Rapport explicatif du projet : TransConnect

Le projet Trans Connect est un outil de gestion pour la société Trans Connect permettant simplement et intuitivement à son directeur, Mr. Dupond, d’avoir accès aux informations dont il souhaite prendre connaissance et donc de pouvoir gérer sa société de la meilleure façon. Notre travail était de fournir un outil possédant différentes fonctionnalités à la fois du côté du client, du salarié mais aussi de l’historique des commandes.

Afin de travailler efficacement en groupe, nous avons utilisé GitHub qui nous a permis de versionner / merge notre projet de la façon la plus pratique possible.

Dans ce rapport nous allons développer les fonctionnalités de notre application de gestion que nous avons mise en place et plus particulièrement les outils qui ont été compliqués à mettre en place.

1. Système de sauvegarde optionnel

Tout d’abord notre application possède 3 axes : les clients, les salariés et les commandes.

Les clients sont une base qui s'enrichit au fur et à mesure de l’utilisation de l'application comme pour les commandes. Quant aux salariés, ceux-ci sont déjà présents de base et peuvent également être enrichis même s’il existe déjà un organigramme bien rempli au lancement du programme. Maintenant, une fois l’application lancée, vous pouvez décider d’effectuer des commandes, de créer des clients, de les modifier ou supprimer et d’embaucher ou de licencier des salariés.

Une fois cela effectué, si vous êtes satisfait de vos changements dans l’application, vous pouvez choisir de sauvegarder vos nouvelles modifications dans le menu principal en sélectionnant l’option 5 : Sauvegarder. Cela sauvegardera vos nouvelles données en plus des anciennes à la fois pour les salariés (l'organigramme), les clients. Pour faire cela, nous avons utilisé des fichiers au format json.

1. Fichiers json

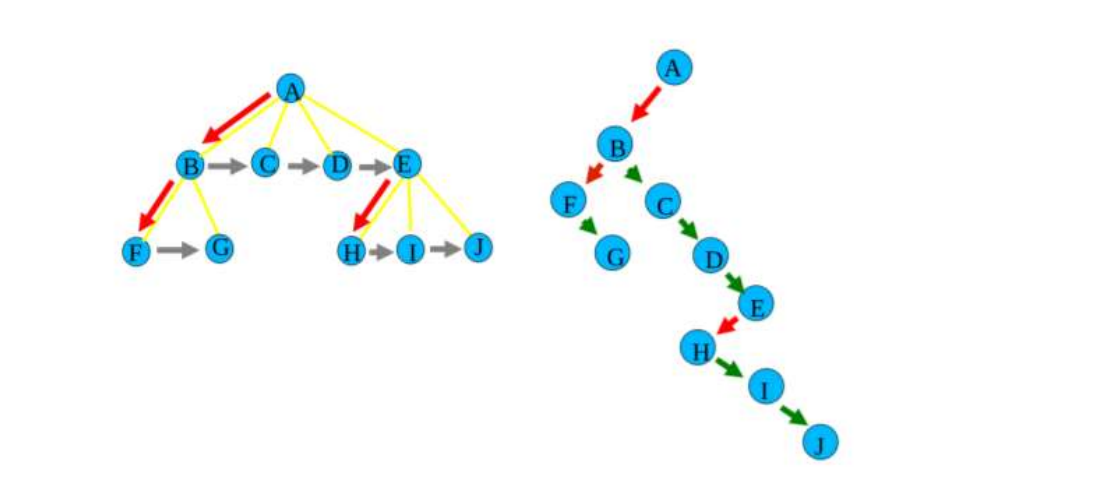
Pour effectuer tout notre système de sauvegarde et de base auto-enrichissante dans notre application, nous avons utilisé des fichiers locaux au format .json. Nous avons utilisé plus spécifiquement NewtonSoft Json.NET qui permet de serializer et deserializer des fichiers json c’est-à-dire de passer nos fichiers json en liste de salariés ou autre et inversement de passer ces listes ou autre en fichiers json. Cela nous a donc permis facilement de transférer nos données dans un fichier qui resterait localement afin de faire en sorte que l'application continue de s’auto enrichir même si le programme est arrêté. Voici un exemple de lecture de fichier json.



1. Organigramme des salariés - Arbre binaire / n-aire

Concernant notre organigramme de la société, nous avions comme contrainte de le réaliser sous forme d’un arbre binaire mais avec un nombre de fils qui peut être supérieur à 2. Donc, lorsqu’on lit notre organigramme grâce à notre fichier json local, on récupère un salarié qui est notre Directeur de l’entreprise et on déroule ensuite l’arbre afin de récupérer tout l'organigramme. Notre arbre est donc un arbre n-aire mais en binaire c’est-à-dire que chaque nœud peut avoir un fils et un frère au maximum.

De plus, on peut accéder à un des enfants d’une même fratrie mais pour accéder au reste de la fratrie on doit se déplacer grâce aux frères. Voici une illustration pour imager nos propos :



Ensuite, nous avons dû effectuer des méthodes permettant de pouvoir embaucher et licencier des salariés. Pour cela, nous avons utilisé la récursivité en traversant l’arbre jusqu’à trouver le salarié possédant le nom et le prénom que l’utilisateur recherche. Pour cela nous utilisons l’algorithme suivant :

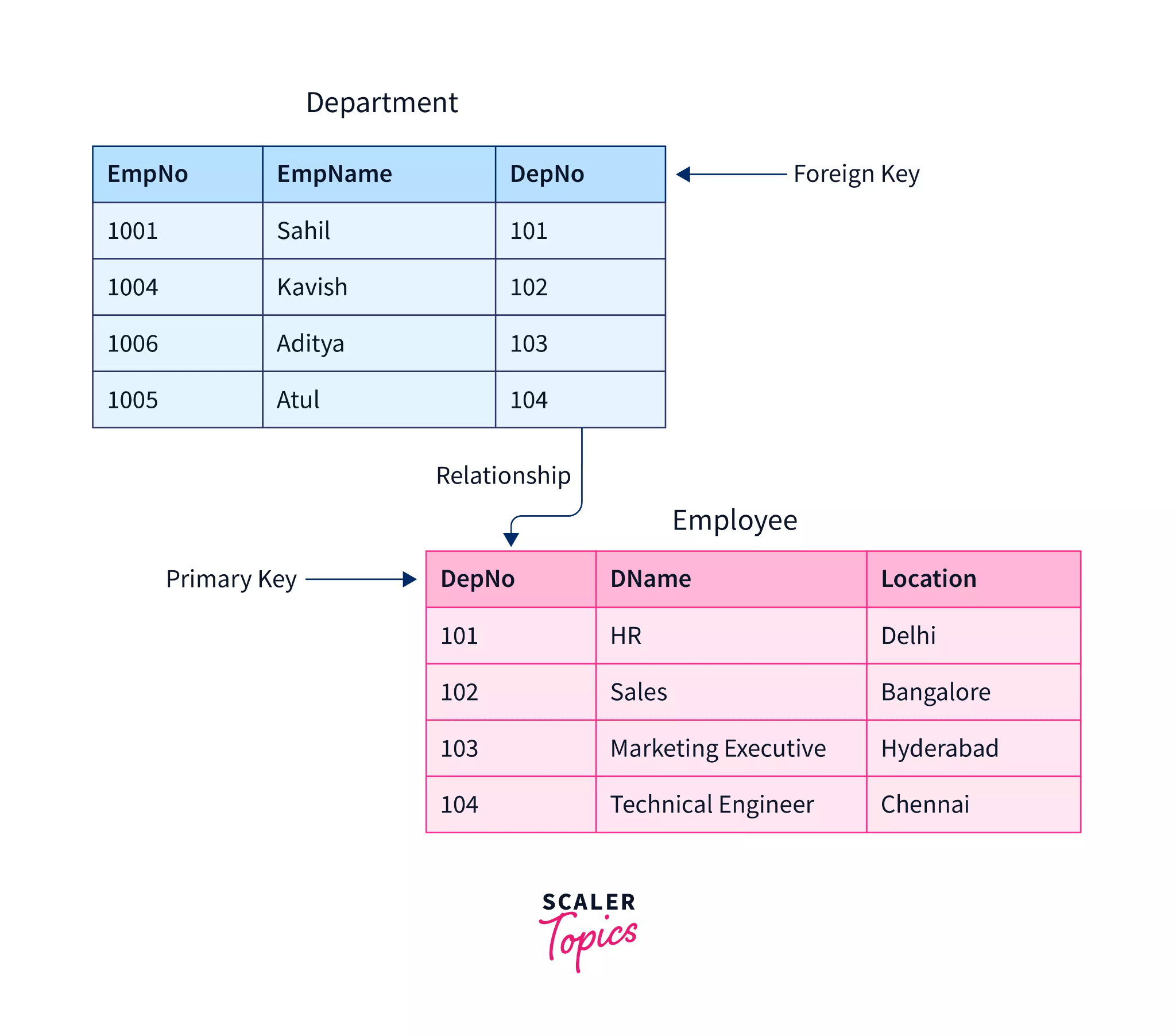


Notre méthode publique appelle donc la méthode private en démarrant notre appel récursif par le Directeur comme premier test. Ensuite, on descend l’arbre au fur et à mesure en passant par les Frères puis les Enfants.

1. Graphe des commandes

Enfin, nous avons utilisé pour notre module commande un graphe permettant de mettre en relation les villes de départ et d’arrivée de nos commandes. Cela a pour intérêt de nous permettre de, à chaque nouvelle commande, calculer la distance du trajet ainsi que le temps de trajet que mettra chaque chauffeur à effectuer la livraison. Pour cela, nous appliquons l’algorithme de dijkstra afin de trouver le chemin le plus rapide entre 2 villes données, présentes dans la classe Graph. La classe graph prend comme entrée un fichier csv qui initialise tous les vertex dans le fichier distance il y a villeA villeB, distance et le temps du trajet. Dans notre algorithme nous prenons seulement en compte la distance pour le chemin le plus court. La méthode Shortestpath retourne un Tuple des trois items dans l’ordre suivant : Chemin à parcourir (string), distance (int), temps de trajet (TimeSpan).

D’après le principe des foreign key, nous relions les chauffeurs, les clients et les commandes entre eux lors du constructeur de la commande qui s'occupe initialiser le propriétés commande (client / chauffeur) et client, chauffeur (commandes).



On utilise la distance dans le calcul du prix de la commande. De plus, nous avons implémenté une fonction Chauffeur libre qui retourne un chauffeur libre le jour de la commande si il n’y a aucun chauffeur libre la fonction retour null.