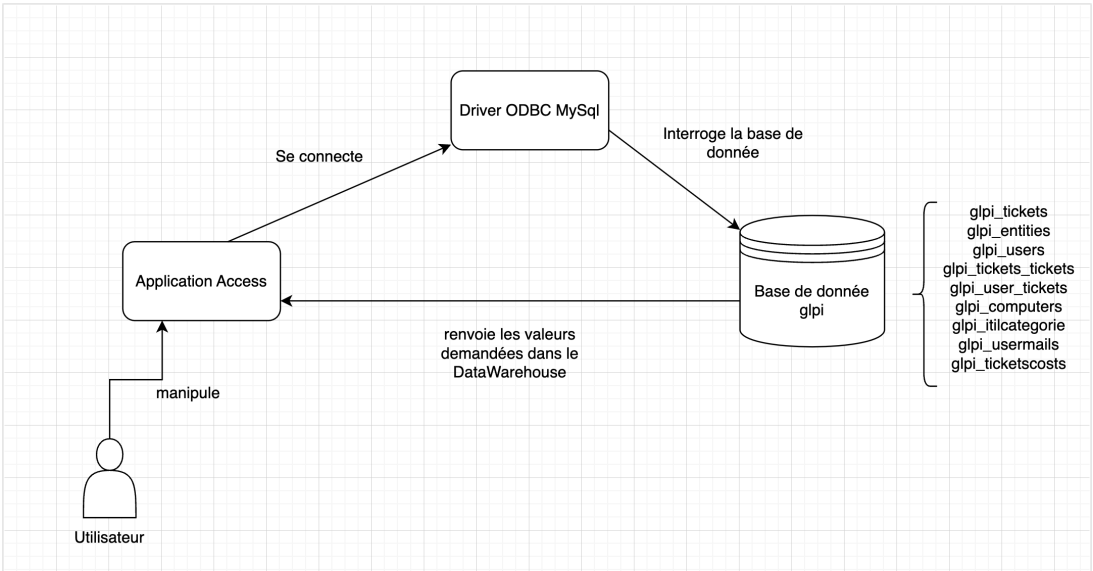


FIGURE 7 – Schéma de la conception de la base de données.

Sur le schema au dessus, vous pouvez visualiser comment chaque éléments s'articule entre eux pour former le data warehouse. L'application du tableau de bord se connecte au driver ODBC pour interroger la base de données **GLPI** et remplir la table `DataWarehouse_Tickets` avec les champs uniquement essentiels. Cette table est ensuite utilisée pour alimenter le tableau de bord, permettant une visualisation et une analyse des données des tickets.

Un défi majeur lors de la conception du data warehouse a été de gérer les incohérences dans les données de **GLPI**. Par exemple, certains tickets n'avaient pas d'entité ou de catégorie associée, ce qui générait des valeurs nulles dans les champs correspondants. Pour résoudre ce problème, j'ai utilisé la fonction `Nz` dans les requêtes pour remplacer les valeurs nulles par des valeurs par défaut (par exemple, " " pour `nom_entities`), garantissant ainsi la cohérence des données retranscrites pour les analyses. Vous trouverez plus d'élement concernant cette partie dans la section **2.3.2**.

En résumé, le data warehouse`DataWarehouse_Tickets` est une table de travail qui centralise les données essentielles des tickets, facilitant ainsi leur analyse et leur visualisation dans le tableau de bord. Elle est alimentée par des requêtes **SQL** exécutées via du code **VBA**, permettant une flexibilité et une adaptabilité aux besoins d'analyse du service informatique. Cette table est essentielle car, elle permet de centraliser les données et donc d'avoir une sauvegarde de la base de données **GLPI**. Elle permet aussi d'intermédiaire entre la base de donnée et le tableau de bord et sert de parade pour ne pas casser la base de données **GLPI**.

2.2.3 Conception du formulaire interactif

Un autre aspect clé de la conception a été la création d'un **formulaire interactif** dans **Microsoft Access** pour permettre aux utilisateurs de visualiser et de modifier les données. Ce formulaire, nommé `Formulaire`, devait répondre à plusieurs besoins identifiés lors de l'analyse : afficher les données des tickets, permettre leur modification, et offrir des options d'exportation et de visualisation graphique.

Le formulaire est structuré comme suit :

— **Un sous-formulaire** affichant les données de `DataWarehouse_Tickets` sous forme de grille, avec des colonnes comme `id`, `nom_entities`, `nom_type`, `nom_categorie`, `date`, et `date_close`. Ce sous-formulaire permet à l'utilisateur de sélectionner un ticket pour le modifier.

— **Trois combo boxes** (`comboNomEntities`, `comboNomType`, `comboNomCategorie`) pour modifier les champs `entities_id`, `type`, et `id_categorie` d'un ticket sélectionné. Ces combo boxes sont remplies dynamiquement à partir des tables de référence (`glpi_entities`, `DataWarehouse_Tickets`, et `glpi_itilcategories`).

— **Des boutons** pour filtrer les données, sélectionner un ticket, sauvegarder les modifications, exporter les données vers Excel, et ouvrir un formulaire de graphiques.

— **Section WithSecure** : une partie du formulaire permet d'extraire les données de **WithSecure**. L'utilisateur peut sélectionner un fichier `.xlsx` contenant ces données, puis les informations pertinentes sont affichées dans une zone de texte. Cela facilite l'analyse des données de sécurité et leur intégration dans le tableau de bord. De plus, un bouton permet de calculer les pourcentages de **WithSecure**, générant automatiquement un fichier Excel qui est sauvegardé sur le bureau de l'utilisateur.

Un défi important lors de la conception du formulaire a été de modéliser les données modifiables et de garantir leur mise à jour cohérente dans la base de données. Par exemple, lorsqu'un utilisateur modifie le type d'un ticket via `comboNomType`, il fallait s'assurer que cette modification soit répercutée à la fois dans `tickets_filtres_seul` et dans `DataWarehouse_Tickets`, utilisée pour stocker les données du ticket sélectionné. Pour cela, j'ai conçu une logique en **VBA** qui compare les valeurs actuelles et nouvelles avant d'exécuter une requête `UPDATE`. J'ai également ajouté des vérifications pour éviter les erreurs, comme s'assurer qu'un ticket est bien sélectionné avant de permettre une modification.

Un autre défi a été de concevoir les combo boxes pour qu'elles soient à la fois intuitives et performantes. Par exemple, pour `comboNomType`, j'ai initialement utilisé une requête statique pour afficher "Incident" et "Demande", mais j'ai finalement opté pour une requête dynamique basée sur les données de `DataWarehouse_Tickets` :

```
Me!comboNomType.RowSource =
"SELECT DISTINCT type, nom_type FROM DataWarehouse_Tickets
WHERE nom_type IN ('Incident', 'Demande') ORDER BY nom_type;"
```

Cela garantissait que les options affichées étaient toujours cohérentes avec les données réelles de la base, tout en permettant à l'utilisateur de sélectionner un type (1 ou 2) qui serait utilisé dans les requêtes `UPDATE`.

Une subtilité dans cette conception de combo box interactive a été la gestion de la combo box de catégorie `comboNomCategorie`. Celle-ci est remplie à partir de la table `glpi_itilcategories`, car il est possible que `DataWarehouse_Tickets` ne contienne pas toutes les catégories disponibles dans **GLPI**. Pour résoudre ce problème, j'ai décidé d'importer la table `glpi_itilcategories` en local via le driver **ODBC**. À l'ouverture du formulaire, un script vérifie si cette table existe ; si elle est absente, elle est automatiquement importée.

2.2.4 Conception de la table de stockage

Pour pouvoir gérer les valeurs de retour de chaque requête **SQL**, j'ai décidé de créer une table de stockage, `TableStockageResultats`, qui stocke les résultats de chaque requête. Cette table est composée des champs suivants :

- `id_requete` : qui identifie de manière unique chaque requête stockée dans la table. Ce champ permet de différencier les résultats provenant de différentes requêtes (par exemple, une requête pour les tickets clos en moins de 4 heures, une autre pour les tickets par catégorie).
- `Mois` : représentant le mois pour lequel les données sont calculées (de 1 à 12). Ce champ peut être nul (`NULL`) si l'utilisateur souhaite des résultats annuels sans spécifier un mois particulier.
- `Annee` : indiquant l'année des données (par exemple, 2025). Ce champ est obligatoire pour toutes les requêtes, car les analyses sont toujours effectuées sur une base annuelle.
- `Entite` : qui stocke le nom de l'entité concernée par la requête (par exemple, "CASC" ou "Écoles"). Cela permet de filtrer les résultats par entité si nécessaire.

— `Resultats` : qui contient le résultat calculé par la requête, comme un pourcentage ou un nombre (par exemple, le pourcentage de tickets clos en moins de 4 heures). Ce champ est le cœur de la table, car il stocke la valeur principale à visualiser.

— `Libellé` : qui décrit la signification du résultat (par exemple, "Pourcentage de Tickets clos en moins de 4h"). Ce champ est utilisé pour afficher une description claire dans les graphiques ou les rapports.

La table `TableStockageResultats` joue un rôle clé dans la gestion des données pour les graphiques. Elle permet de centraliser les résultats des requêtes **SQL** complexes, qui sont souvent utilisées pour alimenter les visualisations dans le formulaire `Graphiques`. Par exemple, la requête suivante, utilisée pour calculer le pourcentage de tickets clos en moins de 4 heures pour l'entité CASC, insère ses résultats directement dans cette table :

```
PARAMETERS Mois Short, Annee Short;
 INSERT INTO TableStockageResultats
 (id_requete, Mois, Annee, Entite, Resultats, Libellé)
 SELECT 1 AS id_requete, IIf([Mois] Between 1 And 12,[Mois],Null) AS Mois,
     [Annee] AS Annee, 'CASC' AS Entite, Count(*)*100/
         (SELECT COUNT(*)
         FROM DataWarehouse_Tickets  WHERE entities_id = 1
         AND (Month(date_ouverture) = [Mois] OR [Mois] IS NULL)
         AND Year(date_ouverture) = [Annee]) AS Resultats,
 FROM DataWarehouse_Tickets
 WHERE (((Month([date_ouverture]))=[Mois])
 AND ((DataWarehouse_Tickets.entities_id)=1)
 AND ((DateDiff('h',[date_ouverture],[date_fermeture]))<=4)
 AND ((Year([date_ouverture]))=[Annee]))
 OR (((DataWarehouse_Tickets.entities_id)=1)
 AND ((DateDiff('h',[date_ouverture],[date_fermeture]))<=4)
 AND ((Year([date_ouverture]))=[Annee])
 AND (([Mois]) Is Null));
```

FIGURE 8 – Exemple d'une requête SQL qui récupère le pourcentage de tickets clos en moins de 4 heures pour l'entité CASC.

Dans cet exemple, la requête insère une ligne dans `TableStockageResultats` avec `id_requete = 1`, le mois et l'année spécifiés par l'utilisateur, `Entite = 'CASC'`, le pourcentage calculé dans `Resultats`, et un libellé descriptif. Ces données peuvent ensuite être extraites pour alimenter un graphique dans **Microsoft Access**, comme un diagramme en barres ou un graphique en secteurs.

Un défi lors de la conception de cette table a été de s'assurer que les champs soient suffisamment flexibles pour gérer différents types de requêtes. Par exemple, le champ `Mois` peut être nul pour des analyses annuelles, et le champ `Resultats` doit pouvoir stocker des valeurs

décimales pour les pourcentages. De plus, j'ai ajouté des index sur les champs `id_requete`, `Mois`, et `Annee` pour optimiser les performances lors de l'extraction des données pour les graphiques. Il était convenu d'utiliser uniquement le data warehouse pour ne pas avoir à traiter les tables et leurs données via le driver **ODBC**.

2.2.5 Conception des graphiques

Enfin, j'ai conçu un formulaire séparé, `Graphiques`, pour répondre au besoin de visualisation des données. Ce formulaire utilise des contrôles de type "Graphique" dans **Microsoft Access**, alimentés par des requêtes **SQL** spécifiques. Par exemple, pour afficher le pourcentage de tickets clos en moins de 4 heures pour une entité précise (dans le cas suivant CASC), j'ai conçu une requête comme suit :

```
PARAMETERS Mois Short, Annee Short;
INSERT INTO TableStockageResultats
( id_requete, Mois, Annee, Entite, Resultats, Libellé )
SELECT 1 AS id_requete, IIf([Mois] Between 1 And 12,[Mois],Null) AS Mois,
[Annee] AS Annee, 'CASC' AS Entite, Count()*100/
(SELECT COUNT(*)
    FROM DataWarehouse_Tickets  WHERE entities_id = 1
    AND (Month(date_ouverture) = [Mois] OR [Mois] IS NULL)
      AND Year(date_ouverture) = [Annee]) AS Resultats,
     'Pourcentage de Tickets clos en moins de 4h' AS Libellé
FROM DataWarehouse_Tickets
WHERE (((Month([date_ouverture]))=[Mois])
 AND ((DataWarehouse_Tickets.entities_id)=1)
 AND ((DateDiff('h',[date_ouverture],[date_fermeture]))<=4)
 AND ((Year([date_ouverture]))=[Annee]))
 OR (((DataWarehouse_Tickets.entities_id)=1)
 AND ((DateDiff('h',[date_ouverture],[date_fermeture]))<=4)
 AND ((Year([date_ouverture]))=[Annee])
 AND (([Mois]) Is Null));
```

Lors de la conception des graphiques, j'ai dû prendre en compte les performances, car certaines requêtes pouvaient être lentes sur de grandes quantités de données. Pour optimiser cela, j'ai prévu d'ajouter des index sur les champs fréquemment utilisés dans les requêtes (comme `date` et `date_close`) dans `DataWarehouse_Tickets`. J'ai également conçu les graphiques pour qu'ils soient modulables, permettant à l'utilisateur de choisir les métriques à afficher (par exemple, par mois, année, ou catégorie) via des filtres sur le formulaire.

Une fois le bouton **Lancer les graphiques** cliqué, le code **VBA** associé exécute la requête **SQL** propre à chaque graphique (chaque graphique est nommé de manière unique pour pouvoir les différencier facilement dans le code **VBA**). Par exemple, pour le graphique des tickets clos en moins de 4 heures, le code exécute la requête **SQL** correspondante et affiche les résultats dans un graphique à barres.

2.2.6 Conception des exports vers Excel

Un des besoins exprimés par l'équipe informatique était la possibilité d'exporter les données vers Excel pour faciliter les analyses externes. Pour répondre à ce besoin, j'ai conçu une fonctionnalité d'exportation dans le formulaire principal, `Formulaire`, accessible via un bouton dédié nommé `btnExportGeneral`. Cette fonctionnalité permet de générer un fichier Excel contenant les données de la table `DataWarehouse_Tickets` et de le sauvegarder directement sur le Bureau de l'utilisateur.

Le défi principal de cette fonctionnalité était de s'assurer que l'exportation fonctionne sur n'importe quelle machine, indépendamment du chemin d'accès au Bureau de l'utilisateur. Pour cela, j'ai utilisé la fonction `Environ("USERPROFILE")` en **VBA** pour récupérer dynamiquement le chemin du profil utilisateur, auquel j'ai ajouté le sous-répertoire `\Desktop` pour cibler le Bureau. Le fichier exporté est nommé `DataWarehouse_Tickets.xlsx` pour refléter son contenu.

Le code **VBA** suivant, associé à l'événement `Click` du bouton `btnExportGeneral`, implémente cette fonctionnalité :

J'ai utilisé de stocker tout les exports excel dans le bureau de l'utilisateur pour éviter de devoir changer le chemin d'accès à chaque fois.

Ce code commence par déclarer une variable `filePath` pour stocker le chemin du fichier Excel à générer. La fonction `Environ("USERPROFILE")` est utilisée pour construire ce chemin en ajoutant `\Desktop\DataWarehouse_Tickets.xlsx`, ce qui garantit que le fichier sera

```
Private Sub btnExportGeneral_Click()
On Error GoTo ErrorHandler

    Dim filePath As String

    filePath = Environ("USERPROFILE") & "\Desktop\DataWarehouse_Tickets.xlsx"
    DoCmd.TransferSpreadsheet acExport, acSpreadsheetTypeExcel12Xml, "DataWarehouse_Tickets", filePath, True
    MsgBox "Les données ont été exportées avec succès vers " & filePath
    Exit Sub

ErrorHandler:
    MsgBox "Une erreur s'est produite lors de l'exportation des données : " & Err.Description
End Sub
```

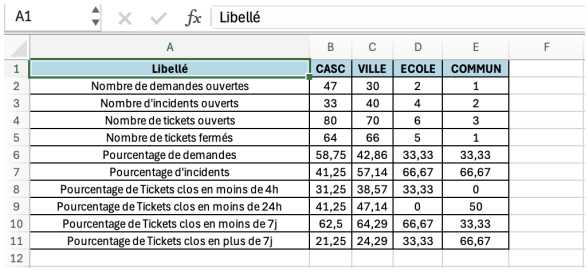
FIGURE 9 – Partie du code VBA gérant l'exportation des données vers Excel.

sauvegardé sur le Bureau de l'utilisateur, quel que soit son environnement. Ensuite, la méthode `DoCmd.TransferSpreadsheet` est appelée pour exporter la table `DataWarehouse_Tickets` vers un fichier Excel au format `.xlsx` (spécifié par `acSpreadsheetTypeExcel12Xml`). L'argument `True` indique que les noms des champs de la table doivent être inclus comme en-têtes dans le fichier Excel, facilitant ainsi son utilisation pour des analyses externes.

Pour améliorer l'expérience utilisateur, j'ai ajouté un message de confirmation via `MsgBox`, qui affiche le chemin complet du fichier exporté une fois l'opération terminée. Enfin, une gestion des erreurs est implémentée avec un bloc `On Error GoTo ErrorHandler`, qui affiche un message d'erreur détaillé en cas de problème (par exemple, si le Bureau est inaccessible ou si Excel n'est pas installé).

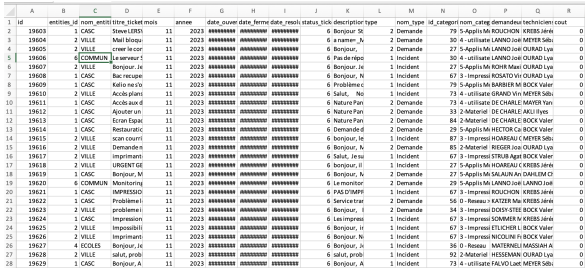
Cette approche simple et efficace répond aux besoins de l'équipe informatique tout en minimisant les risques d'erreurs liés à des chemins d'accès fixes. Cependant, une limitation potentielle est que le fichier est systématiquement nommé `DataWarehouse_Tickets.xlsx`, ce qui pourrait écraser un fichier existant portant le même nom. Une évolution future pourrait consister à permettre à l'utilisateur de choisir le nom et l'emplacement du fichier via une boîte de dialogue.

Il était aussi possible de moduler l'export vers Excel en fonction de ce qu'on exporte, si nous décidons d'exporter les resultats des statistiques, alors l'excel n'aura pas la même structure qu'une simple exportation. Voici la différence entre l'export des statistiques et l'export de la table `DataWarehouse_Tickets` :



Libellé	CASC	VILLE	ECOLE	COMMUN
Nombre de demandes ouvertes	47	30	2	1
Nombre d'incidents ouverts	33	40	4	2
Nombre de tickets ouverts	80	70	6	3
Nombre de tickets fermés	64	66	5	1
Pourcentage de demandes	58,75	42,86	33,33	33,33
Pourcentage d'incidents	41,25	57,14	66,67	66,67
Pourcentage de Tickets clos en moins de 4h	31,25	38,57	33,33	0
Pourcentage de Tickets clos en moins de 24h	41,25	47,14	0	50
Pourcentage de Tickets clos en moins de 7j	62,5	64,29	66,67	33,33
Pourcentage de Tickets clos en plus de 7j	21,25	24,29	33,33	66,67

(a) Extrait de l'export des statistqiues, avec une mise en forme.



(b) Extraut de l'export du DataWarehouse, sans mise en forme.

FIGURE 10 – Comparaison entre l'export des statistiques et l'export de la table `DataWarehouse_Tickets`.

La première exportation est celle des statistiques, elle est plus complexe que l'exportation de la table `DataWarehouse_Tickets` car elle contient des données plus complexes et elle est mise en forme pour être plus lisible. L'export de la table `DataWarehouse_Tickets` est plus simple et brut, il n'a pas de mise en forme particulière et utilise simplement l'instance de `DoCmd.TransferSpreadsheet` pour exporter les données.

2.2.7 Gestion d'ouverture du formulaire et connection ODBC

Lorsqu'un utilisateur ouvre le formulaire principal du tableau de bord, il est essentiel de s'assurer que toutes les ressources nécessaires sont disponibles pour un fonctionnement optimal de l'application. Pour cela, j'ai implémenté une sous-procédure `VBA` dans l'événement `Form_Open`, qui orchestre deux étapes clés : l'établissement de la connexion à la base de données **GLPI** via un driver **ODBC** et la vérification de la présence de la table `glpi_itilcategories`, avec son importation si nécessaire.

```
Private Sub Form_Open(Cancel As Integer)
    Call RegDB_mysql
    If Not TableExists("Local_glpi_categories") Then
        Call ImportGlpiItilCategories
    Else
        Debug.Print "LA table existe deja"

    End If

End Sub
```

FIGURE 11 – Extrait de la fonction d'initialisation du formulaire, appelant `RegDB_mysql` pour établir la connexion **ODBC** et vérifiant l'existence de la table `glpi_itilcategories`.

La connexion **ODBC** est établie par la sous-procédure `RegDB_mysql`, illustrée ci-dessous. Cette fonction configure une source de données **ODBC** nommée `ESSAI_MySQL` en utilisant le pilote `MySQL ODBC 9.2 ANSI Driver`. Elle définit les paramètres de connexion, ouvre une connexion à la base **MariaDB** de **GLPI** à l'aide de l'objet `ADODB.Connection`, et exécute une requête de test pour vérifier la validité de la connexion. Une gestion rigoureuse des erreurs est intégrée pour afficher un message en cas de problème (par exemple, serveur inaccessible) et fermer proprement la connexion, garantissant ainsi la robustesse de l'initialisation. La connexion établie est stockée dans une variable globale `gConn`, réutilisable pour d'autres opérations, comme l'importation de tables ou l'exécution de requêtes **SQL**.

Une fois la connexion établie, la fonction `TableExists` vérifie si la table `glpi_itilcategories` est présente dans la base **Microsoft Access**. Cette table, contenant les catégories des tickets, est cruciale pour remplir dynamiquement la combo box `comboNomCategorie` du formulaire interactif. Si elle est absente, la sous-procédure `ImportGlpiItilCategories` importe les données depuis **GLPI** via le driver **ODBC**, en utilisant une requête **SQL** pour intégrer les catégories dans une table locale. Cela garantit que toutes les catégories disponibles sont accessibles, même si elles ne figurent pas encore dans `DataWarehouse_Tickets`.

```
Sub RegDB_mysql()
    On Error GoTo ErrHandler

    Dim strDSN As String, strAttr As String, strODBCDrv As String
    Dim rs As Object
    Dim strConn As String
    Dim strSQL As String
    Dim databaseList As String
    Dim connectionSuccessful As Boolean

    connectionSuccessful = False

    strDSN = "ESSAI_MySQL"
    strODBCDrv = "MySQL ODBC 9.2 ANSI Driver"
    strAttr = "SERVER=128.128.128.0" & vbCr & _
              "DATABASE=nom_dataBase" & vbCr & _
              "PORT=3306" & vbCr & _
              "OPTION=579" & vbCr & _
              "UID=GLPI" & vbCr & _
              "PWD={mdpmdp}" & vbCr & _
              "CHARSET=utf16"

    DBEngine.RegisterDatabase strDSN, strODBCDrv, True, strAttr
    If Not gConn Is Nothing Then
        If gConn.State = 1 Then gConn.Close
        Set gConn = Nothing
    End If
    Set gConn = CreateObject("ADODB.Connection")
    strConn = "DSN=" & strDSN & ";"
    gConn.Open strConn
    strSQL = "SHOW DATABASES;"
    Set rs = CreateObject("ADODB.Recordset")
    rs.Open strSQL, gConn, 0, 1

    connectionSuccessful = True

    rs.Close
    Set rs = Nothing
    Exit Sub

ErrHandler:
    MsgBox "Erreur " & Err.Number & " : " & Err.Description, vbCritical, "Erreur"
    If Not rs Is Nothing Then
        If rs.State = 1 Then rs.Close
        Set rs = Nothing
    End If
    If Not gConn Is Nothing Then
        If gConn.State = 1 Then gConn.Close
        Set gConn = Nothing
    End If
    If Not connectionSuccessful Then
        Debug.Print "Échec de la connexion à MySQL."
    End If
    Exit Sub
End Sub
```

FIGURE 12 – Code de la sous-procédure `RegDB_mysql`, configurant et ouvrant la connexion **ODBC** à la base de données **GLPI**.

Cette initialisation est fondamentale pour le bon fonctionnement du tableau de bord. La connexion **ODBC** permet un accès en temps réel aux données de **GLPI**, tandis que la vérification et l'importation conditionnelle de `glpi_itilcategories` assurent que le formulaire est immédiatement opérationnel. Une limitation est que les paramètres de connexion dans `RegDB_mysql` sont codés en dur, ce qui peut compliquer la maintenance si l'infrastructure change. Une amélioration future pourrait externaliser ces paramètres dans un fichier de configuration pour plus de flexibilité.

En résumé, la gestion de l'ouverture du formulaire repose sur une initialisation robuste combinant une connexion **ODBC** fiable et une préparation des ressources locales. Cette approche garantit une expérience utilisateur fluide et une application fonctionnelle dès son lancement.

2.2.8 Choix techniques et contraintes

La conception a été guidée par plusieurs contraintes techniques et temporelles. Tout d'abord, le choix de **Microsoft Access** comme outil principal a influencé la manière dont le data warehouse et le formulaire ont été conçus. Access, dans sa version 2019 utilisée par la CASC, offre des fonctionnalités intégrées pour la gestion des bases de données et la création de formulaires, ce qui en faisait un choix adapté à la familiarité de l'équipe informatique avec cet outil. Cependant, il présente des limitations, notamment pour les requêtes complexes impliquant plusieurs tables, comme celles nécessaires pour extraire et transformer les données de **GLPI**. C'est pourquoi j'ai utilisé **VBA** pour écrire certaines requêtes **SQL** personnalisées (comme la création du data warehouse), ce qui m'a permis de contourner ces limitations et d'automatiser des tâches comme la centralisation des données dans `DataWarehouse_Tickets`.

Une autre contrainte était le temps limité du stage (8 semaines). Cela m'a conduit à prioriser la conception des fonctionnalités essentielles, comme la centralisation des données dans le data warehouse et la modification des tickets via le formulaire, tout en laissant des évolutions possibles pour l'avenir. Par exemple, l'ajout de filtres dynamiques plus avancés, comme une analyse prédictive du volume de tickets par mois, ou l'intégration de nouvelles métriques dans les graphiques, comme le temps moyen de résolution par technicien, ont été envisagés de ma part, mais reportés ou annulés à une phase ultérieure en raison des contraintes temporelles.

Les compétences de l'équipe informatique de la CASC ont également influencé les choix techniques. La personne utilisant le tableau de bord étant plus à l'aise avec **Microsoft Access** qu'avec des technologies comme PHP ou des frameworks web, il était essentiel de concevoir une solution qui puisse être maintenue et adaptée par eux sans nécessiter une formation approfondie sur de nouveaux outils. Cela a renforcé la pertinence du choix d'Access, malgré ses limitations, et m'a conduit à privilégier des solutions simples et bien documentées.

Enfin, j'ai dû m'assurer que la conception était suffisamment documentée pour permettre à l'équipe informatique de la CASC de maintenir et faire évoluer le système après mon départ. Pour cela, j'ai inclus des commentaires détaillés dans le code **VBA**, expliquant la logique de chaque fonction et procédure, et j'ai fourni un diagramme de classe (voir Figure 13) pour illustrer la structure du data warehouse et les relations entre les tables. De plus, j'ai organisé une démonstration finale a mon maitre de stage pour lui présenter le fonctionnement du tableau de bord et répondre à ses questions, facilitant ainsi le transfert de connaissances.

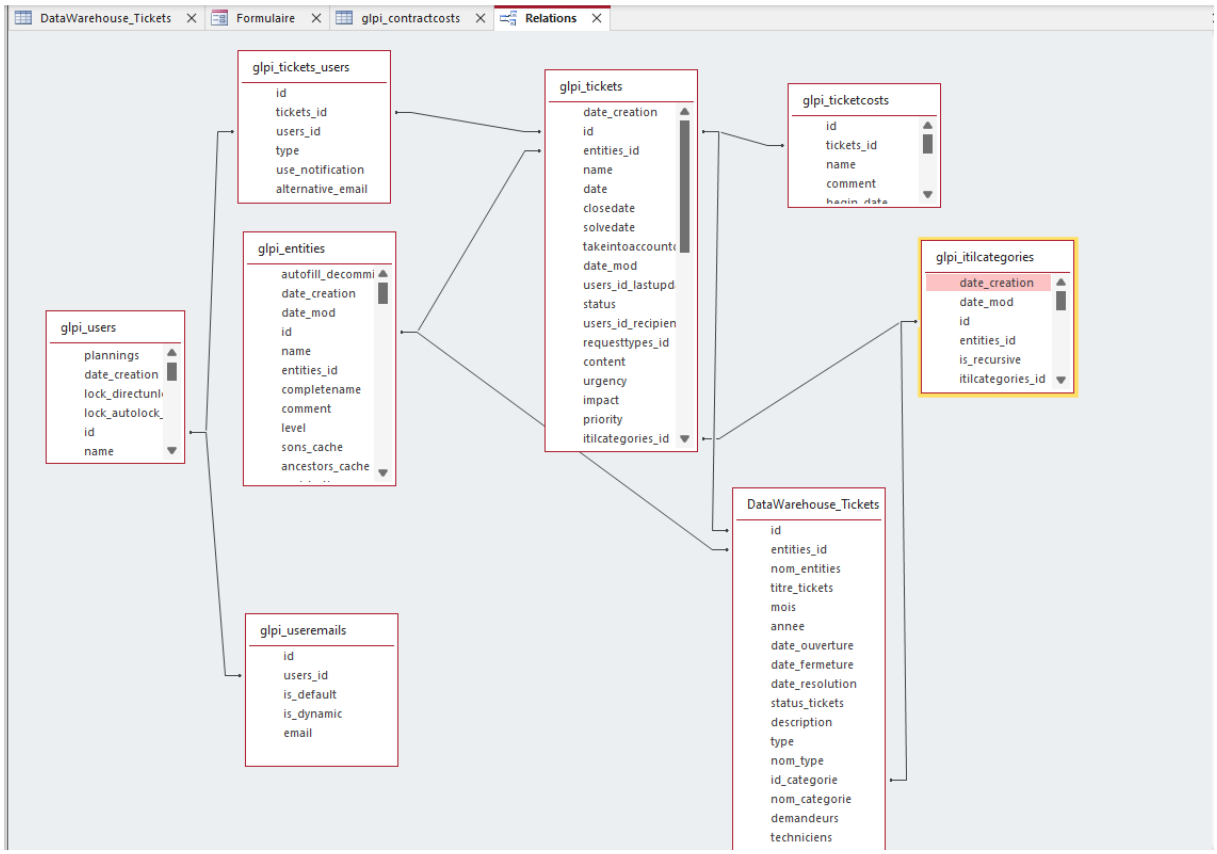


FIGURE 13 – Diagramme de classe de la base de données, illustrant les relations entre les tables du data warehouse.

Ce diagramme de classe illustre la structure du data warehouse, mettant en évidence les relations entre les différentes tables. Par exemple, la table `DataWarehouse_Tickets` est au centre, reliant les tickets aux entités et catégories via des clés étrangères. Cela permet de visualiser rapidement comment les données sont organisées et comment elles interagissent entre elles, facilitant ainsi la compréhension du système pour l'équipe informatique de la CASC. J'ai utilisé l'onglet **Relation** de **Microsoft Access** pour créer ce diagramme de classe, en ajoutant les liens entre chaque table manuellement.

2.3 Difficultés rencontrées

Lors du développement du tableau de bord, j'ai dû faire face à plusieurs difficultés qui ont ralenti mon travail, mais m'ont appris à contourner les problèmes et à trouver des solutions.

2.3.1 Affichage des balises et caractères spéciaux dans les données GLPI

Un problème notable lors du développement a été la présence de balises dans les descriptions des tickets extraites de **GLPI**. Par exemple, un champ pouvait contenir du texte comme `&#60;p&#62 Problème réseau &#60;/p&#62`, rendant les données illisibles dans le tableau de bord. Pour résoudre ce problème, j'ai créé une fonction **VBA** nommée `CleanText`, qui supprime les balises et conserve uniquement le texte brut. Cette fonction parcourt la chaîne de caractères, identifie les balises (délimitées par `&#60;p&#62` et `&#60;/p&#62`), et les remplace par des espaces. Voici un extrait du code de la fonction `CleanText` :

```
Private Function CleanText(text As String) As String

    If text = "" Or text = "Aucune description" Then
        CleanText = text
        Exit Function
    End If

    text = Replace(text, "&#60;p&#62;", " ", , , vbTextCompare)
    text = Replace(text, "&#60;/p&#62;", " ", , , vbTextCompare)
    text = Replace(text, "&#60;p class=", " ", , , vbTextCompare)
    text = Replace(text, "&#62;", " ", , , vbTextCompare)
    text = Replace(text, "&#38;nb", " ", , , vbTextCompare)
    text = Replace(text, "sp;", " ", , , vbTextCompare)
    text = Replace(text, "&#60;br", " ", , , vbTextCompare)
    text = Replace(text, "&#60;strong", " ", , , vbTextCompare)
    text = Replace(text, "&#60;/strong", " ", , , vbTextCompare)
    text = Replace(text, "&#60;span style=", " ", , , vbTextCompare)
    text = Replace(text, "&#60;/span", " ", , , vbTextCompare)
    text = Replace(text, "&#60;span", " ", , , vbTextCompare)
```

FIGURE 14 – Extrait de la fonction `CleanText` pour supprimer les balises HTML.

J'ai déduit que les balises se trouvant autour des zone de texte étaient des balises HTML car nous retrouvons le symbole **p** dans une balise d'ouverture et dans une balise de fermeture. Ce qui mène a croire que ce sont des balises HTML. Je me suis alors demandé si il n'y avait pas une fonction de filtrage interne à Microsoft Access qui permettrait de filtrer les balises HTML. J'ai alors cherché sur le site de Microsoft ainsi que sur StackOverFlow et j'ai trouvé certaines réponses mais aucunes fonctionnait réellement dans mon code. J'ai alors décidé de créer manuellement cette fonction en repérant les balises qui se répétaient dans les champs de texte.

Un problème similaire est apparu avec des caractères spéciaux (par exemple, des caractères non-ASCII comme à ou é) dans les champs titre, description, et demandeurs/techniciens. Ces caractères provoquaient des erreurs lors des modifications via le formulaire interactif et affectaient la lisibilité des données affichées. Pour résoudre ce problème, j'ai créé une fonction **VBA** distincte, `Remplacer_caracteres_specials`, en m'inspirant de `CleanText`. Cette fonction remplace les caractères problématiques par leurs équivalents ASCII ou les supprime. J'ai choisi de séparer ces deux fonctions pour faciliter le débogage : en cas d'erreur, il était plus simple d'identifier si le problème venait du traitement des balises HTML ou des caractères spéciaux. Voici un extrait du code de `Remplacer_caracteres_specials` :

```
Function Remplacer_caracteres_specials(ByVal strText As Variant) As String
    On Error Resume Next

    Dim result As String

    If IsNull(strText) Then
        Remplacer_caracteres_specials = " "
        Exit Function
    End If

    result = strText
    result = Replace(result, "Ã©", "é") ' é
    result = Replace(result, "Ã¨", "è") ' è
    result = Replace(result, "Ãª", "ê") ' ê
    result = Replace(result, "Ã«", "ë") ' ë
    result = Replace(result, "Ã¢", "â") ' â
    result = Replace(result, "Ã¤", "ä") ' ä
    result = Replace(result, "Ã¹", "ù") ' ù
    result = Replace(result, "Ã»", "û") ' û
    result = Replace(result, "Ã¼", "ü") ' ü
    result = Replace(result, "Ã®", "î") ' î
    result = Replace(result, "Ã¯", "ï") ' ï
    result = Replace(result, "Ã´", "ô") ' ô
    result = Replace(result, "Ã¶", "ö") ' ö
    result = Replace(result, "Ã§", "ç") ' ç
    result = Replace(result, "Ã€", "À") ' À
    result = Replace(result, "Ã‰", "É") ' É
    result = Replace(result, "ÃŠ", "Ê") ' Ê
    result = Replace(result, "Ã‹", "Ë") ' Ë
    result = Replace(result, "©", "é")
    result = Replace(result, "Ã©", "é")

    result = Replace(result, "Ã", "")
    result = Replace(result, "Â", "")
    result = Trim(result)
    While InStr(result, "  ") > 0
        result = Replace(result, "  ", " ")
    Wend
    If result = "" Then
        Remplacer_caracteres_specials = "Inconnu"
    Else
        Remplacer_caracteres_specials = result
    End If
End Function
```

FIGURE 15 – Extrait de la fonction `Remplacer_caracteres_specials` pour gérer les caractères spéciaux.

La fonction `Remplacer_caracteres_specials` est appliquée aux champs titre, description, et demandeurs/techniciens lors de leur traitement pour l'affichage dans le formulaire interactif, ainsi que lors des mises à jour effectuées via les combo boxes. Par exemple, lorsqu'un utilisateur modifie un champ via le formulaire, la fonction est appelée pour s'assurer

que les caractères spéciaux sont correctement gérés avant d'exécuter la requête `UPDATE` dans `DataWarehouse_Tickets`. Cette approche a permis de prévenir les erreurs de compatibilité et d'améliorer la lisibilité des données pour les utilisateurs. Pour illustrer, un titre de ticket contenant `Problème d'accès réseau & imprimante` serait transformé en `Probleme d'acces reseau et imprimante` après passage par `Remplacer_caracteres_specials`. Cette transformation garantit que les données sont exploitables dans le tableau de bord et conformes aux attentes des utilisateurs. Cette solution a été testée avec plusieurs ensembles de données **GLPI**, et les résultats ont montré une réduction significative des erreurs lors des modifications, ainsi qu'une amélioration de la clarté des informations affichées. Une évolution future pourrait consister à fusionner `CleanText` et `Remplacer_caracteres_specials` en une seule fonction pour simplifier la maintenance, tout en conservant une gestion modulaire des différents types de caractères problématiques.

2.3.2 Gestion des incohérences dans les données GLPI

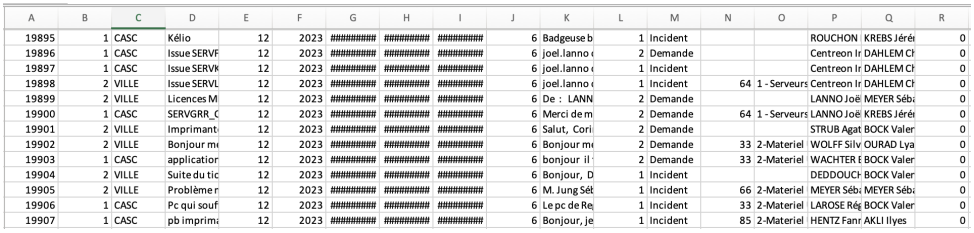
Un autre défi a été la présence d'incohérences dans les données **GLPI**, comme des tickets sans catégorie associée, ce qui générait des valeurs nulles dans les analyses. Pour résoudre ce problème, j'ai utilisé la fonction `Nz` dans les requêtes **SQL** pour remplacer les valeurs nulles par des valeurs par défaut (par exemple, "" pour `nom_categorie`). Cela a permis de garantir la cohérence des données lors des calculs de pourcentages ou des visualisations dans les graphiques.

La fonction NZ a été particulièrement utile dans l'insertion des champs du DataWarehouse. J'ai rencontré certaines incohérences dans les champs de catégories car les tickets n'ayant aucunes catégories ne s'inséraient pas dans le DataWarehouse. J'ai donc utilisé la fonction `Nz` pour remplacer les valeurs nulles par une chaîne vide, ce qui a permis de garantir que toutes les colonnes du data warehousecontiennent des valeurs valides. Par exemple, un ticket sans catégorie serait inséré avec une valeur vide dans le champ `id_categorie` `nom_categorie`, évitant ainsi des erreurs lors des analyses ultérieures. Voici un extrait de code illustrant l'utilisation de `Nz` dans une requête d'insertion pour les catégories :

```
sql = "SELECT id AS id_categorie, completename AS nom_categorie FROM glpi_itilcategories WHERE id = " & rsTickets!itilcategories_id
Set rsCategorie = CreateObject("ADODB.Recordset")
rsCategorie.Open sql, gConn, 0, 1
If Not rsCategorie.EOF Then
    rsDW!id_categorie = rsCategorie!id_categorie
    Dim categoryName As String
    categoryName = Remplacer_caracteres_specials(Nz(rsCategorie!nom_categorie, " "))
    rsDW!nom_categorie = IIf(categoryName = "", " ", categoryName)
Else
    rsDW!nom_categorie = " "
End If
rsCategorie.Close
Set rsCategorie = Nothing
```

FIGURE 16 – Extrait de la fonction d'insertions de valeurs dans le DataWarehouse, utilisant `Nz` pour gérer les valeurs nulles dans les catégories.

Comme vous pouvez le voir dans la partie du code ci-dessus, la fonction `Nz` est utilisée pour remplacer les valeurs nulles dans le champ `nom_categorie` par une chaîne vide (" "). Cela garantit que toutes les lignes insérées dans le data warehousecontiennent des valeurs valides, même si certaines catégories sont manquantes dans les données d'origine. Cette approche a permis de réduire les erreurs lors des analyses et d'améliorer la qualité globale des données dans le tableau de bord.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19895		1	CASC	Kélio	12	2023	##########	##########	##########	6	Badgeuse b	1	Incident			ROUCHON	KREBS Jéré	0
19896		1	CASC	Issue SERVF	12	2023	##########	##########	##########	6	joel.lanno	2	Demande			Centreon Ir	DAHLEM Ch	0
19897		1	CASC	Issue SERVI	12	2023	##########	##########	##########	6	joel.lanno	1	Incident			Centreon Ir	DAHLEM Ch	0
19898		2	VILLE	Issue SERVL	12	2023	##########	##########	##########	6	joel.lanno	1	Incident	64	1 - Serveurs	Centreon Ir	DAHLEM Ch	0
19899		2	VILLE	Licences M	12	2023	##########	##########	##########	6	De : LANN	2	Demande			LANNO Joë	MEYER Séba	0
19900		1	CASC	SERVGRR_C	12	2023	##########	##########	##########	6	Merci de m	2	Demande	64	1 - Serveurs	LANNO Joë	KREBS Jéré	0
19901		2	VILLE	Imprimanti	12	2023	##########	##########	##########	6	Salut, Cori	2	Demande			STRUB Agat	BOCK Valer	0
19902		2	VILLE	Bonjour m	12	2023	##########	##########	##########	6	Bonjour m	2	Demande	33	2-Materiel	WOLFF Silv	OURAD Lya	0
19903		1	CASC	applicatior	12	2023	##########	##########	##########	6	bonjour il	2	Demande	33	2-Materiel	WACHTER E	BOCK Valer	0
19904		2	VILLE	Suite du tic	12	2023	##########	##########	##########	6	Bonjour, D	1	Incident			DEDDOUCH	BOCK Valer	0
19905		2	VILLE	Problème r	12	2023	##########	##########	##########	6	M. Jung Séb	1	Incident	66	2-Materiel	MEYER Séba	MEYER Séba	0
19906		1	CASC	Pc qui souf	12	2023	##########	##########	##########	6	Le pc de Re	1	Incident	33	2-Materiel	LAROSE Rég	BOCK Valer	0
19907		1	CASC	pb imprima	12	2023	##########	##########	##########	6	Bonjour, je	1	Incident	85	2-Materiel	HENTZ Fanr	AKLI Ilyes	0

FIGURE 17 – Extrait du data warehouseavec des valeurs nulles dans les champs `nom_categorie` et `id_categories`.

2.3.3 Performances des requêtes SQL complexes

Les requêtes **SQL** utilisées pour calculer les métriques (comme le pourcentage de tickets clos en moins de 4 heures) étaient parfois lentes, surtout avec un grand volume de données. Pour optimiser les performances, j'ai ajouté des index sur les champs fréquemment utilisés dans les requêtes, comme `date` et `date_close` dans `DataWarehouse_Tickets`, ainsi que `id_requete`, `Mois`, et `Annee` dans `TableStockageResultats`. J'ai également simplifié certaines requêtes en évitant des sous-requêtes inutiles lorsque cela était possible, ce qui a réduit le temps d'exécution et amélioré l'expérience utilisateur dans le tableau de bord.

Les requêtes qui étaient les plus lentes étaient celles qui nécessitaient des insertions dans des tables, comme les insertions dans un data warehouseou les insertions de statistiques dans la table de stockage de resultats. En essayant de résoudre ce problème, j'ai alors remarqué que faire des requêtes SQL simple pour gerer les insertions étaient beaucoup trop couteux en ressoures. Je suis alors passé par des requêtes **VBA** pour gérer les insertions dans le data warehouseet la table de stockage de résultats. Cela a permis de réduire le temps d'exécution des requêtes et d'améliorer les performances globales du tableau de bord. Grâce aux requêtes VBA, je suis passé de 5 minutes pour insérer toutes les valeurs d'une année (environs 3372 lignes pour une douzaine de colonnes) à moins de 2 minutes pour la même année. J'ai aussi remarqué que les requêtes **VBA** étaient plus simples à lire et à comprendre que les requêtes **SQL** complexes, ce qui a facilité la maintenance du code et la création d'autres code similaire pour les insertions.

2.3.4 Limitation de Microsoft Access pour les interfaces complexes

Microsoft Access a montré des limites dans la gestion des interfaces utilisateur complexes, notamment pour les formulaires interactifs. Par exemple, le rafraîchissement automatique du sous-formulaire après une modification via une combo box ne fonctionnait pas toujours comme prévu. Pour contourner ce problème, j'ai ajouté des appels explicites à la méthode `Requery` en **VBA** (par exemple, `Me!subformTickets.Form.Requery`) pour forcer le rafraîchissement des données affichées, garantissant ainsi une mise à jour cohérente de l'interface après chaque modification.

Ces difficultés, bien qu'elles aient représenté des obstacles, m'ont permis de développer une approche plus rigoureuse dans la gestion de projet et de mieux comprendre les outils et technologies utilisés. Elles ont également renforcé ma capacité à anticiper les problèmes potentiels dans des projets futurs.

Les limitations se sont aussi fait ressentir lors de la création du formulaire regroupant tout les graphiques. Comme les graphiques semblaient être couteux en ressources, Microsoft Access redémarrait à chaque création de nouveau graphique. Je me suis alors questionné sur la structure de mon code et demandé si l'erreur ne venait pas de la manière dont j'avais codé les graphiques précédents. Je n'ai pas trouvé d'erreur à ce niveau-là. J'ai donc du être patient et créer chaque graphique indépendamment.

Lors de la mise en production, j'avais réfléchie à un formulaire de navigation qui permettrait de naviguer entre les différents formulaires(Formulaire principal et formulaire de Graphiques). Mais lors de la mise en place du formulaire de navigation (propre à Microsoft Access), le sous formulaire qui permettait de visualiser les tickets ainsi que de les modifier, ne fonctionnait pas. Il était impossible de le manipuler. J'en ai déduit que l'erreur provenait du fait que j'ai imbriqué le sous formulaire dans le formulaire principal qui est lui même dans le formulaire de navigation. J'ai alors contourné le problème en créant un bouton de navigation qui permet d'accéder au formulaire de graphiques. Ce bouton a une particularité, il est possible d'accéder au formulaire de graphiques uniquement si la table de stockage de résultats est remplie et générée. Cela permet d'éviter une erreur lors du lancement des graphiques interactifs. J'ai aussi ajouté un message d'erreur si l'utilisateur essaye d'accéder au formulaire de graphiques sans que la table de stockage de résultats soit remplie. Cela permet d'éviter des erreurs lors de l'utilisation du tableau de bord et d'améliorer l'expérience utilisateur.

2.3.5 Gestion des erreurs

Pour assurer la robustesse du tableau de bord, j'ai implémenté une gestion rigoureuse des erreurs dans le code **VBA**. Des blocs `On Error GoTo` ont été intégrés pour capturer et traiter les erreurs potentielles, notamment lors de l'exécution des requêtes **SQL** ou des connexions à la base de données via le driver **ODBC**. En cas d'erreur, des messages détaillés sont affichés à l'utilisateur via des `MsgBox`, précisant la nature du problème (par exemple, une connexion au serveur échouée ou une syntaxe de requête incorrecte). Parallèlement, des instructions `Debug.Print` ont été utilisées pour journaliser les erreurs ainsi que les actions des fonctions dans la console de débogage de **Microsoft Access**, facilitant ainsi l'analyse des incidents lors du développement.

En complément, j'ai exploité le débogueur intégré de **Microsoft Access** pour inspecter les valeurs des variables en temps réel et identifier l'origine des erreurs. Cette approche combinée a permis de réduire le temps de résolution des problèmes, d'améliorer la fiabilité de l'application et d'offrir une expérience utilisateur plus fluide grâce à des retours d'erreur clairs et exploitables.

3 Conclusion

Ce stage, réalisé du 17 février au 11 avril 2025 au sein de la **Communauté d'Agglomération Sarreguemines Confluences (CASC)**, a été une expérience particulièrement enrichissante, tant sur le plan technique que personnel.

Sur le plan technique, ce stage m'a permis de consolider et d'approfondir mes compétences dans plusieurs domaines clés de l'informatique. J'ai acquis une maîtrise avancée de **SQL** pour la manipulation et l'analyse des données, notamment à travers la création de requêtes complexes pour extraire des métriques précises. La conception du data warehouse m'a offert une compréhension approfondie de la gestion des bases de données relationnelles, en particulier dans un contexte où les données sont dispersées et nécessitent un traitement rigoureux. De plus, le développement en **VBA** m'a permis d'automatiser des processus et de créer des interfaces interactives, renforçant ainsi mes compétences en programmation. La gestion des incohérences dans les données **GLPI**, comme les valeurs nulles ou les caractères spéciaux, m'a appris à adopter une méthodologie systématique pour nettoyer et transformer les données, tout en optimisant les performances des requêtes. Enfin, l'utilisation de **Microsoft Access** dans un cadre professionnel m'a donné une perspective pratique sur les interactions entre bases de données et interfaces utilisateur.

Sur le plan personnel, ce stage a marqué une étape importante dans mon développement. Intégré au sein d'une équipe de 11 membres, sous la supervision de **Monsieur Christophe Dahlem**, j'ai gagné en autonomie et en confiance dans un environnement professionnel. La planification de mes tâches à l'aide d'un diagramme de Gantt m'a appris à gérer mon temps efficacement, tout en respectant des délais serrés. Collaborer avec mes collègues m'a sensibilisé à l'importance d'une communication claire et d'un travail d'équipe, des compétences essentielles pour ma future carrière en informatique. Cette expérience m'a également permis de mieux comprendre les attentes d'un service informatique dans une collectivité, renforçant mon intérêt pour les projets mêlant gestion de données et outils décisionnels.

En conclusion, ce stage a été une opportunité précieuse pour mettre en pratique mes connaissances académiques tout en développant de nouvelles compétences techniques et personnelles. Les défis rencontrés, comme la gestion des limitations de **Microsoft Access** ou l'optimisation des performances, m'ont préparé à aborder des projets futurs avec plus de rigueur et de confiance.

4 Glossaire

Vous trouverez ci-dessous les définitions des termes techniques utiles à la compréhension du rapport de stage. Certains de ces termes sont utilisés dans le rapport.

CASC : Communauté d'Agglomération Sarreguemines Confluences.

Catégorie : Classification des tickets en fonction de leur nature ou de leur objet.

Clé étrangère : Champ qui établit une relation entre deux tables en référençant la clé primaire d'une autre table.

Clé primaire : Champ qui identifie de manière unique chaque enregistrement dans une table.

Combo Box : Contrôle graphique qui permet de sélectionner une valeur dans une liste déroulante.

Data warehouse : Entrepôt de données qui stocke et gère les données provenant de différentes sources pour faciliter l'analyse et la prise de décision.

Diagramme de cas d'utilisation : Diagramme qui représente les interactions entre les acteurs et le système.

Diagramme de classe : Diagramme qui représente la structure statique d'un système en montrant les classes et les relations entre elles.

Entité : Dans le cas du tableau de bord, peut être une école, un service ou une direction.

EPCI : Établissement public de coopération intercommunale.

Fonction : Bloc de code en `VBA` qui effectue une tâche spécifique et renvoie un résultat.

Formulaire : Interface utilisateur qui permet de saisir et de visualiser des données dans une base de données.

Gantt : Diagramme qui représente les tâches d'un projet et leur planification dans le temps.

GLPI : Application open-source de gestion des services informatiques et de gestion des actifs.

Graphique : Représentation visuelle des données sous forme de diagramme ou de tableau.

Jointure : Opération `SQL` qui permet de combiner les données de deux tables en fonction d'une condition.

Macro : Ensemble d'instructions qui permettent d'automatiser des tâches dans une application.

Microsoft Access : Application de gestion de bases de données, qui permet de stocker des informations et de les manipuler.

Requête : Instruction `SQL` qui permet d'interroger une base de données pour récupérer des données.

SQL : Langage de requêtes structurées utilisé pour interagir avec les bases de données relationnelles.

Sub : Sous-procédure en `VBA` qui permet d'organiser le code en le divisant en parties plus petites et plus faciles à gérer.

Table : Structure de données qui stocke des informations sous forme de lignes et de colonnes.

Tableau de bord : Outil de visualisation des données qui permet de suivre et d'analyser les performances d'une entreprise ou d'un projet.

Ticket : Demande de support ou de service soumise par un utilisateur à l'équipe informatique.

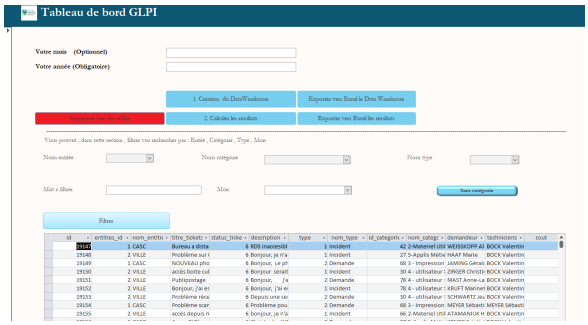
Update : Commande `SQL` qui permet de modifier les données d'une table.

VBA : Langage de programmation utilisé pour automatiser des tâches dans certaines applications Microsoft.

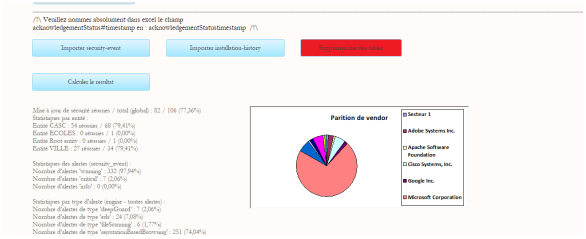
5 Annexes

5.1 Extrait du formulaire interactif

L'extrait ci-dessous montre une partie du formulaire interactif conçu dans Microsoft Access pour visualiser et modifier les données des tickets. Ce formulaire permet aux utilisateurs de sélectionner un ticket, de modifier ses informations (comme l'entité ou la catégorie), et d'exporter les données vers Excel.



(a) Partie haute du formulaire.



(b) Partie basse du formulaire.

FIGURE 18 – Extrait du formulaire interactif, divisé en deux parties : la partie haute (a) et la partie basse (b).

5.2 Exemple de transformation des données

Les tableaux ci-dessous illustrent un extrait de données brutes extraites de GLPI et leur transformation dans le data warehouse.

TABLE 1 – Extrait de données brutes de la table `glpi_tickets`.

id	entities_id	type	itilcategories_id
1	1	1	66
2	4	2	34

TABLE 2 – Extrait de données transformées dans `DataWarehouse_Tickets`.

id	entities_id	nom_entities	type	nom_type	id_categorie	nom_categorie
1	1	CASC	1	Incident	66	2-Materiel Util > Téléphone mobile
2	4	Écoles	2	Demande	34	3 - Impression

5.3 Les tables qui sont traitées ou créées

Voici une capture d'ecran des tables qui sont traitées ou créées dans le cadre de ce projet. Ces tables sont essentielles pour le bon fonctionnement du tableau de bord.

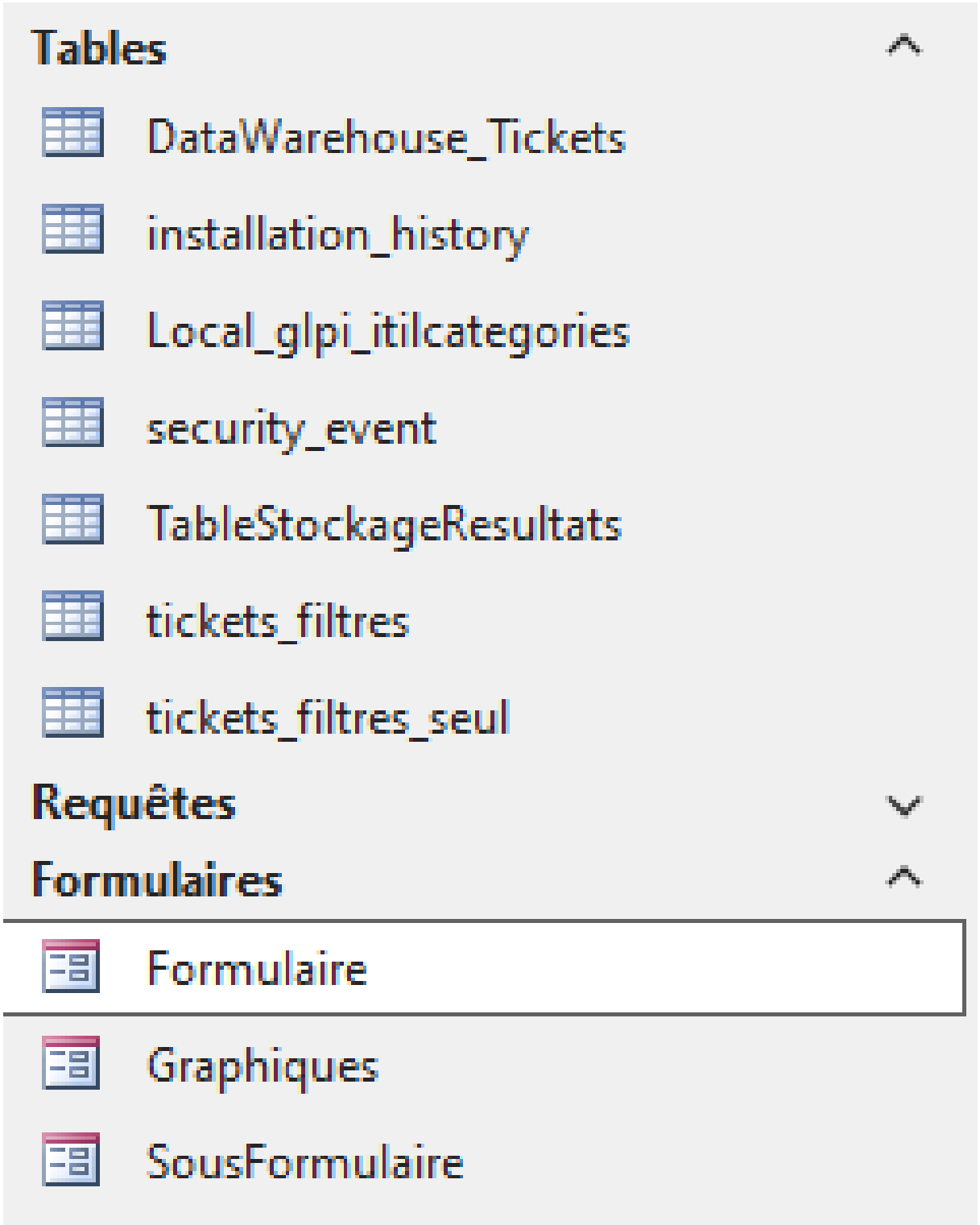


FIGURE 19 – Différentes tables utiles au bon fonctionnement du tableau de bord.

5.4 Partie du formulaire de graphique

L'extrait ci-dessous montre une partie du formulaire de graphique conçu dans Microsoft Access pour visualiser les données des tickets. Ce formulaire permet aux utilisateurs de visualiser des grpahiques intéractifs en fonction d'un mois et d'une année donnée.

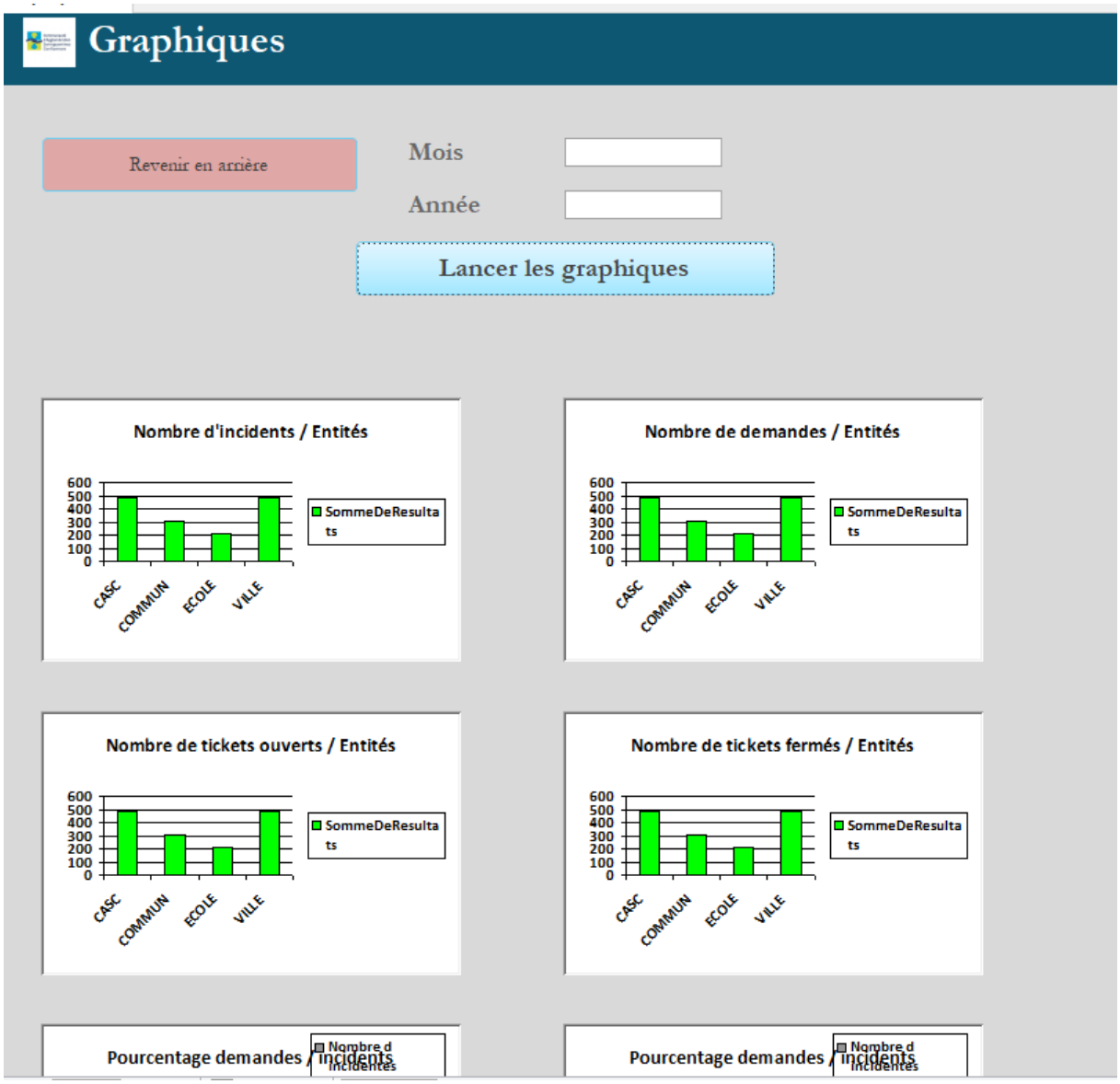


FIGURE 20 – Extrait du formulaire de graphique, permettant de visualiser les données des tickets.

5.5 Résumé du premier jour de stage

Le premier jour de mon stage, le 17 février 2025, a été une immersion marquante dans l'environnement professionnel du service informatique de la **Communauté d'Agglomération Sarreguemines Confluences (CASC)**. La matinée a débuté par une visite de la salle des serveurs, où j'ai pu découvrir l'infrastructure physique qui soutient les opérations informatiques de l'agglomération. Cette visite m'a permis de mieux comprendre l'importance de l'administration des systèmes et des réseaux dans un contexte professionnel. Par la suite, j'ai été présenté à mes collègues, dont **Monsieur Christophe Dahlem**, mon maître de stage, ainsi que les autres membres de l'équipe, ce qui a facilité mon intégration au sein du service.

L'après-midi, j'ai accompagné **Monsieur Alan Massia** et **Monsieur Sébastien** pour une intervention à l'hôtel de ville de Sarreguemines et à la médiathèque. Nous avons configuré des routeurs dans les salles serveurs de ces deux sites, une tâche essentielle pour maintenir la connectivité réseau. De plus, nous avons effectué des tests dans le cadre d'un projet de migration vers une nouvelle infrastructure réseau basée sur **Nutanix**, une solution de virtualisation et de gestion des données. Ces activités m'ont exposé à des aspects variés de l'informatique, allant de la gestion matérielle (serveurs, routeurs) à la planification stratégique d'une mise à jour d'infrastructure.

6 Bibliographie

Références

6.1 Sources

[1] Microsoft, *Documentation officielle de Microsoft Access*, Microsoft Corporation, 2025. Disponible sur : https://learn.microsoft.com/fr-fr/office/vba/api/overview/access.

[2] GLPI Project, *Documentation officielle de GLPI*. Disponible sur : https://glpi-project.org/documentation/.

[3] W3Schools, *Tutoriel sur SQL*. Disponible sur : https://www.w3schools.com/sql/.

[4] Communauté d'Agglomération Sarreguemines Confluences, *Présentation de la CASC*. Disponible sur : https://www.agglo-sarreguemines.fr/ (Consulté le 25 mars 2025).

[5] (Draw.io), *Création de diagrammes*. Disponible sur : https://app.diagrams.net/ .

[6] Département Informatique, IUT Nancy-Charlemagne, *SQL dans un autre langage*, 2023-2024-2025. Ressource interne.

[7] Reddit, *Forum de discussion*. Disponible sur : https://www.reddit.com/.

[8] Developpez.com , *Forum d'aide sur Microsoft Access*. Disponible sur : https://ledzeppii.developpez.com/odbc-access/.

Table des figures