Rapport Intermédiaire :

Présentation du contexte du projet :

Dans le cadre de la matière de Projet Long , nous avons décidé de travailler sur le projet de recherche “Requêtes en SQL et leur graphe des dépendances”.
Le but de ce projet est de produire un logiciel utilisateur qui permet de visualiser les relations entre transactions SQL (écrite en pl/pgsql).
Le résultat attendue est une interface graphique mettant à disposition de l’utilisateur un graphe , composé de :

- Noeuds : Représentant les transactions
- Arêtes : Représentant les relations entre ces transactions.

Dans ce projet nous distinguons trois types de relations (dépendances) inter-transactions , à savoir :

- si un fichier N1 lit un attribut X dans la table T et un fichier N2 écrit dans ce même attribut X , dans la même table T alors le graphe(Q) doit :
  - contenir l'arête “rw,T(T.clé_primaire).X” , entre N1 et N2 .
  - contenir l'arête “wr,T(T.clé_primaire).X” , entre N2 et N1 .
- si un fichier N1 écrit un attribut X dans la table T et un fichier N2 écrit dans ce même attribut X , dans la même table T alors le graphe(Q) doit :
  - contenir l'arête “ww,T(T.clé_primaire).X” , entre N1 et N2 .
  - contenir l'arête “ww,T(T.clé_primaire).X” , entre N2 et N1 .

Les éléments suivant sont définies à la date du : 20 Avril 2020

Etat des lieux du logiciel :

Dans cette partie, nous allons exposer les parties et fonctionnalités présentent et fournis par le logiciel en ce jour.

Tout d’abord le logiciel est composé de deux parties, la première correspond à un Parseur qui analyse des fichiers SQL (Transaction) et plus précisément les opérations présentent à l’intérieur de ces fichiers. La seconde partie correspond à l’interface utilisateur, qui affiche le résultat obtenu de la première partie. Elle permet à l’utilisateur d’avoir une modélisation visuelle sous forme de graphe des transactions présentent dans les fichiers SQL traités dans la première partie.

Fonctionnalités du Parseur :

Le parseur , écrit en python, permet la création d'un fichier au format ".graphml" suivant une syntaxe bien précise (cf. syntaxe graphml classique), et un fichier au format ".gogol" (cf. voir partie sur le ".gogol") qui permet de donner de plus amples informations concernant les dépendances.

Le parseur analyse le contenu de chaque fichier ".sql" présent dans le répertoire sélectionné et applique une suite d'opérations , définie comme suit :

- Analyse du fichier "GenDB.sql" , pour définir les différentes tables et leurs clés primaires.
- Analyse l'ensemble des fichiers ".sql" pour établir la liste de :
    - leurs attributs présent en lecture (conditionnels ou non).
    - leurs attributs présent en Écriture (conditionnels ou non).

Une fois ces opérations effectuées , le parseur analyse l'ensemble des éléments générés pour produire une liste de dépendances.

Dans les fichiers générés nous avons :

Le ".graphml" qui permet :

- de distinguer les relations entre deux noeuds , ainsi que les clés primaires nécessaire à ces dernières.
- de distinguer également les relations conditionnelles (présente dans un "If") de celle dites "obligatoire" (non-conditionnelles).

le ".gogol" qui permet :

- pour chaque dépendance, de voir les lignes (dans le fichier source et le fichier destination) responsables de cette dernière.

Détails du ".gogol" :

C'est un format de fichier que nous avons décidé de créer pour le bien de notre projet car trop peu d'informations ne pouvaient être stocké dans le .graphml.

Voici la syntaxe de ce dernier :

```
<Relation ID="nom_dependance" SRC="fichier_source.sql" DST="fichiers_destination.sql"
CONDITION=Conditionnelle? >
        <SRC>
          l: x   Ligne_responsable_de_cette_dépendances_dans_le_fichier_source
        </SRC>
        <DST>
          l: y    Ligne_responsable_de_cette_dépendances_dans_le_fichier_destination
        </DST>
</Relation>
```

Fonctionnalité de l'Interface Utilisateur :
Les différentes fonctionnalités de l'Interface Utilisateur permettent de :

_ visualiser sous forme de graphe les transactions correspondantes aux fichiers ".sql" donnés en entrée du Parseur (cf: partie "Présentation du contexte du projet").

_ visualiser les dépendances des relations entre les transactions ainsi que leur raisons de dépendances.

_ rendre mobile les diverses parties du graphe pour permettre à l'utilisateur gérer la disposition du graphe afin d'en optimiser sa compréhension.

_ définir et changer des éléments de style liées aux éléments du graphe. Cette fonctionnalité a pour but de fournir à l'utilisateur la possibilité de mettre en évidence certains types de dépendances à sa guise mais aussi de changer le style visuel du graphe afin d'en adapter l'esthétique.

_ exporter le résultat du Parseur sous différents format, notamment sous les extensions .pdf, .gif, .jpeg, .jpg, .ps, .dot, .xdot, .png, .fig, .json, .svg .

Extensions prévus :

La liste des extensions prévus correspond aux diverses tâches d'implémentation que nous avons prévus ajouter avant la fin de PLONG.
Ces tâches ont été réfléchis au préalable et sont en passe d'être implémentées d'ici peu.
Les tâches en questions concernent essentiellement des détails et éléments en supplément du code fournis et expliciter dans la partie précédente.

_ gestion de chemin des arêtes après déplacement d'un sommet
_ amélioration du style de la Side Bar située du côté gauche de la fenêtre lors d'un click sur une arête.
_ amélioration de l'exportation : génération d'un graphe avec des arêtes numérotées et d'une légende (afin d'améliorer la visibilité du graphe généré)
_ permettre à l'utilisateur de masquer/afficher les noeuds qu'il souhaite dans le graphe

_ Optimisation de la recherche de ligne dans le Parseur Python.

_ Optimisation des relations Write-Write
_ Possibilité de nommer le fichier de création de la BD autrement que par “GenDB.sql” pour permettre une meilleure utilisation.

Extensions envisageables :

La liste des extensions qui suit correspond aux extensions nécessaire à la complétion du sujet de recherche “Requêtes en SQL et leur graphe des dépendances”.
Elle comprend la partie de recherche sur les cycles critiques, mais aussi diverses fonctionnalités utiles qui pourraient rendre plus agréable d’utilisation l’Interface Utilisateur.
De plus, il faudra prendre en compte les extensions énoncés dans la partie précédentes qui n’auraient pas été implémentées pour diverses raisons à la fin du semestres de PLONG.

Une première liste d’extension concernant les cycles critiques pourrait être :

- Permettre à l’utilisateur de mettre en évidence les cycles critiques de longueur N.
- Permettre à l’utilisateur de savoir s’il existe un cycle critique quelconque.

Une deuxième partie correspondant plus à l’interface graphique pourrait être composée de :

- Gérer les positions d’entrées/sorties des arêtes dans les noeuds
- Définir des règles plus pointues concernant le chemin suivi par les arêtes
- Permettre de sauvegarder les diverses configurations de style entre deux utilisations du logiciel

Une troisième partie concernant le Parseur Python :

- Produire des tests complémentaires aux différents tests créent jusqu’alors
- Améliorer le parser pour l’adapter à un plus large choix de fichier (autre syntaxe que pl/pgsql).
- Adapter des fichiers de transactions “.java” en “.sql” avec la syntaxe pl/pgsql.