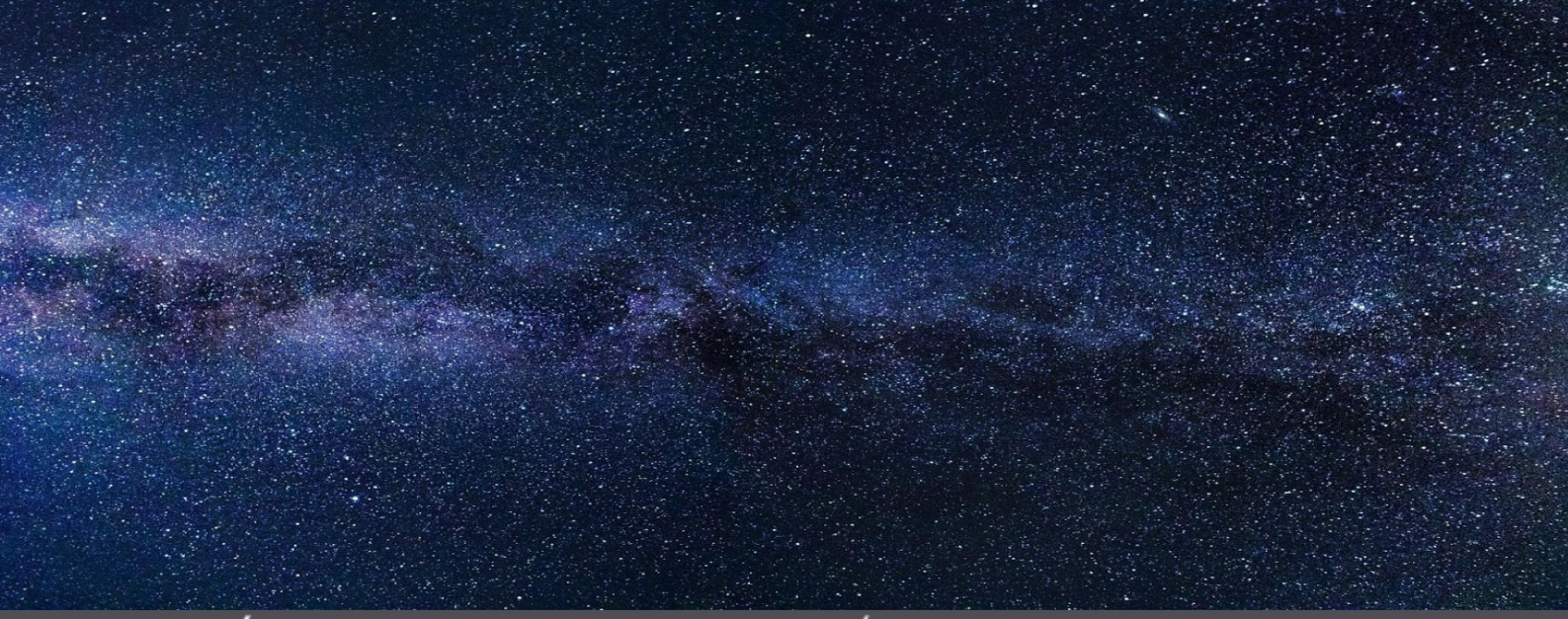


UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Développement d'un module Javascript pour échanger des données structurées au sein de l'Observatoire Virtuel

Rapport de stage ST40 - A2019

Tuteur en entreprise

Anaïs OBERTO
Laurent MICHEL
Gregory MANTELET

Tuteur UTBM

Claude RENAUD

Observatoire astronomique de Strasbourg

utbm université de technologie Belfort-Montbéliard

Remerciements

Tout d'abord, je tiens à remercier Mme. Anaïs OBERTO, M. Laurent MICHEL ainsi que M. Grégory MANTELET pour m'avoir dirigé durant les cinq mois de mon stage. Je remercie Mme. Anaïs OBERTO, qui a consacré énormément de temps à m'aider, me conseiller et m'avoir permis de mieux comprendre le sujet de mon stage. Je la remercie pour m'avoir donné toute l'aide dont j'avais besoin afin d'accomplir mes objectifs. Je remercie M. Laurent MICHEL pour ses conseils, son aide, sa disponibilité et il m'a partagé son expérience dans ce domaine qui m'intéresse tout particulièrement. Je remercie aussi M. Grégory MANTELET pour m'avoir aidé à trouver et résoudre les problèmes concernant l'utilisation de base de donnée et le langage ADQL.

Je souhaite remercier l'observatoire astronomique de Strasbourg pour m'avoir accueillie durant ces 5 mois. Je remercie M. André SCHAAFF pour avoir organisé plusieurs activités afin que nous puissions apprendre de nombreuses informations sur l'astronomie. Je remercie aussi tous mes collègues qui nous ont présenté leurs travaux au début du stage : cela nous a permis de connaître le fonctionnement de chaque service de l'observatoire et du CDS (Centre de Données astronomiques de Strasbourg).

Par ailleurs, je remercie toutes les personnes dans l'observatoire pour leur gentillesse envers moi.

En outre, je remercie Mme. Mireille JACQUOT, responsable du service des stages à l'UTBM, qui a suivi toute la partie recherche de stage et M. Claude RENAUD, mon enseignant-suiveur à l'UTBM, qui a veillé au bon déroulement du stage.

Enfin, je remercie toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage.

Sommaire

Figures

1. INTRODUCTION

Dans le cadre de ma deuxième année en cycle ingénieur au département Génie Informatique, à l'Université Technologique de Belfort-Montbéliard, j'ai effectué mon ST40, stage de six mois en tant qu'assistant ingénieur, au sein de l'observatoire de Strasbourg du 02 septembre 2019 au 07 février 2020. Ce stage est une première découverte du rôle de l'ingénieur au travail, des tâches et des responsabilités relatives à ce métier ainsi que de la méthodologie de raisonnement logique pour résoudre des problématiques algorithmiques qu'on peut représenter à l'aide d'un langage de programmation.

Au cours de ce stage j'ai intégré le projet portant sur le développement d'un module pour une interface Web, dont les responsables sont Mme. Anaïs OBERTO, M. Laurent MICHEL et M. Grégory MANTELET. J'ai aussi eu l'occasion de travailler avec les membres du CDS (Centre de Données de Strasbourg). En astronomie, beaucoup de sites web d'archives proposent un moyen d'interroger leur base de données avec une méthode commune appelée "TAP" (Table Access Protocol) et avec le langage "ADQL" (Astronomical Data Query Language).

Mon travail consiste à se connecter sur n'importe quel serveur d'accès aux données astronomiques proposant un service appelé TAP, et analyser le schéma cette base de données pour en extraire des données. Lorsque nous entrons une requête ADQL, elle sera exécutée sur un serveur d'archives et la reconstruction des données dans le contexte TAP. Ainsi, les principaux points réalisés durant ma période de stage sont:

- Module de médiation: conçu de manière à pouvoir être utilisé sur n'importe quel service TAP, il implémentera les fonctions suivantes:
  - Modélisation du schéma: Analyse du TAP_SCHEMA pour en extraire une représentation de la hiérarchie des données sérialisable en JSON.
  - Génération des requêtes: Le module devra être capable de générer des requêtes ADQL complexes à partir de contraintes placées sur différents noeuds de ce schéma.
  - Exécution des requêtes: Exécution dans un contexte TAP et reconstruction des objets complexes sous la forme d'instances sérialisables en JSON.
- Module de présentation: son rôle est de masquer le JSON à l'utilisateur, de faciliter la saisie des contraintes et de permettre la navigation dans les données retrouvées. Il sera d'abord dédié à la base Simbad du Centre de Données de Strasbourg. Il peut ensuite être adapté à d'autres plateformes de base de données implémentant le protocole TAP.

Ce rapport sera articulé à travers différents axes. Dans un premier temps, nous allons commencer par faire une introduction sur la présentation de l'Observatoire de Strasbourg. Puis nous allons présenter les objectifs et la feuille de route de mon stage. Une fois cette partie réalisée nous allons pouvoir rentrer dans le détail de mon stage. Nous développerons donc les parties concernants le service TAP et enfin les implémentations. Afin de conclure mon rapport, j'aborderai quelques points concernant mon expérience personnelle et ce que ce stage m'a apporté.

2. PRÉSENTATION DE L'ENTREPRISE

2.1. HISTORIQUE ET ÉVOLUTION

L'Observatoire astronomique de Strasbourg (ObAS) est en réalité le troisième observatoire de Strasbourg : le premier avait été construit en 1673 sur une des tours d'enceinte de la ville (l'astronome Julius Reichelt a notamment joué un rôle dans sa mise en place), et le second en 1828 sur le toit des bâtiments de l'Académie.

Situé à 1 km à l'est de la cathédrale, l'existence de l'Observatoire de Strasbourg procède d'une forte décision politique : lorsque l'Alsace-Moselle est cédée à l'Allemagne après la guerre franco-prussienne de 1870, l'empereur Guillaume II d'Allemagne décide de faire de Strasbourg une vitrine en triplant la superficie de la ville. Il y installe une université comprenant un jardin botanique et un observatoire astronomique. Édifice de style néo-renaissance, construit entre 1876 et 1880 sur les plans de l'astronome allemand August Winnecke, l'observatoire est inauguré le 22 septembre 1881.

Il est constitué de trois bâtiments : une Grande Coupole, un bâtiment des salles méridiennes avec deux coupoles, et un bâtiment à usage de bureau et de résidence. Ils sont curieusement reliés entre eux par un couloir en forme de « Y », couloir couvert pour se protéger des intempéries. Le bâtiment est doté de quatre frontons sur lesquels sont figurés l'Aurore, le Soleil, la Lune et l'Aurore boréale. La Grande Coupole en fer, de 9,2 mètres de diamètre et pesant 34 tonnes, contient le Grand Réfracteur, une lunette de 48,7 cm d'ouverture et 7 m de focale, construite en 1877, la plus grande d'Europe au moment de son installation, la troisième de France en taille, après celles de Meudon et Nice. Un rail permet de faire le tour de la grande coupole à une plus petite lunette permettant la découverte de comètes. Après avoir été équipée d'une lunette de 13,6 cm construite en 1879, puis d'un télescope de 60 cm jusqu'en janvier 2012, la coupole nord du bâtiment des salles méridiennes est dotée de deux télescopes de 35 cm, équipés d'une caméra CCD et d'un spectrographe. Quant à la coupole sud, elle abrite une lunette de 21 cm après avoir eu une lunette de 16,2 cm construite en 1876.

L'Observatoire a subi, tout comme la région dans son ensemble, les vicissitudes de l'Histoire, changeant plusieurs fois de nationalité. Durant la Seconde Guerre mondiale, il eut même un directeur de chaque nationalité, l'un à Strasbourg, l'autre à Clermont-Ferrand où l'université de Strasbourg s'était exilée. Des directeurs (Esclangon, Danjon, Egret) furent également par la suite directeurs de l'observatoire de Paris. En 1981 l'observatoire est doté d'un planétarium, aménagé dans une ancienne salle méridienne.

La lunette de l'Observatoire est la 3e plus grande de France par sa taille ainsi que le Planétarium de Strasbourg dont il a eu la responsabilité de 1986 à 2008, et désormais intégré au Jardin des Sciences de l'Université de Strasbourg. Il dispose également d'un riche patrimoine d'instruments et d'ouvrage anciens.

Figure 1 : Observatoire de Strasbourg

2.2. EVOLUTION SCIENTIFIQUE

La vocation initiale concerne pour partie l'astronomie de position, ainsi que l'observation de comètes, météorites et étoiles variables. Vint ensuite la photométrie de nébuleuses, l'observation d'étoiles doubles. Lors du retour de Strasbourg à la France, le nouveau directeur, Mr Esclangon, maintient le haut niveau de l'Observatoire. Installant électricité, téléphone, RSF et machine-outils.

Il s'intéresse à la chronométrie (il sera ultérieurement l'initiateur de l'horloge parlante). Son successeur, Mr Danjon, perfectionne l'instrumentation (photomètre, lunette méridienne, astrolabe). Pierre Lacroute prend néanmoins acte que l'observation astrométrique au sol a atteint ses limites instrumentales. À partir de 1965 il songe à l'observation satellitaire et propose le concept du satellite Hipparcos en 1973 à l'Agence spatiale européenne (l'European Space Research Organization, à cette époque). Dans le même temps, il développe l'archivage informatique, qui contribuera à donner naissance au centre de données stellaires, qui deviendra ensuite le Centre de Données astronomiques de Strasbourg.

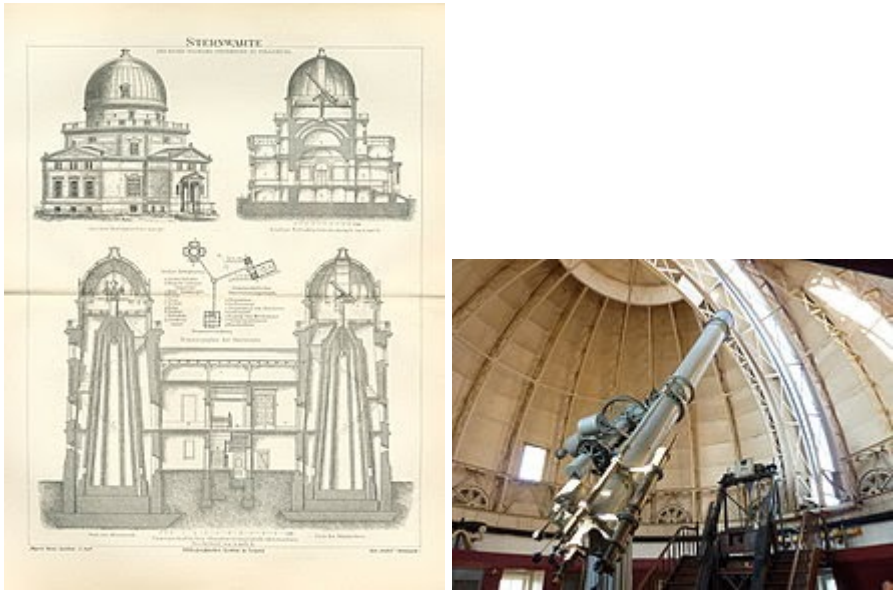


Figure 2 : Plan et lunette astronomique de l'observatoire de Strasbourg

2.3. ACTIVITÉS

L'Observatoire est un observatoire des sciences de l'univers de l'Institut National des Sciences de l'Univers (INSU), et une Unité Mixte de Recherche (UMR 7550) du Centre National de la Recherche Scientifique (CNRS) et de l'Université de Strasbourg. Comme pour tout observatoire des sciences de l'univers, plusieurs missions doivent être remplies : recherche, enseignement, services d'observation et diffusion des connaissances.

2.4. RECHERCHE ET ÉQUIPE

Les activités de recherche sont organisées autour de trois axes scientifiques :

- Astrophysique des hautes énergies : physique des astres compacts en fin d'évolution, accrétion, éjection et phénomènes magnétohydrodynamiques.
- Galaxies : populations stellaires, propriétés chimiques et dynamiques de la Galaxie et des galaxies proches, milieu intergalactique, grandes structures, et dynamique gravitationnelle.
- Méthodes de gestion de l'information et exploitation scientifique des grands relevés, en relation avec le centre de données astronomiques de Strasbourg.

En tant qu'unité de recherche, l'ObAS est structuré en deux équipes scientifiques, l'équipe GALHECOS et le CDS.

- **Équipe GALHECOS** : L'équipe « Galaxies, High Energy, Cosmology, Compact Objects & Stars » (GALHECOS) étudie la formation et l'évolution des galaxies et de notre Galaxie dans un contexte cosmologique, au travers de leurs populations stellaires, de la dynamique des étoiles et de la matière noire, et des effets de rétroaction liés notamment à l'activité de leur trou noir central. Elle s'intéresse aux sources galactiques et extragalactiques émettrices en rayons X, objets compacts (étoiles à neutron, naines blanches, etc.) et noyaux actifs de galaxies.
- **Équipe CDS** : Il s'agit d'un centre de données au cœur de la coopération internationale. Depuis sa création en 1972, le Centre de données astronomiques de Strasbourg (CDS) est dédié à la collecte et à la diffusion mondiale de données astronomiques et d'informations connexes. Le CDS héberge :
  - La base de données astronomiques SIMBAD, la base de données de référence mondiale pour l'identification des objets astronomiques ;
  - VizieR, le service de catalogue pour la collection de référence CDS de catalogues et tableaux astronomiques publiés dans des revues universitaires ;
  - Le logiciel interactif Aladin Sky Atlas pour l'accès, la visualisation et l'analyse des images ;
  - Surveys, catalogues et données astronomiques.

  Les astronomes du monde entier utilisent ses ressources, via ces services. La logique de l'observatoire virtuel y était donc déjà présente. De plus, il est un des principaux acteurs de l'IVOA, qui développe les standards nécessaires pour assurer l'interopérabilité des archives et des services astronomiques. J'ai effectué mon stage au sein de cette équipe.

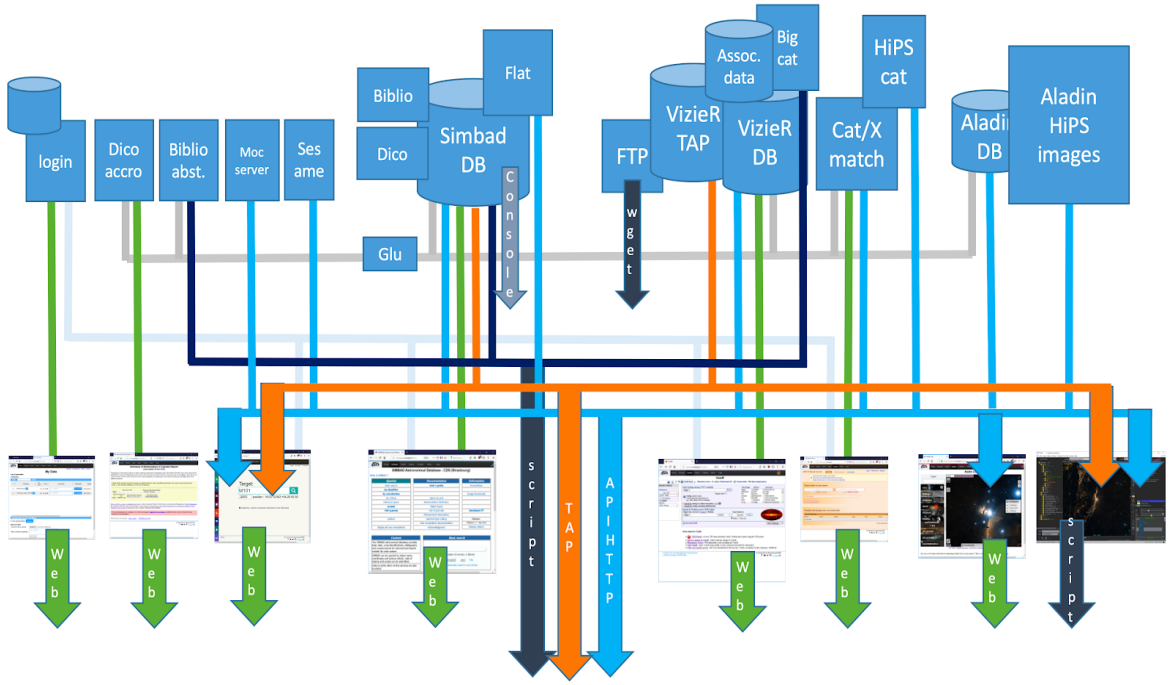


Figure 3 : CDS Information systems components

2.5. ENSEIGNEMENT

L'observatoire délivre la spécialité astrophysique de la mention physique du Master. Il participe également aux enseignements des licences et Master Sciences, à la préparation à l'agrégation et au CAPES, à la direction de stage, à la formation continue, etc.

2.6. SERVICES D'OBSERVATIONS

L'observatoire est membre du consortium Survey Science Center de la mission XMM-Newton, mais c'est probablement le Centre de Données astronomiques de Strasbourg (CDS) qui contribue le plus à la renommée de l'établissement. Le CDS offre comme service : Simbad (identification, bibliographie de 9 millions d'objets, hors système solaire), VizieR (service de catalogues), Aladin (atlas du ciel donnant accès à plus de 5 téraoctet d'images et servant de portail à l'observatoire virtuel, un miroir du service de bibliographie des journaux astronomiques de référence.

Figure 4 : Satellite XMM Newton

2.7. DIFFUSIONS DES CONNAISSANCES

La diffusion de la culture scientifique à l'Observatoire se fait principalement par le biais du Planétarium qui a pour vocation la vulgarisation et la diffusion des connaissances, avec lequel l'observatoire collabore étroitement .
Depuis 2008, le planétarium appartient à l'Université. L'Observatoire maintient cependant une forte activité de diffusion de la culture scientifique et technique lors de divers évènements, comme les journées du patrimoine ou la fête de la science. Le Planétarium propose des spectacles pour le grand public et les groupes scolaires régulièrement renouvelés qui permettent de découvrir de manière ludique les avancées des sciences de l'Univers.
Le Planétarium propose également une visite guidée de la Grande Coupole de l'Observatoire Astronomique de Strasbourg avant et/ou après chaque spectacle de planétarium.

3. PRÉSENTATION DU STAGE

3.1. CONTEXTE

Dans cette section, nous allons décrire les services et les outils sur lesquels s'appuieront mes travaux.

3.1.1. L'Observatoire Virtuel

L'Observatoire Virtuel est un consortium mondial d'observatoires et de centres de données astronomiques développant des standards d'interopérabilité permettant un accès uniforme aux données du domaine. L'Observatoire Virtuel (OV) est une initiative communautaire internationale en astronomie. Il vise à permettre l'accès électronique mondial aux archives de données astronomiques mises à disposition par les observatoires spatiaux, terrestres et d'autres bases de données d'observation du ciel ainsi que de nombreux projets et centres de données dans le monde.

L'IVOA (International Virtual Observatory Alliance) est une organisation qui débat et approuve les normes techniques nécessaires pour rendre les bases de données astronomiques interopérables. Elle sert également de cadre de discussion, de partage des idées et de la technologie de l'OV. De plus, elle est un organe important de promotion et de diffusion des standards de l'OV. Un certain nombre de ces standards ont eu une place centrale dans mon travail:

- **TAP** (Table Access Protocol) est un protocole de service Web, donnant accès à des catalogues astronomiques sous forme de tableau de données. TAP peut recevoir de la part d'utilisateurs des requêtes d'accès aux données et peut leur renvoyer les résultats. La requête doit pouvoir utiliser ADQL (Astronomical Data Query Language). Les requêtes peuvent être exécutées de manière synchrone ou asynchrone. Le résultat de la requête doit être renvoyé par VOTable, ou par FITS, JSON, etc. Nous pouvons également spécifier le nombre d'affichages par requête et choisir le mode asynchrone (i.e. batch mode). En fait, TAP décrit un mécanisme d'exécution de requête sur des données tabulaires.

- **ADQL** (Astronomical Data Query Language) est le langage utilisé par l'IVOA pour représenter les requêtes astronomiques publiées sur les services de l'OV. ADQL est basé sur le SQL (Structured Query Language). L'OV possède un certain nombre d'ensembles de données tabulaires et beaucoup d'entre eux sont stockés dans des bases de données relationnelles, ce qui fait de SQL un moyen d'accès

pratique. Un sous-ensemble de la grammaire SQL a été étendu pour prendre en charge les requêtes spécifiques à l'astronomie. Différents services de l'OV ont des besoins différents en termes de complexité des requêtes et ADQL se pose dans ce contexte.

- **VOTable** est une norme XML pour l'échange de données représentées comme un ensemble de tables. Dans ce contexte, une table est un ensemble non ordonné de lignes, chacune d'une structure uniforme, comme spécifié dans la description de la table (les *métadonnées de* la table). Chaque ligne d'un tableau est une séquence de cellules de tableau, et chacune d'entre elles contient soit un type de données primitif, soit un tableau de valeurs primitives. VOTable est conçu comme un format de stockage et d'échange flexible pour les données tabulaires, avec un accent particulier sur les tables astronomiques.

3.1.2. TapHandle

TapHandle est un client Web générique permettant d'accéder à des bases de données compatibles avec le protocole TAP de l'Observatoire Virtuel. Les résultats de requêtes sont actuellement affichés sous forme tabulaire. Dans TapHandle, il existe de nombreuses bases de données d'observatoires. Avec TapHandle, nous pouvons sélectionner la base de données que nous voulons interroger et il peut afficher tout le contenu de la base de données.

3.1.3. Simbad

Les éléments contenus dans la base de données sont des étoiles et des objets non stellaires. Par exemple, il contient environ 4 500 000 étoiles et environ 3 500 000 objets non stellaires (galaxies, nébuleuses planétaires, amas, novae et supernovae, etc.). Les corps du système solaire en sont exclus.
Chaque objet dispose de ses propres données. Les étoiles sont décrites par un type, des coordonnées, un mouvement propre, une vitesse radiale, une parallaxe, un type spectral, des magnitudes et flux pour plusieurs longueurs d'onde et notes. Les coordonnées sont stockées dans le système ICRS (International Celestial Reference System). Pour des galaxies, *Simbad* dispose d'un type, de coordonnées, d'un décalage vers le rouge, de magnitudes et flux intégrés pour plusieurs longueurs d'onde, d'un type morphologique et dimension.
Quand nous cherchons dans *Simbad*, nous obtenons les principales informations. Nous pouvons aussi voir des catalogues associés à cet objet. Si nous cliquons sur le catalogue, nous accédons au contenu spécifique du catalogue affiché dans VizieR.

3.2. SUJET

Le sujet de mon stage est “Développement d’un module Javascript pour échanger des données structurées au sein de l’Observatoire Virtuel”. Parmi les standards, figure un protocole d’accès à des bases relationnelles nommé TAP. Ce protocole est basé sur ADQL, un langage de requêtes dérivé de SQL, et sur le TAP_SCHEMA, une description standard des méta-données. Comme dans toute base relationnelle, il est possible de stocker des données complexes telles que des hiérarchies d’objets en les répartissant sur plusieurs tables reliées entre elles par des clés de jointures. De tels enregistrements complexes peuvent être sélectionnés et leurs structures peuvent être reconstruites par des requêtes SQL contenant des jointures multiples. Mais le service TAP actuel oblige les utilisateurs à saisir eux-mêmes les instructions ADQL. Tout d'abord, l'utilisateur doit apprendre ce langage et également se familiariser avec les relations et le contenu des tables dans la base de données afin d'obtenir les données souhaitées. Mais ce processus peut être difficile pour une personne découvrant la base de données.

Nous voulons donc concevoir un module qui permet d’obtenir facilement les données choisies. L'utilisateur peut sélectionner la table dont il souhaite afficher les données principales. Dans un second temps il peut saisir les contraintes des données et ainsi sélectionner la ou les tables qui peut/peuvent être connectée(s) à la table racine. Ce module devrait être conçu de manière à pouvoir être utilisé sur n’importe quel service TAP.

3.3. ENVIRONNEMENT ET LOGICIELS

- **VSCode**
  Visual Studio Code, logiciel de développement et de production d’applications Web.
  Version: 1.38.1

- **HTML**
  L’HyperText Markup Language est le langage de balisage conçu pour représenter les pages web.

- **CSS**
  Cascading Style Sheets est un langage informatique qui décrit la présentation des documents HTML et XML.

- **Javascript**
  JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives. C'est un langage orienté objet à prototypes.

- **Typescript**
  TypeScript est un sur-ensemble de types JavaScript, qui peut être compilé en JavaScript pur. Le JavaScript compilé peut s'exécuter sur n'importe quel navigateur.

- **JQuery**
  JQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web. Nous avons également utilisé des composants graphiques de jQuery (JQuery UI).

- **Ajax**
  Asynchronous JavaScript and XML. AJAX est une technologie qui permet d'interroger un serveur et ainsi d'en récupérer des données de manière asynchrone.

- **JSON**
  JSON (JavaScript Object Notation) est un format d'échange de données léger. Il est facile à lire et à écrire pour les humains. Il est facile à analyser et à générer pour les machines. Dans mon projet, Les données sont fournies au format JSON.

- **Firefox/Chrome**
  Nous utilisons les navigateurs Chrome/Firefox pour exécuter le fichier HTML et pour tester l'application grâce à ses outils de développement.

- **Github**
  GitHub est un service en ligne qui permet d'héberger ses repositories de code. GitHub est un outil gratuit pour héberger du code open source, et propose également une partie payante pour les projets de code privés. Le code source du projet est publié sur Github.

3.4. PLAN DE DÉVELOPPEMENT

3.4.1. Prise en main du code Python

Le début de mon stage a été de se familiariser avec un fichier Python écrit par M. MICHEL. Dans le même temps, je devais utiliser le script Python existant pour trouver les données de l'observatoire et s'entraîner à générer du JSON. Dans ce processus, je me suis progressivement familiarisé avec la structure de la base de données de l'observatoire et l'utilisation des instructions ADQL. Dans le même temps, l'idée d'un modèle d'initialisation JSON a également commencé.

3.4.2. Concevoir la page et générer un JSON

J'ai besoin de concevoir moi-même une page pour entrer et afficher les résultats afin de tester le programme normalement. La première page contient quatre bases de données, ce sont: *Simbad*, *GAVO*, *VizieR*, *CAOM*. Ce sont quatre bases de données de l'OV.
La page Web peut générer des fichiers JSON et des instructions ADQL contenant des restrictions basées sur le formulaire sélectionné et des restrictions entrées manuellement. Et grâce au JSON, nous pouvons réorganiser les données venant du résultat de la requête ADQL.

3.4.3. Mettre en œuvre la fonction

Les fichiers des principales fonctions du stage sont rédigés en caractères dactylographiés. Le document est divisé en quatre sections. Voir un schéma global figure 5. La première partie consiste à analyser le schéma de la base de données en fonction de la table racine d'entrée. Ensuite, j'utilise AJAX pour effectuer une recherche de données, puis utilise les données obtenues pour générer un fichier JSON simple contenant des relations de table. La deuxième partie consiste à formater les données de l'objet ajax reçu afin d'obtenir les données souhaitées. La troisième partie consiste à analyser le fichier JSON généré contenant des données pour obtenir des instructions ADQL. La quatrième partie consiste à obtenir les données du fichier JSON généré et à les convertir en éléments HTML, afin qu'elles puissent être affichées plus clairement sur la page. Il inclut également la possibilité de générer des fichiers JSON qui ajoutent des restrictions aux fichiers JSON simples.

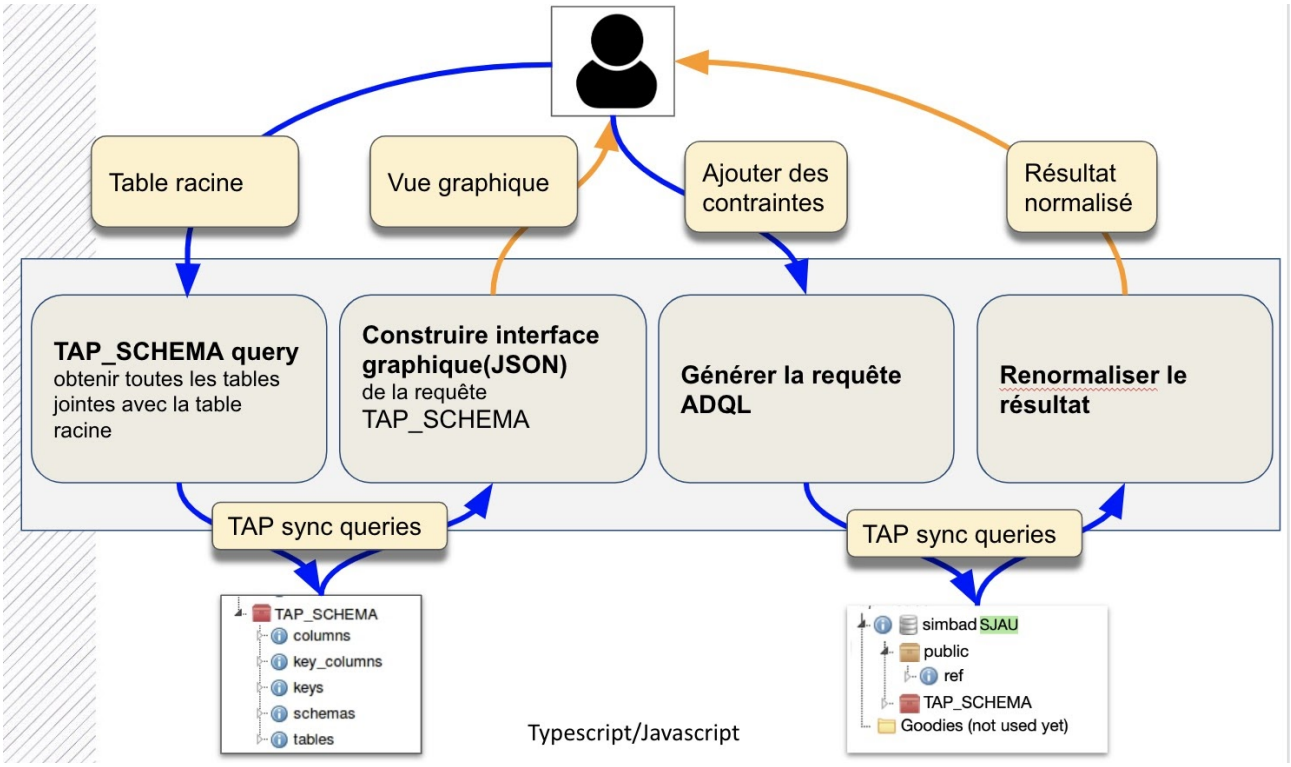


Figure 5 : Fonction du stage

L'utilisateur peut sélectionner une table racine. Selon la table racine, il va obtenir toutes les tables jointes avec la table de base. Et après, nous pouvons alors construire une interface où les utilisateurs peuvent entrer des contraintes. Lorsque l'utilisateur entre des contraintes, la page va générer une nouvelle instruction ADQL et interroge à nouveau. Les résultats de la requête sont ensuite affichés sur la page.

3.5. DÉROULEMENT DU STAGE

Le plan du stage s'est déroulé selon un plan qui suit les points suivants :

- Des spécifications ainsi qu'un plan de travail ont été fournis en début de stage.
- L'étudiant disposera d'un plan de marche avec des jalons.
- Il devra décrire l'avancement des travaux sur une plateforme collaborative.
- Il sera amené à présenter son travail devant les personnels de l'Observatoire.
- La priorité sera mise sur la finalisation des fonctions développées plutôt que sur l'implémentation de tous les modules spécifiés.
- Seule la description des fonctionnalités à réaliser est spécifiée. Le détail de l'implémentation est laissé au libre arbitre du stagiaire avec une validation des encadrants.
- Les fonctionnalités déjà prototypées devront juste être adaptées et finalisées.

3.6. SUIVI DU STAGE

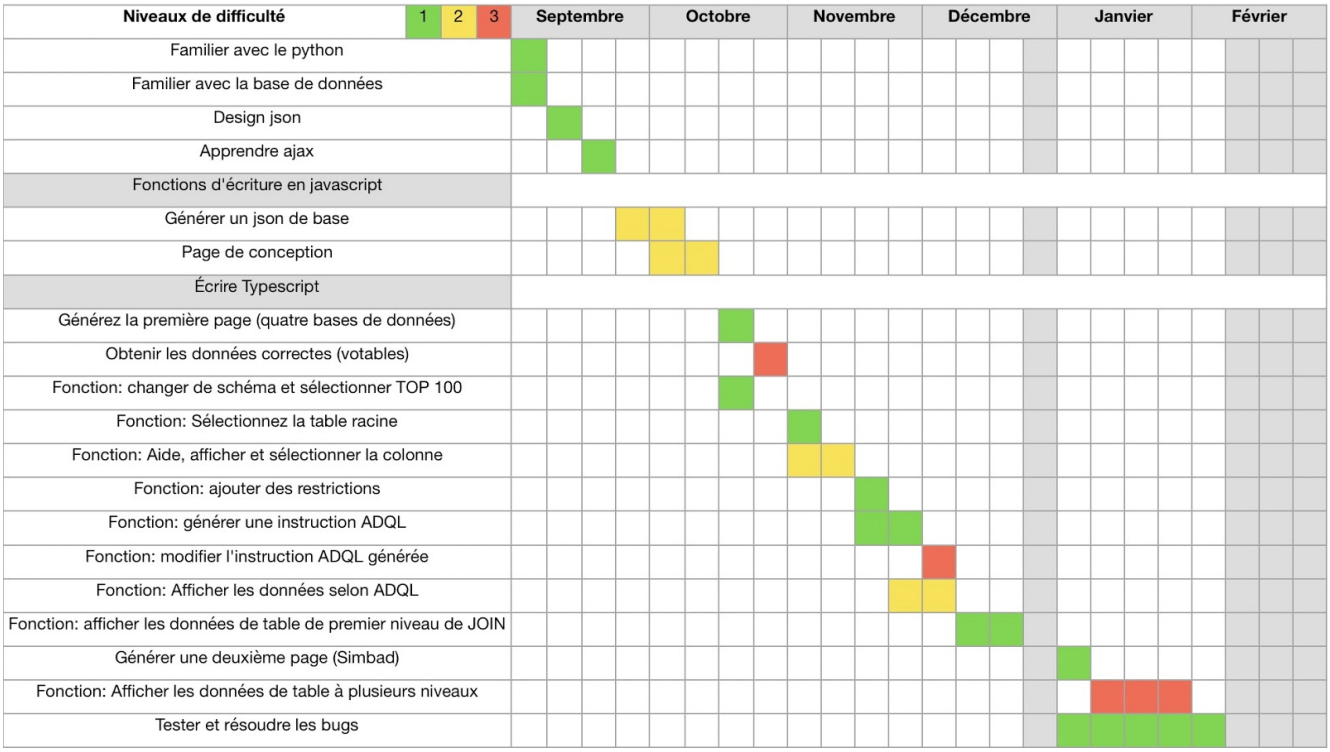


Figure 6 : Déroulement du stage

4. TRAVAUX RÉALISÉS

Afin de mieux comprendre le projet, j'ai besoin d'expliquer quelques petits termes au début.

- **la table racine**

La table racine est celle à partir de laquelle nous allons chercher toutes les relations dans le schéma. Dans la discussion suivante, il s'agira de la table de départ de toutes les requêtes ADQL.

- **TAP_SCHEMA**

Lorsque nous voulons obtenir les données de la relation connexe entre les tables, nous devons accéder à un schéma contenu dans tous les services de TAP: TAP_SCHEMA. Comme mentionné dans la section précédente, dans la plupart des bases de données des observatoires, il y a un service appelé TAP. Le Table Access Protocol (TAP) est un

protocole de service Web qui donne accès à des collections de données tabulaires appelées collectivement un ensemble de tables. Tous les services TAP doivent prendre en charge un ensemble de tables dans un schéma nommé TAP_SCHEMA qui décrit les tables et les colonnes incluses dans le service. Ainsi, lorsque nous devons obtenir la relation entre les tables de la base de données et les colonnes de base des tables, nous accédons à ce schéma.

Il y a cinq tables dans TAP_SCHEMA: TAP_SCHEMA.schemas, TAP_SCHEMA.tables, TAP_SCHEMA.columns, TAP_SCHEMA.keys, TAP_SCHEMA.key_columns.

TAP_SCHEMA.tables: cette table contient toutes les informations de la table dans la base de données. Par exemple, la description de la table, le schéma de la table et type de table.

TAP_SCHEMA.keys: key indique la connexion entre les deux tables. Dans ce tableau, chaque connexion a son propre identifiant, qui est key_id. Contient également le pointeur vers la clé (from_table, target_table).

TAP_SCHEMA.key_columns: cette table est la table complémentaire de TAP_SCHEMA.keys. Ces deux tables sont connectées par key_id et le nom de la colonne de la table cible (from_column, target_column).

En respectant l'ordre dans lequel elles sont exécutées. Il y aura quatre parties qui sont :

La première partie est que lorsque nous sélectionnons la base de données (dans ce rapport, nous utiliserons *Simbad* comme base de données) et le schéma, nous obtenons un fichier JSON de base. Ce sera la partie la plus fondamentale de ce projet.

La deuxième partie est que nous sélectionnons la table racine que nous voulons afficher. Ensuite, nous pouvons ajouter des contraintes. À ce stade, nous devons lire le JSON que nous venons de générer pour lire les relations entre les différentes tables, puis générer un nouveau JSON de la table racine vers la table que nous voulons joindre. Ce nouveau JSON contiendra des informations de jointure et des informations de restriction de la table racine à la dernière table.

La troisième partie est que nous allons lire le json contenant les informations de restriction et générer la déclaration ADQL.

La quatrième partie est l'instruction ADQL, qui interrogera le service TAP et renverra des informations, qui seront affichées sur la page après l'intégration. Dans la même section, les informations de la table sous la table racine peuvent être affichées.

4.1. PARTIE 1 : CRÉER LE FICHIER JSON DE BASE

4.1.1. Initialiser

Nous sélectionnons tout d'abord différente bases de données et différent schémas pour générer le fichier JSON correspondant à des base de données.

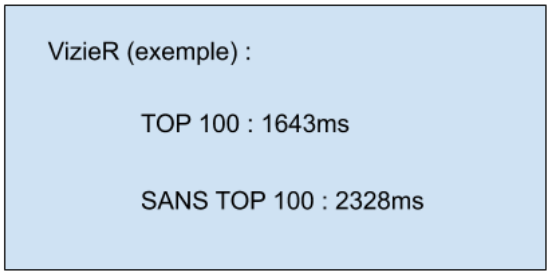
- **Sélection du schéma**

Sur la page Web, nous pouvons cliquer sur le menu déroulant derrière le nom de la base de données et sélectionner la base de données pour le schéma par défaut. (Le schéma par défaut de *Simbad* est "public"). Nous pouvons également saisir le schéma que nous voulons interroger dans la zone de saisie et cliquer sur le bouton Modifier pour exécuter le programme.

Figure 7 : Sélection du schéma

- **L'émergence du TOP 100**

Le but du TOP 100 est d'éviter que les données de la base de données soient trop compliquées et que le programme s'exécute trop longtemps.
Par exemple, nous sélectionnerons la base de données Vizier pour générer le premier fichier json. La figure suivante montre la différence entre le temps de fonctionnement de top100 et aucun top100.



S'il y a plus de données dans la base de données, il faudra plus de temps pour fonctionner sans TOP100.

4.1.2. Design JSON

La relation entre les tables de la base de données *Simbad* est illustrée ci-dessous:

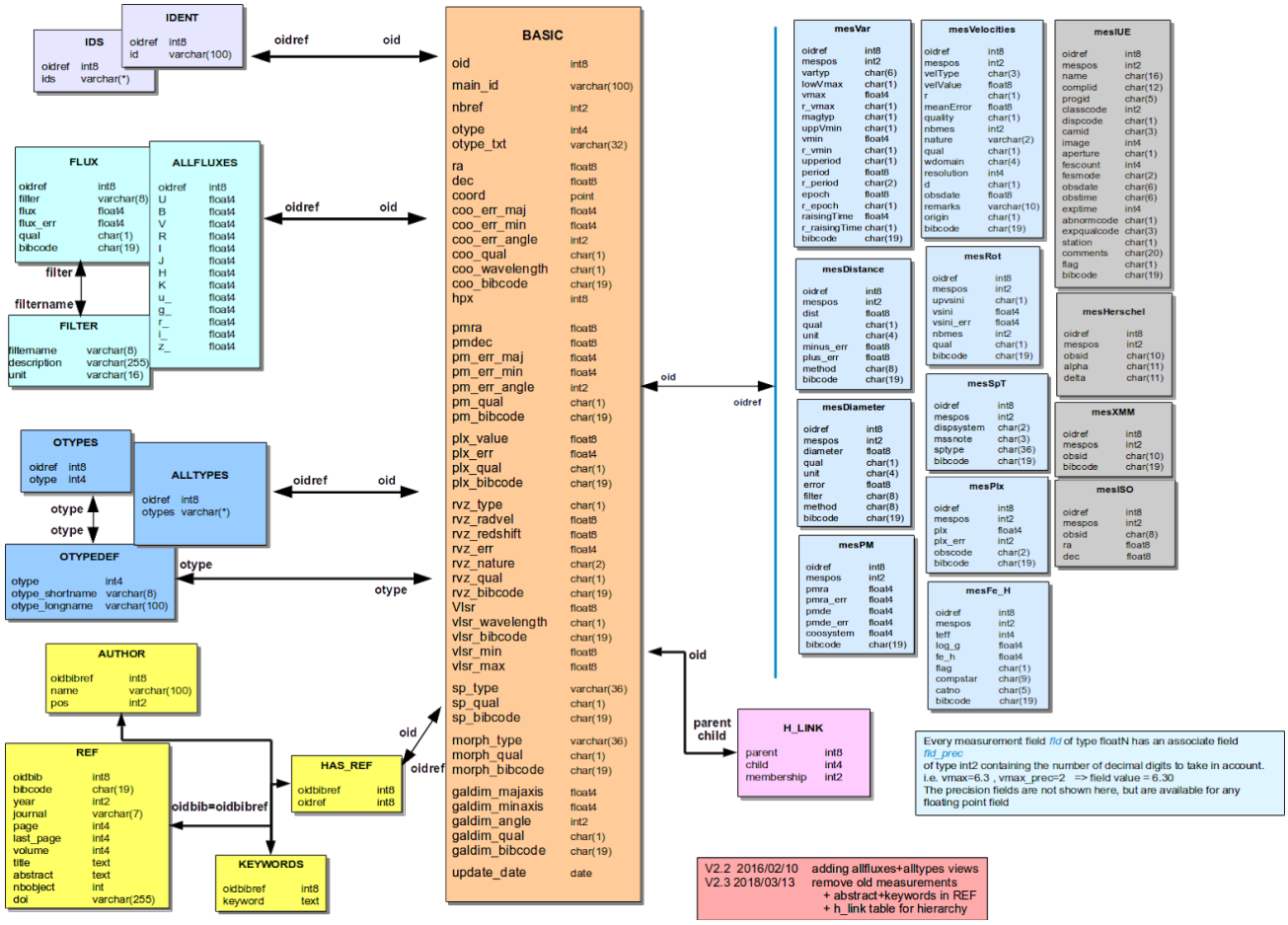


Figure 8 : les tables de la base de données *simbad*

Nous réalisons que les relations entre certaines tables forment une boucle sans fin. Par exemple, *basic-otypes-otypedef-basic*. Si nous voulons inclure les informations de connexion entre toutes les tables dans le JSON, le JSON sera trop lourd. Nous avons donc décidé que le JSON le plus basique ne devrait contenir que les informations de la table racine et de la première table qui lui est connectée. Par exemple la table ref:

```
"ref": {
            "schema": "public",
            "description": "Bibliographic reference",
            "join_tables": {
                  "has_ref": {
                        "schema": "public",
                        "columns": [],
                        "constraints": "",
                        "from": "oidbibref",
                        "target": "oidbib"
                  },
                  "author": {
                        "schema": "public",
                        "columns": [],
                        "constraints": "",
                        "from": "oidbibref",
                        "target": "oidbib"
                  },
                  "keywords": {
                        "schema": "public",
                        "columns": [],
                        "constraints": "",
                        "from": "oidbibref",
                        "target": "oidbib"
                  }
            }
      }
```

La table ref a trois tables qui lui sont connectées: ***has_ref***, ***author***, ***keywords***. Ainsi, dans la table ref, le JSON inclura le schéma où se trouve la référence, ses informations de description et les tables associées (join_tables). La propriété join_tables est constituée des trois tables qui viennent d'être mentionnées. De même, ils incluront le schéma de ces trois tables, puis nous pourrons remplir par la suite les colonnes et les contraintes. Et surtout le nom de la clé qui joint les deux tables. *target* est l'identifiant de la table la plus externe, qui est la table *ref*. *from* est l'identifiant de la table jointe.

4.1.3. Générer le JSON

4.1.3.1. AJAX

La base de données est accessible par URL pour interroger les données que nous voulons obtenir. Afin d'obtenir les données, nous choisissons d'utiliser la technologie AJAX. Dans TapService.ts, nous initialisons l'URL, le nom du schéma, le nom de la base de données et si nous choisissons TOP100.

Afin d'obtenir les données que nous voulons et analyser le schéma demandé, nous avons besoin d'instructions *ADQL* en entrée. En même temps, nous devons définir le format de données de sortie sur *VOTable*. Par exemple:

```
Query(adql:string){
   let site:string = this.url;
   var reTable;
   reTable = $.ajax({
     url: `${site}`,
     type: "GET",
     data: {query: `${adql}`, format: 'votable', lang:
'ADQL', request:'doQuery'},
     async:false
     })
   .done(function(result:any){
      return result;
   })
 return reTable;
 }
```

Dans la fonction ci-dessus, nous avons utilisé AJAX pour l'acquisition de données. Mais à l'heure actuelle, la réponse que nous avons obtenue est un objet incluant les données qui nous intéressent. Nous devons extraire et ré-organiser les données de cet objet pour faciliter leur utilisation.

Pour obtenir le premier fichier JSON de base, nous avons les instructions *ADQL* suivantes:

QueryTable：//Obtenir tous les noms et descriptions des tables.

```
SELECT
DISTINCT
TOP 100
T.table_name as table_name,
T.description
FROM tap_schema.tables as T
WHERE T.schema_name = \''+schema_name+'\'
```

QueryLink：//Obtenir toutes les données clés.

```
SELECT
TOP 100
tap_schema.keys.from_table as from_table,
tap_schema.keys.target_table as target_table,
tap_schema.keys.key_id,
tap_schema.key_columns.from_column,
tap_schema.key_columns.target_column
FROM tap_schema.keys
JOIN tap_schema.key_columns
ON tap_schema.keys.key_id = tap_schema.key_columns.key_id
```

4.1.3.2 VOTableTools.ts (Nom de fichier TypeScript)

Pour le traitement des données *VOTable (Format de sortie)*, tout est géré dans VOTableTools.ts. VOTableTools (Le même nom de classe que le nom de fichier) est une classe statique. Elle contient quatre fonctions statiques.

D'abord, nous passons l'objet des données de requête dans cette classe. Puis nous extrayons toutes les informations textuelles et nous les stockons dans la variable contentText. Le mode de codage des ces informations est détecté à partir de leur contenu.

```
var method = contentText.indexOf("base64");
```

- Encodage base64

Si la méthode variable est égale à -1, cela signifie que le mode de codage du texte est *base64*. Les modes d'encodage dans *Simbad* et *GAVO* sont tous les deux en *base64*. En informatique, *base64* est un codage de l'information utilisant 64 caractères, choisis pour

être disponibles sur la majorité des systèmes. Le principe du codage *Base64* consiste à utiliser des caractères US-ASCII (caractères non accentués) pour coder tout type de données codées sur 8 bits.

Au début, j'ai essayé d'écrire une fonction par moi-même pour transformer des données tronquées en données que je voulais obtenir. Mais cette fonction est très difficile à lire. Après le décodage de certaines phrases, les données sont toujours étranges. Si nous souhaitions modifier la fonction pour des situations particulières, cette fonction ne serait plus applicable à toutes les plateformes.

Donc, M. MICHEL m'a donné un fichier Javascript sur *VOTable*. Ce fichier est stocké dans le dossier module / jsimports. Son nom est votable.js. Dans ce fichier, il contient les fonctions de décodage *base64*. Mais en fait, lors de l'utilisation de ce fichier, il m'est difficile d'initialiser et d'exécuter les fonctions à l'intérieur. Donc, afin de faire fonctionner ce fichier dans mon programme, j'ai apporté quelques modifications aux fonctions de ce fichier pour finalement réaliser le décodage de *base64*.

À ce stade, il appellera une fonction content2Rows(). Dans cette fonction, VOTableParser() est une fonction instanciée de votable.js. Nous utilisons la fonction loadFile() pour passer le contenu obtenu dans la fonction pour l'initialisation, et elle renverra les données décodées. Il enregistrera les données directement dans un tableau à deux dimensions. Comme il s'agit d'un tableau à deux dimensions, le traitement des données devient compliqué. Nous effectuons donc un traitement secondaire sur les données afin qu'un tableau unidimensionnel puisse contenir toutes les données.

- Pas de mode d'encodage

Si la variable de méthode n'est pas égale à -1, cela signifie que le mode de codage du texte n'est pas *base64*. À l'heure actuelle, nous pouvons utiliser jQuery pour collecter et intégrer des données pour contentText. Le code est le suivant:

```
$(contentText).find('RESOURCE[type="results"]').each(function
(){
     $(this).find("TR").each(function(){
          for(let i:number=0; i<this.childNodes.length; i++){
            reData.push(this.childNodes[i].textContent);
          }
     });
})
```

Il trouvera d'abord la section des résultats des données. Ensuite, à travers la boucle, chaque donnée est stockée dans un tableau.

Ceci est le flux complet du travail VOTableTools. Il obtient l'objet des données VOTable, analyse le contenu des données et obtient enfin le tableau unidimensionnel où les données sont stockées.

4.1.3.3. createJson()

Cette méthode se trouve dans le fichier TapService.ts. Cette méthode est la fonction principale qui peut générer le premier fichier JSON. Il obtient d'abord tous les noms de table dans la base de données:

```
allTtable = this.allTable();//Get the array containing the
names of the tables
```

Ensuite, il obtiendra toutes les informations sur le lien:

- allLink()

Cette fonction n'est pas aussi simple que la précédente. Une fois cette fonction interrogée par la fonction de this.allLinkQuery(), les données ne peuvent pas être utilisées directement par nous. Parce que les données renvoyées sont un tableau unidimensionnel et que chaque lien contient l'ID de la table cible, celui de la table d'origine, l'ID de la colonne cible et celui de la colonne d'origine. Compte tenu de la cohérence des données, nous ré-organisons les données à l'aide d'une boucle. Chaque donnée du tableau bidimensionnel allLinkRe[][] contient toutes les données d'un lien, comme suit:

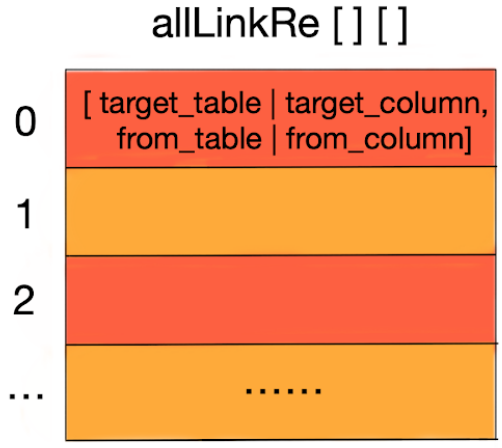


Figure 9 ：Structure des données de la table allLinkRe

Lorsque nous avons toutes les informations sur les tables et les connexions, nous commençons à générer le JSON. Tout d'abord, il utilise le nom de la table comme boucle la plus externe pour obtenir les informations de jointure de toutes les tables. Parce que

dans différentes bases de données, certains noms de table incluent un schéma. Nous utilisons donc la fonction getRightName() pour supprimer le schéma:

```
getRightName(table:string){
   if(table.indexOf(this.schema)==-1){
     return table;
   }
   else{
     return table.replace(new
RegExp(this.schema+'.','g'),"");
   }}
```

Deuxièmement, nous comparons le nom de la table avec target_table et from_table dans allLinkRe. Ceci est utilisé pour enregistrer la direction du lien.

Nous rencontrerons plusieurs liens entre les deux mêmes tables dans certaines bases de données. Par conséquent, le type de données de la colonne cible et des valeurs de colonne dans le dictionnaire sera modifié en tableaux pour enregistrer toutes les informations de liaison.

Afin de pouvoir enregistrer toutes les informations de lien, la fonction trouvera d'abord les données où target_table et from_table sont égales à partir des données enregistrées arrLink, puis comparera les liens. S'il s'agit de la même table mais pas du même lien, les données seront enregistrées directement dans arrLink. Mais pour le moment, car le premier lien a déjà été enregistré. Nous ne pouvons pas utiliser une copie superficielle pour simplement affecter le premier lien à la table. J'ai donc besoin d'utiliser json.stringify() pour le clonage en profondeur des données : （Si le clonage en profondeur n'est pas effectué, les données pointeront vers la même zone de stockage, ce qui rendra les procédures suivantes difficiles à effectuer normalement.）

```
temp1= JSON.parse(JSON.stringify(arrLink[ft[0]].from));
```

Lorsque la boucle la plus externe se termine, nous obtenons une variable de dictionnaire la plus élémentaire **data**, qui contient des informations sur la table et des informations sur les tables qui y sont connectées (join_tables). Le contenu du dictionnaire sera affiché dans la première partie de la page dans le fichier javascript via la fonction:

```
JSON.stringify (data, undefined, 6):
```

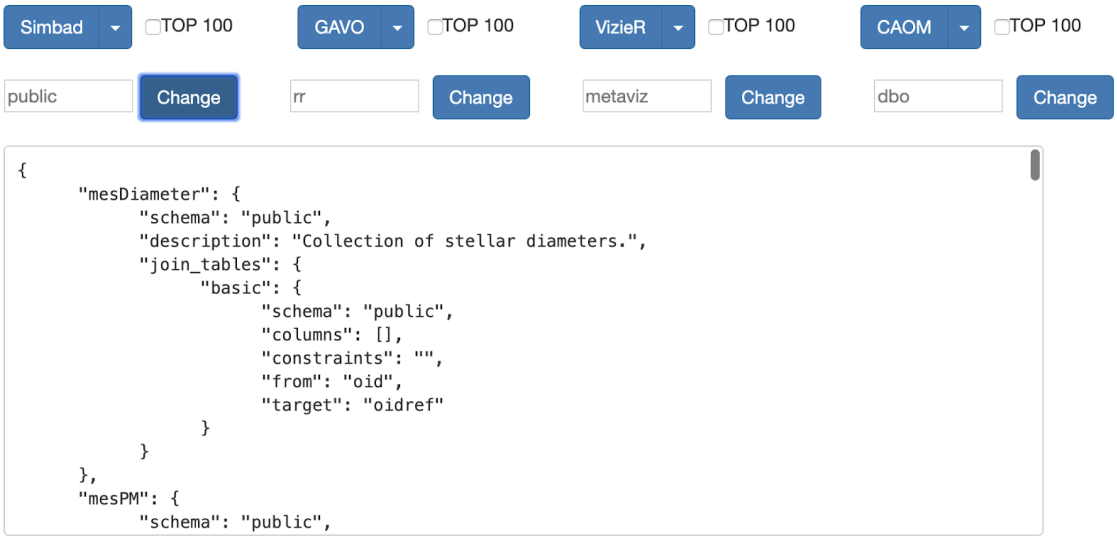


Figure 10 : JSON généré sur la page

Dans la deuxième partie de la page Web, afin d'afficher le fichier JSON de manière plus concise, nous lisons le nom de la table et la description de la table dans le premier fichier JSON, et affichons finalement les informations simplement pour la commodité des utilisateurs à vérifier les informations de jointure.

```
mesDiameter : Collection of stellar diameters.
    basic : General data about an astronomical object

mesPM : Collection of proper motions.
    basic : General data about an astronomical object

keywords : List of keywords in a paper
    ref : Bibliographic reference

author : Author of a bibliographic reference
    ref : Bibliographic reference

mesISO : Infrared Space Observatory (ISO) observing log.
    basic : General data about an astronomical object

mesMK : Collection of spectral types.
    basic : General data about an astronomical object

ref : Bibliographic reference
    has_ref : Associations between astronomical objects and their bibliographic referen
    author : Author of a bibliographic reference
    keywords : List of keywords in a paper
```

Figure 11 ：Relations de table sur la page

4.2. PARTIE 2 : CRÉER LE FICHIER JSON JOINT

Si nous voulons afficher toutes les informations de connexion à la table, le fichier JSON deviendra très lourd. Nous considérons donc la génération sélective de fichiers JSON. Le deuxième fichier JSON est généré sur la base des informations du premier fichier JSON.

Afin de permettre aux utilisateurs d'interroger plus facilement, nous avons décidé de laisser les utilisateurs choisir une table principale, puis développer en fonction de cette table principale. Le contenu du JSON se ramifie comme un arbre, avec la table principale comme racine, et les tables qui y sont connectées seront ses feuilles. Le nœud enfant est toujours la table jointe au nœud précédent. Nous l'utilisons pour développer le reste de l'arbre. Mais parce qu'il y a trop de tables dans certaines bases de données, nous avons décidé de sélectionner les tables. Plus on rejoint de tables, plus sa complexité sera élevée. Enfin, nous sélectionnons les cinq premières tables que les utilisateurs peuvent choisir.

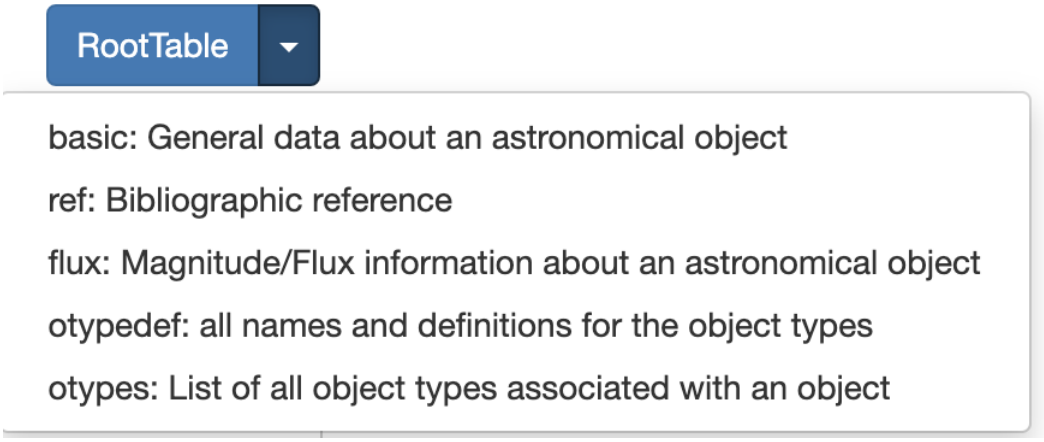


Figure 12 ：Sélectionner la table racine

Selon la figure ci-dessus, basic est la table avec le plus de JOINs (Jointure).

Nous pouvons sélectionner n'importe laquelle de ces 5 tables pour générer un deuxième fichier JSON.

4.2.1. Générer le JSON

Après avoir sélectionné la table racine, le nom de la table racine sera transmis à json2Html() en tant que paramètre.

- Petit problème

1. Il est impossible de prédire la hauteur de cet arbre.
S: Nous allons écrire des fonctions sous une forme récursive. Les fonctions récursives sont json2HtmlJoin(table, list_exist, flag) et ifJoin(data, list_exist, root). Dans cette fonction, le fichier JSON dans la première partie des données, list_exist est le nom de la table qui a été stocké et root est la table racine. La fonction json2HtmlJoin est utilisée pour formater le document HTML.

2. La connexion entre les tables peut former une boucle fermée, c'est-à-dire que les nœuds sont connectés à la table racine.
S: Créer une variable qui stocke le nom de la table qui apparaît: list_exist.

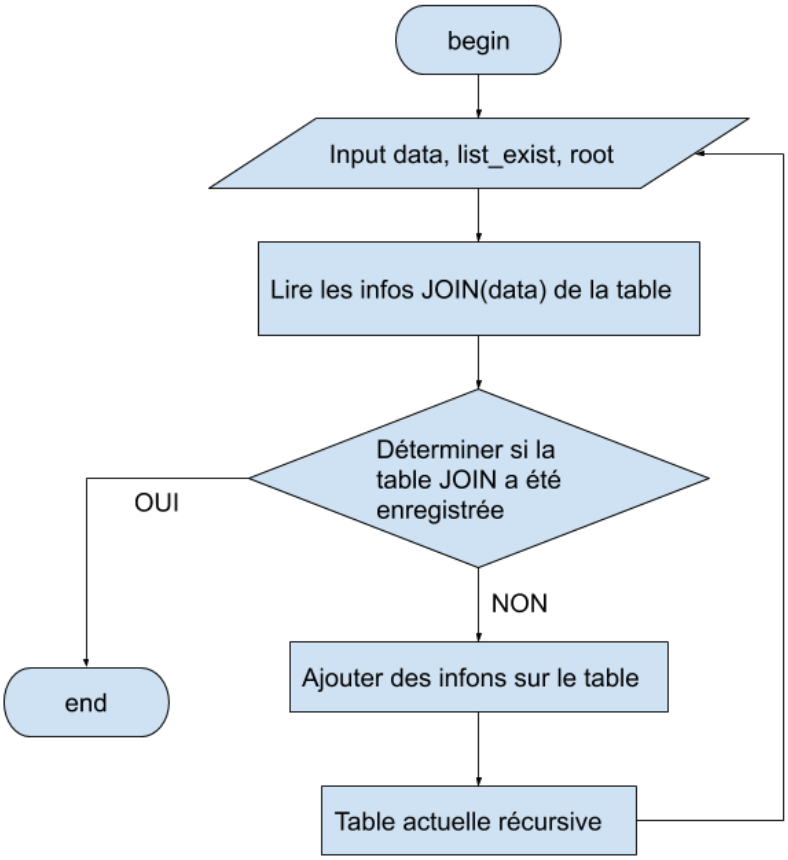


Figure 13 ：ifJoin(data, list_exist, root)

Lorsque nous parcourons cette série de fonctions récursives, nous pouvons obtenir un fichier JSON avec une structure arborescente enracinée dans la table racine. Il ne contient aucune restriction.

En utilisant la même logique que la fonction récursive ci-dessus, nous avons les fonctions json2Html() et json2HtmlJoin() pour générer des fichiers HTML. Dans ces fonctions, nous devons considérer que si toutes les chaînes sont concaténées, cela dépassera le nombre de mots sur la ligne. J'ai donc choisi d'utiliser un tableau pour enregistrer les documents HTML. Une fois tous les documents enregistrés dans le tableau, j'ai assemblé toutes les données, comme indiqué dans le code suivant:

```
json2Html (table:string):string{
......
```

```
    joinTable.push("<B>"+ table +"</B>"+ ": "+"<font color =
\"#545454\">"+ this.getDescription(table)+"</font>"+
"<br/>");
    joinTable.push("<button type=\"button\" id = "+"\"b1"+
table + "\" name = \"Cbutton\" class=\"btn
btn-primary\">Aide</button>"+"<input id="+"\"1"+table+"\"
type=\"text\" name = \"Cinput\" style = \"width: 500px\"
placeholder=\"constrains\">"+ "<br/>");
......
    htmlbuffer=joinTable.join('');
    return htmlbuffer;}
```

Lorsque nous obtenons le document HTML, les informations de connexion de la table apparaissent dans la deuxième partie de la page (avec basic comme table racine):

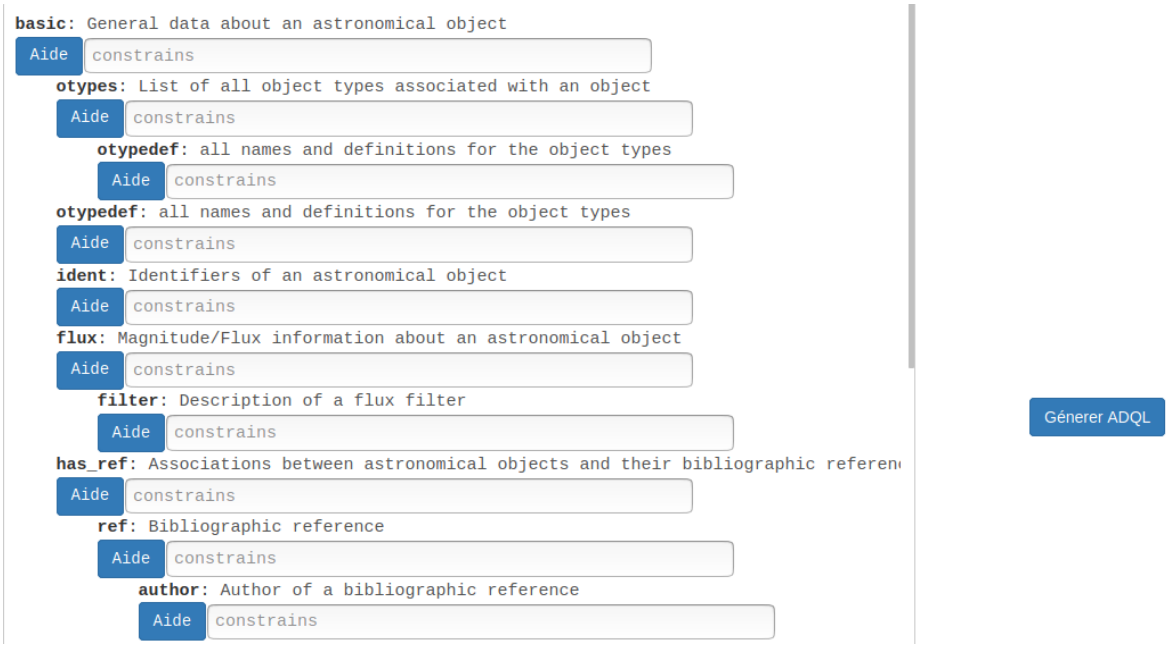
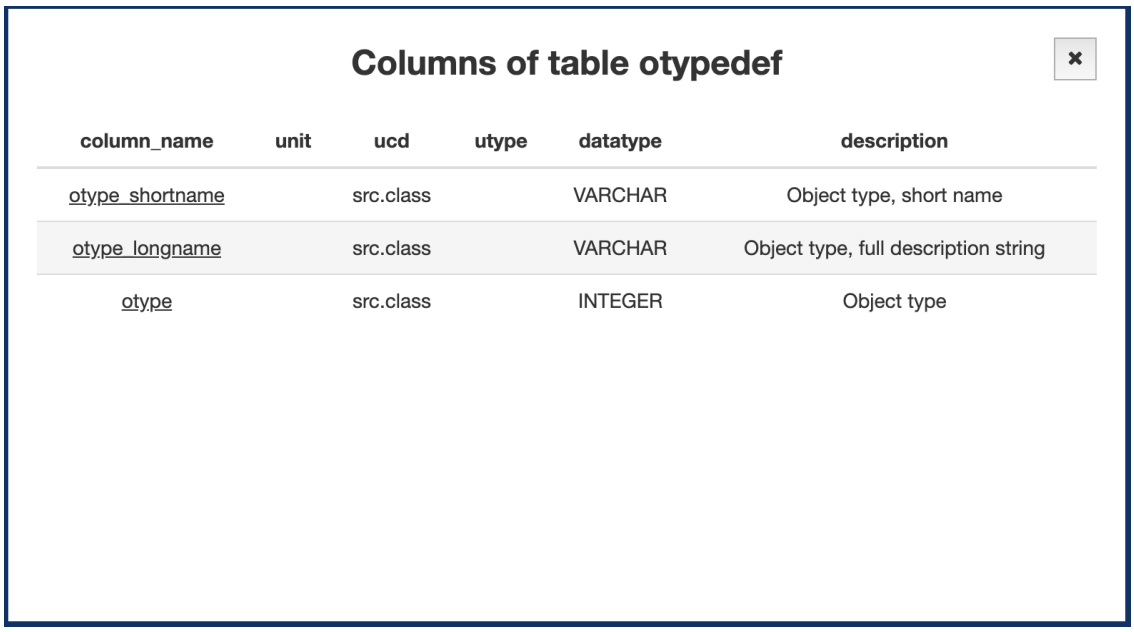


Figure 14 : Page fonctionnelle générée à partir de la table racine

4.2.2. Ajouter contraintes

Nous considérons que les utilisateurs peuvent ne pas connaître les colonnes des tables. Nous avons mis en gras le nom de la table et ajouté une description de la table à l'utilisateur afin que les utilisateurs en aient une compréhension générale. Dans la ligne suivante, il y a un bouton Aide. Lorsque nous cliquons sur le bouton Aide, une boîte de dialogue apparaît. Cette fenêtre affiche toutes les informations de cette table. Comme indiqué ci-dessous (prenons la table otypedef comme exemple):



Columns of table otypedef

column_name	unit	ucd	utype	datatype	description
otype_shortname		src.class		VARCHAR	Object type, short name
otype_longname		src.class		VARCHAR	Object type, full description string
otype		src.class		INTEGER	Object type

Figure 15 ：Colonnes de la table otypedef

Dans le formulaire ci-dessus, nous pouvons obtenir toutes les informations de la table sélectionnée. Lorsque nous voulons ajouter une restriction à certaines données de la table, nous pouvons cliquer sur les données soulignées sous nom_colonne. Par exemple, lorsque nous voulons restreindre otype_shortname, nous cliquons sur otype_shortname, et la colonne est automatiquement ajoutée dans la zone de saisie correspondante:

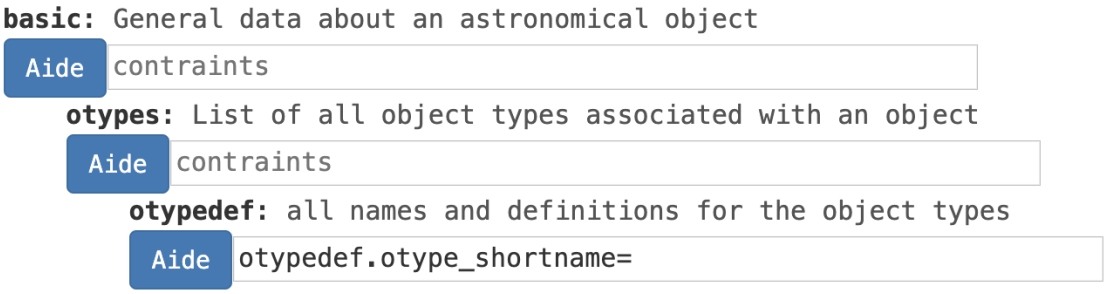


Figure 16 ：Page après avoir sélectionné la colonne à l'aide d'Aide

Cela facilite la saisie des informations par les utilisateurs et ajoute automatiquement le nom de la table avant la colonne.

Une fois les informations de restriction ajoutées, nous pouvons cliquer sur le bouton Requête à côté. Après avoir cliqué sur le bouton Requête, le programme place les informations JSON dans le tableau correspondant en fonction de la position des informations de restriction d'entrée.

4.2.3. JSON avec contraintes

Lorsque nous cliquons sur la requête, le programme intercepte les informations JSON qui ne sont associées à ces tables qu'en fonction de la table associée, et y ajoute les informations de restriction.

Mais lors de la génération d'un fichier JSON à ce moment, il existe deux possibilités. Tout d'abord, il n'y a pas d'entrée de restriction où seule la table racine est restreinte. À ce stade, toutes les informations de la table racine sont interrogées, c'est-à-dire que seul le JSON de la table racine est généré. Le second est une restriction conventionnelle sur les tables autres que la table racine.

Lorsqu'il n'y a aucune restriction, un tableau contenant uniquement le nom de la table racine est généré. S'il existe des restrictions, un tableau contenant les informations de connexion est généré. Dans l'exemple ci-dessus, nous avons ajouté des restrictions à la table otypedef. Nous allons donc générer un tableau résultant de la jointure basic-otypes-otypedef. Parmi eux, basic est connecté aux otypes, et les otypes sont connectés à otypedef.

Lorsque nous avons ce tableau d'informations, nous appellons une fonction appelée CreateJsonWithoutColumns. Cette fonction est une fonction récursive.

Afin de générer le fichier JSON approprié, la fonction prend en compte quatre cas après avoir lu ce tableau contenant les noms de table: 1. Seulement la table racine. 2. La table racine a join_table. 3. Pas la première table ni la dernière table. 4. La dernière table.

En prenant l'exemple ci-dessus comme exemple, nous obtenons le JSON suivant:

```
{
    "basic": {
        "schema": "public",
        "description": "General data about an astronomical
object",
        "columns": ["\"public\".basic.*"],
```

```
        "constraints": "",
        "join_tables": {
            "otypes": {
                "schema": "public",
                "description": "List of all object types
associated with an object",
                "columns": [],
                "constraints": "",
                "from": "oidref",
                "target": "oid",
                "join_tables": {
                    "otypedef": {
                        "schema": "public",
                        "description": "all names and definitions
for the object types",
                        "columns": [],
                        "constraints":
"otypedef.otype_shortname=’red’",
                        "from": "otype",
                        "target": "otype"
                    }
                }
            }
        }
    }
}
```

4.3. LIRE LE JSON ET GÉNÉRER L’ADQL

4.3.1. json2Requete.ts

Après avoir généré un fichier JSON avec des restrictions, l'étape suivante consiste à lire le fichier JSON et à générer l'instruction ADQL qui lui correspond.

Pour générer des instructions ADQL, nous avons la syntaxe suivante:

```
SELECT
……(column)
FROM
……(join tables)
WHERE
……(constrain)
```

Comme le nombre de tables est inconnu, dans la méthode getAdql(), nous lisons d'abord toutes les valeurs de colonne, les valeurs de contrainte et les valeurs de table de jointure dans le fichier JSON. La fonction qui obtient ces trois valeurs ici est toujours une fonction récursive. Nous devons interroger la valeur de chaque couche, s'il y a une valeur, elle sera stockée dans le tableau. De cette façon, lors de la génération d'instructions ADQL, nous pouvons déterminer la taille du tableau et connecter les données.

Dans column, si sa valeur est supérieure à deux, les deux valeurs sont directement connectées par une virgule <<,>> et une nouvelle ligne <</n>>. Mais pour <<*>> (interroger tout), nous ne devons pas ajouter de virgule après. Ainsi, lors de la génération d'une instruction ADQL, nous devons d'abord vérifier si elle contient une colonne avec <<*>>. S'il contient <<*>>, il doit être ajouté séparément dans la phrase. Pour les autres colonnes, nous pouvons les ajouter normalement. Dans le même temps, comme il y a trop de données dans certaines bases de données, nous ajoutons automatiquement une instruction <<TOP 100>> après le <<SELECT>> comme contrainte. TOP 100 est ajouté par défaut pour éviter que les données ne soient trop longues et que les pages Web ne s'affichent pas correctement.

Il n'y a pas de cas particulier de contrainte. Les deux contraintes ont été précédées d'une nouvelle ligne <</n>>, <<AND>> et d'une nouvelle ligne <</n>>.

Pour une table de jointure, chaque fois que la table lit la valeur de clé de join_tables, elle démarre l'enregistrement. Il ajoutera automatiquement la valeur de schéma et la valeur de table à la clé de jointure directe des deux tables, afin de ne pas rendre la valeur ambiguë dans les requêtes suivantes. Nous considérons que la requête de données doit être organisée dans un certain ordre pour que chaque résultat de requête soit identique, nous avons donc ajouté l'instruction <<ORDER BY>> à la fin de l'instruction ADQL. Elle permet de trier les valeurs clés de la table racine afin que chaque requête soit ordonnée de la première à la dernière.

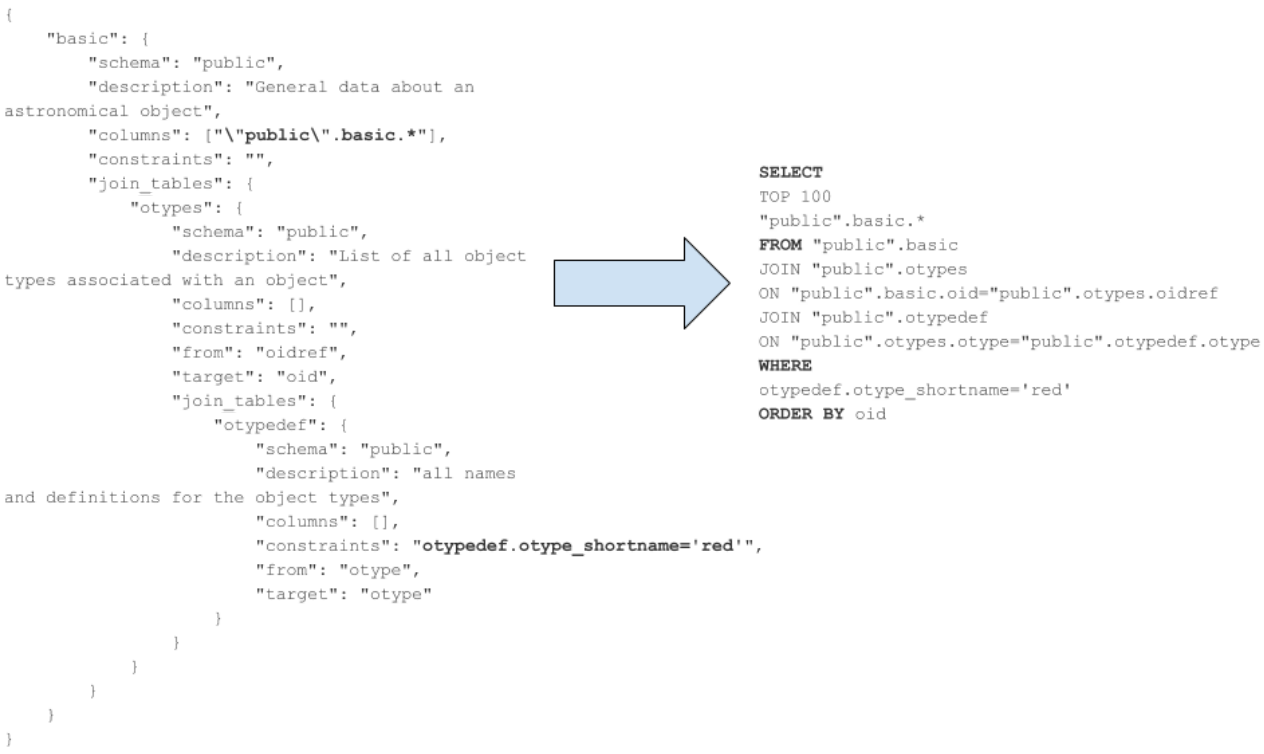


Figure 17 ：Générer des instructions adql basées sur json

Une fois que nous avons l'instruction ADQL, nous pouvons l'interroger. A l'identique de la méthode précédente, nous utilisons AJAX pour effectuer une requête par URL, passer une instruction ADQL en tant que paramètre à la fonction et analyser le texte du résultat obtenu pour obtenir le résultat de la requête. Ensuite, nous mettons l'instruction ADQL que nous venons de générer dans la zone de texte dans la troisième partie, permettant à l'utilisateur d'apporter une deuxième modification à l'instruction ADQL (Comme la référence à la figure qui suit). Dans le même temps, le nombre de données de requête et les données spécifiques de la table racine sont affichées sous la zone de texte.

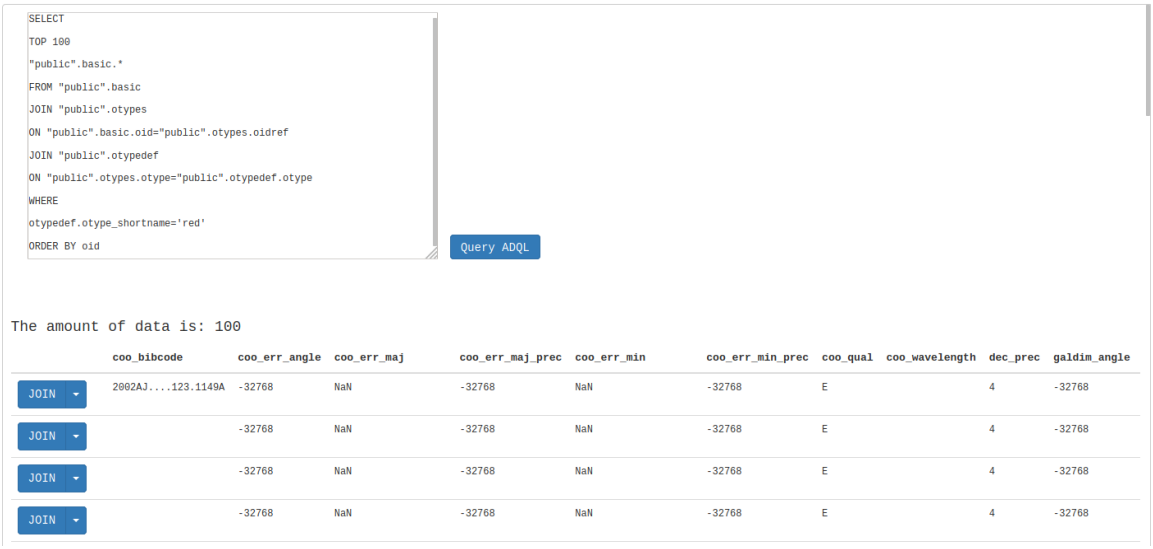


Figure 18 ：Recherche de données avec ADQL et affichage

4.3.2. Correction de l'instruction ADQL

Nous pouvons modifier l'instruction ADQL dans la zone de texte et cliquer sur le bouton Requête à côté de la zone de texte pour interroger à nouveau. Nous pouvons ajouter des instructions DISTINCT (pour éviter la duplication des données) et des instructions TOP 100 lors des requêtes. Même lors de la génération d'instructions ADQL, nous avons automatiquement ajouté les instructions TOP 100. Mais dans la requête modifiée, nous nous assurons toujours que TOP 100 soit présent dans l'instruction ADQL.

Dans le même temps, nous nous assurons que les valeurs de clé entre les tables peuvent être affichées au premier plan de la requête, donc avant que la fonction checkAdql() vérifie l'instruction ADQL. Dans cette fonction de vérification, nous passons un tableau contenant les valeurs de clé entre les tables.

Parce que la fonction qui vérifie l'instruction ADQL s'exécute automatiquement. Ainsi, même si nous n'avons rien modifié dans l'instruction ADQL ci-dessus, après avoir de nouveau cliqué sur le bouton Requête, le contenu de la zone de texte change comme suit:

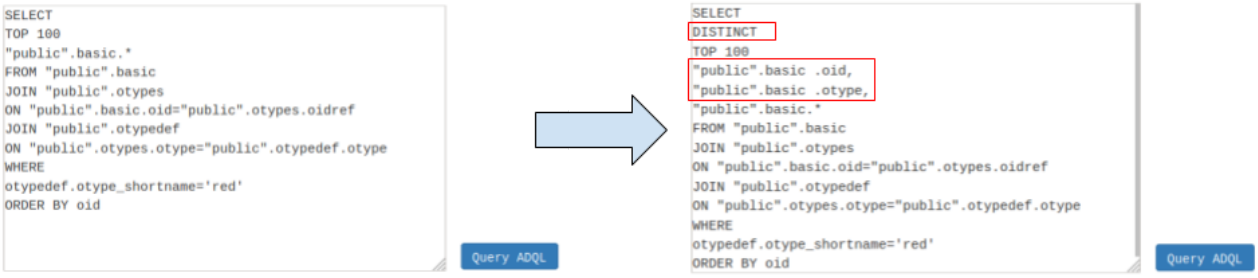


Figure 19 ：Instruction ADQL modifiée

4.4. Afficher plus de données

Les données que nous avons montrées jusqu'à présent sont liées à la table racine. Mais en fait, nous voulons obtenir plus de données. Par exemple, dans l'exemple ci-dessus, sous les mêmes contraintes, nous pouvons visualiser les données des otypes et otypedef ou plusieurs tables.

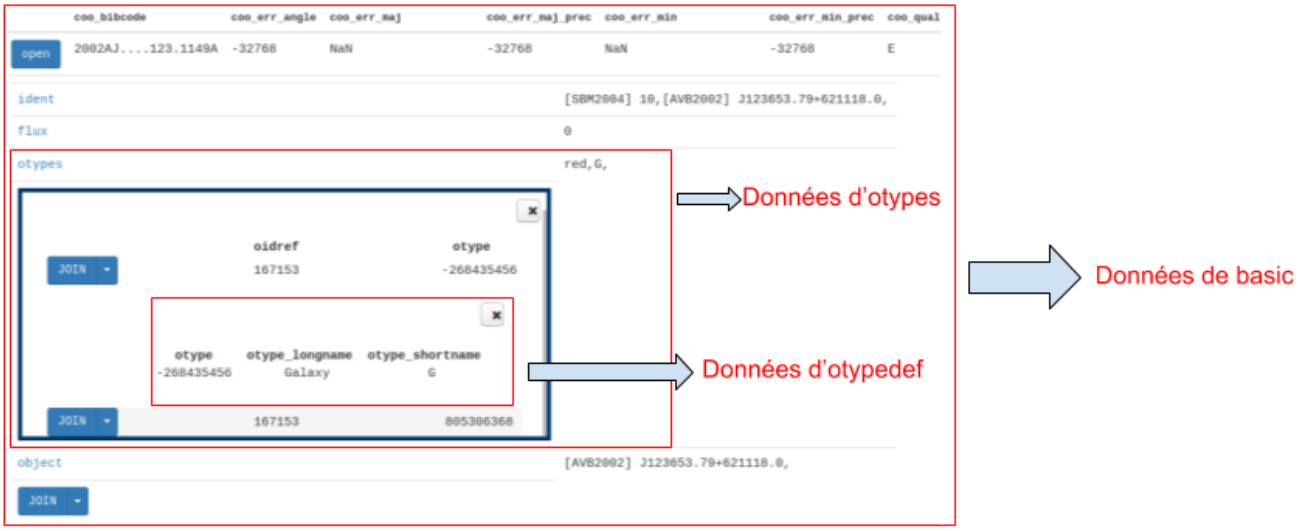


Figure 20 ：Découvrir les tables

Pour pouvoir voir plus de données de table. La valeur id du bouton stocke la valeur de clé entre les tables. Ensuite, lorsque le bouton est cliqué, le programme lit la valeur id du bouton et génère à nouveau une instruction ADQL basée sur la valeur.

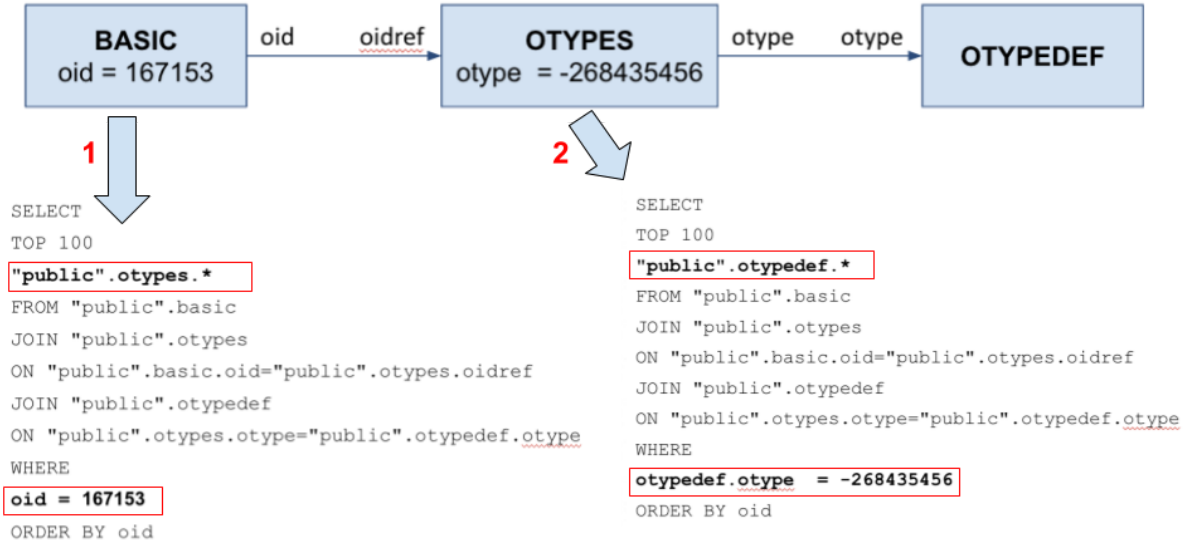


Figure 21 ：Générer ADQL par KeyId

5.CONCLUSION

En conclusion, j’ai bien atteint tous les objectifs. J'ai implémenté deux modules qui ont fonctionné. J'ai analysé le SCHÉMA de la base de données et généré du JSON. Il peut également régénérer des instructions de requête et afficher des données sur la page en fonction des contraintes entrées. J'ai développé des compétences TypeScript et une compréhension de JavaScript pendant le processus de développement. J’ai aussi acquis de nombreuses connaissances et d’expériences dans le domaine informatique durant ce stage.

5.1. DIFFICULTÉS

La plus grande difficulté que j'ai rencontrée pendant le stage a été de se familiariser avec les règles de chaque base de données et de trouver leur terrain d'entente. Le premier est que ce projet ne peut pas avoir trop de spécialisations, sinon il ne peut pas être appliqué à la plupart des services TAP. En d'autres termes, il ne devrait pas y avoir de code spécial pour une certaine base de données. Deuxièmement, la base de données contient une grande quantité de données. Pour garantir la cohérence des résultats de recherche de chaque donnée, vous devez ajouter des restrictions ou restreindre l'instruction ADQL. Par exemple, nous devons préfixer le nom de la table par un nom de schéma ou nous devons faire correspondre l'instruction ADQL avec la contrainte de clé primaire (en ajoutant l'instruction "ORDER BY") et ainsi de suite. Dans le même temps, la recherche de données dans le projet consiste à interroger l'URL via AJAX. Cela rend le projet très dépendant de la base de données. Si le contenu de la base de données ne peut pas être affiché normalement via l'URL elle-même ou si la relation entre les tables de la base de données est incorrecte, le projet ne s'exécutera pas correctement. (parce que la requête ajax repose sur l'url)

5.2. ACQUIS CLÉS

Après avoir résolu la plupart de mes difficultés, j'ai appris l'importance de bien organiser le code. Dans le même projet, il est inévitable qu'il y ait des fonctions en double qui ont la même logique. J'ai donc appris à ré-utiliser le code. Une bonne classification et des commentaires sur les fonctions sont la clé d'une productivité accrue ultérieurement.

6. BIBLIOGRAPHIE

Observatoire astronomique de Strasbourg:
http://astro.unistra.fr/
https://fr.wikipedia.org/wiki/Observatoire_astronomique_de_Strasbourg

Observatoire virtuel(OV):
https://fr.wikipedia.org/wiki/Observatoire_virtuel

CDS home page:
http://cdsweb.u-strasbg.fr/index-fr.gml

Votable:
http://www.ivoa.net/documents/VOTable/20130920/

Jquery:
https://jquery.com/

TAP:
http://www.ivoa.net/documents/latest/TAP.html

ADQL:
http://www.ivoa.net/documents/latest/ADQL.html

Simbad:
http://simbad.u-strasbg.fr/simbad/

TapHandle:
http://saada.unistra.fr/taphandle/

VizieR TAP:
http://tapvizier.u-strasbg.fr/TAPVizieR/

GAVO TAP:
http://dc.zah.uni-heidelberg.de/__system__/adql/query

CAOM TAP:
http://www.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/tap/

mots clefs

Centre ou laboratoire de recherche - Astrophysique - Étude, développement - Informatique - Développements logiciels - Architecture logicielle - Base de données - Logiciel d'analyse de donnée - Méthodologie - Paragraphes solidaires

Haoyun LIAO **Rapport de stage ST40 - A2019**

Résumé

L'Observatoire Astronomique de Strasbourg est un Observatoire des Sciences de l'Univers(OSU). Ses principales missions consistent en l'acquisition de données d'observation, le développement et l'exploitation de moyens appropriés et l'élaboration des outils théoriques nécessaires. L'Observatoire est une unité mixte de recherche de l'Université de Strasbourg et du CNRS (Centre Nationale de la Recherche Scientifique). Il offre la possibilité aux étudiants de suivre une formation de deux ans en Master Sciences et Technologie Mention Physique, spécialité Astrophysique.
Avec l'aide de Mme. Anaïs OBERTO, M. Laurent MICHEL et M. Grégory MANTELET, je me suis connecté avec succès à n'importe quel serveur d'accès aux données astronomiques qui fournissait des services TAP et j'ai analysé l'architecture de la base de données pour en extraire des données. Dans le même temps, un fichier JSON est généré pour enregistrer les contraintes et l'instruction de requête est régénérée pour des données de requête plus simples. J'ai donc eu l'occasion de mettre en pratique les enseignements théoriques reçus au cours de mon cursus universitaire et d'utiliser le langage de programmation TypeScript/Javascript HTML / CSS. De plus, j'ai travaillé sous l'environnement Linux. Pour la gestion de projet j'ai utilisé le système Git ; ce dernier m'a permis d'apprendre la gestion de version pour un projet.

Observatoire Astronomique de Strasbourg
CDS
11 rue de l'université
67000 Strasbourg
ASTRO.U-STRASBG.FR