

Certificat complémentaire en géomatique

Rapport de stage en entreprise :

Restructuration d'une base de données géospatiales et développement d'une extension QGIS

Présenté par

Robin VITAL

Sous la direction de
Grégory Giuliani (Université de Genève)
Bastien Deriaz (Ad Terra Energy)

Juin 2023

DROITS D'AUTEUR

Les citations tirées du présent mémoire ne sont permises que dans la mesure où elles servent de commentaire, référence ou démonstration à son utilisateur. La citation doit impérativement indiquer la source et le nom de l'auteur. La loi fédérale sur le droit d'auteur est applicable.

Remerciements

Je tiens à exprimer ma profonde gratitude envers mes directeurs, M. Gregory Giuliani et M. Bastien Deriaz, pour leur soutien précieux tout au long de ce mémoire. Leurs conseils éclairés et leur encadrement ont été essentiels à ma réussite académique. Je suis reconnaissant pour la confiance qu'ils m'ont accordé et pour avoir partagé leur précieuse expérience.

Je souhaite également remercier Ad Terra Energy pour m'avoir accueilli chaleureusement lors de mon stage et pour leur soutien continu. Je suis honoré de pouvoir continuer à travailler avec une équipe aussi compétente et passionnée.

Mes proches, mes parents, mes amis et mes camarades de l'université méritent également ma gratitude. Leur soutien inconditionnel et leurs encouragements constants ont été des piliers dans ma vie académique et personnelle.

Je suis profondément reconnaissant(e) envers tous ceux qui ont contribué à ma réussite, qu'il s'agisse de mes directeurs, de l'équipe d'Ad Terra Energy, de mes proches, de mes parents, de mes amis ou de mes camarades de l'université.

Résumé

Pendant le stage chez Ad Terra Energy, plusieurs réalisations significatives ont été accomplies dans les domaines de la géomatique et de la gestion de données. Une migration de base de données vers une nouvelle plateforme a été réalisée avec succès, permettant ainsi la maintenance quotidienne des données relatives aux champs d'exploitation. Cette initiative a contribué à assurer l'accessibilité et la mise à jour des informations pour l'ensemble des employés.

Une autre réalisation notable a été l'intégration réussie de données techniques, de production et d'informations sur les puits pétroliers. Cette intégration a renforcé la qualité des données disponibles, favorisant ainsi une meilleure prise de décision. De plus, une extension pour le logiciel QGIS a été développée, améliorant les fonctionnalités existantes en matière de visualisation et d'analyse des données géospatiales.

Ces réalisations témoignent de l'engagement et des compétences de l'équipe dans l'optimisation des processus de gestion des données géospatiales. Elles ont permis d'améliorer l'efficacité opérationnelle de l'entreprise, en fournissant des informations plus précises et en facilitant l'interprétation des données géographiques. Ces avancées ont contribué à une meilleure compréhension des activités de l'entreprise, permettant ainsi une prise de décision plus éclairée.

L'utilisation d'outils avancés et d'une approche proactive dans le domaine de la géomatique et de la gestion des données a été essentielle pour atteindre ces résultats. Ces réalisations démontrent l'importance de l'investissement dans ces domaines, qui jouent un rôle crucial dans la gestion efficace des ressources et dans la maximisation des opportunités dans le secteur de l'énergie.

Table des Matières

Remerciements	3
Résumé	4
I) Introduction	6
1.1) Introduction du rapport.....	6
1.2) Présentation de l'entreprise	6
II) Objectifs	7
2.1) Objectifs du travail.....	7
2.2) Cadres théoriques	7
III) Méthodes et Réalisations	9
3.1) Développement d'une base de données.....	9
3.2) Extensions « Plugins » QGIS.....	12
3.3) Restructuration de l'ancienne géodatabase.....	20
IV) Conclusion.....	23
4.1) Conclusion des réalisations.....	23
4.2) Réflexions sur le déroulement du stage	25
V) Bibliographie	27
VI) Annexes.....	28

I) Introduction

1.1) Introduction du rapport

Dans la continuité de mon cursus au Certificat de géomatique, j'ai eu l'opportunité d'effectuer un stage de six mois au sein de l'organisme Ad Terra Energy à Genève. Cette expérience représentait une opportunité de mettre en pratique les compétences développées dans les domaines de systèmes d'informations géographiques et d'approfondir celles en ressources énergétiques et minérales.

Ce rapport expose les enjeux de géomatiques rencontrés au travers de mon expérience de stage. La première partie présente l'entreprise d'accueil de mon stage. Ensuite, j'aborde les objectifs de mon travail et les différentes tâches effectuées. Enfin, je présente mes réalisations et conclus sur une réflexion critique et générale de mon travail au sein de l'organisme Ad Terra Energy.

1.2) Présentation de l'entreprise

Ad Terra Energy est une entreprise spécialisée dans l'exploration, le stockage ainsi que le développement des ressources minérales et énergétiques dans le monde. Basée à Genève, Ad Terra Energy opère depuis plus d'une dizaine d'années. La compagnie est spécialisée dans l'aide et dans le support aux entreprises, autorités et institutions publiques dans les processus d'extraction et de stockage des sources d'énergies fossiles et renouvelables.

L'organisation a été fondée en 2010 sous le nom initial de Geneva Petroleum Consultants International. Elle était à la base spécifiquement centrée sur les industries du gaz et du pétrole, avant de se diversifier progressivement au fil des années. Dans l'optique de développer une stratégie énergétique durable, la firme a développé une vision globale et inclusive en termes de nouvelles sources d'énergies, de séquestration du carbone et d'exploration des aquifères profonds. Le 1^{er} octobre 2021, l'entreprise changea d'identité pour finalement devenir Ad Terra Energy en fusionnant avec sa filiale Geneva Geo Energy. Elle est composée d'une équipe contenant des profils variés alliant des domaines de l'ingénierie, de l'expertise des champs de la Terre et des sciences appliquées, elle est aussi supportée par divers managers de projets. Benjamin Sallier (PhD) est le directeur général d'Ad Terra Energy, il dirige conjointement avec Bernd Fiebig (PhD Géophysicien) et Michael Suana (PhD Géologiste), en tant que superviseurs directs, des collaborateurs affiliés à l'organisme.

La compagnie accompagne divers projets suisses et internationaux liés à l'exploration des sous-sols par la réalisation de travaux de planification et de supervisions quotidiennes. L'acquisition et l'analyse des différentes données techniques, couplées au travail d'investigation de terrain, est aussi assuré par Ad Terra Energy. Les interprétations des données se font dans un but d'optimisation des potentiels sources géothermiques ou des rendements de réservoir. La société met en avant sa flexibilité dans ses multiples champs d'action. Ad Terra se focalise aussi dans la gestion des portefeuilles d'exploration et dans le développement de champs, pour préparer les outils nécessaires à la prise de décision concernant le déblocage des ressources. La réduction des impacts environnementaux liés aux activités constitue un autre point important des travaux de l'entreprise. Elle intègre donc à son panel d'activité des études de faisabilité en matière d'optimisation des pièges à dioxydes de carbone et de production de gaz. La firme réalise aussi des évaluations liées au domaine du stockage des déchets radioactifs provenant de l'utilisation de l'énergie nucléaire.

De ce fait, au travers de différentes thématiques, la compagnie s'illustre par différentes études de cas tels que la modélisation haute résolution des ressources pétrolières et gazières, l'analyse de processus sismiques géophysiques, des analyses sédimentologique, des stratégies de réduction des émissions de carbone (capture et stockage de carbone...) ou encore le management de base de données.

II) Objectifs

2.1) Objectifs du travail

Au cours de mon stage en entreprise chez Ad Terra Energy, j'ai participé à la réalisation de différentes tâches, principalement liée à la géomatique, mais aussi à d'autres domaines d'activités pour d'autres personnes.

Mes projets principaux se sont effectués dans le cadre de la migration massive d'une base de données d'une plateforme à une autre. Premièrement, j'effectuais les mises à jour journalières et hebdomadaires des différentes données reçus. Dans un contexte de changement d'hébergeur de données, il convenait de tenir à jour toutes les plateformes : l'ancienne encore très active et la nouvelle plateforme hébergeuse. Ainsi, je devais m'assurer que les données liées au champ d'exploitation soient à jour et disponible pour les employés. Je m'occupais donc de l'ajout des données techniques, de production et informatives concernant les puits pétroliers. La base de données était à l'origine hébergée sous Access, pour être ensuite importée progressivement sur PostgreSQL. Cette seconde alternative présentait de nombreux avantages en termes de performance et d'accès aux jeux de données.

Dans un second temps, mon travail a principalement reposé sur la réalisation d'un projet de développement d'extensions pour QGIS. L'objectif principal était de pouvoir faciliter la représentation cartographique des données géo-spatio-temporelles et informations contenues dans la base de données PostgreSQL pour les utilisateurs. Les extensions représentaient une solution intéressante pour permettre aux utilisateurs d'effectuer des requêtes SQL par le biais d'une interface graphique. En effet, sans avoir besoin de maîtriser le langage SQL, les usagers pourraient avoir accès à des informations spécifiques souhaitées, relatives au champ d'extraction des ressources. Ce projet rentrait en intégralité dans le cadre de la migration de la base de données.

Dans la continuité de l'élaboration des extensions pour QGIS, je me suis focalisé sur la géodatabase existante de l'entreprise. Pour ce faire, j'ai recensé l'accessibilité et l'utilité des données présentes dans l'ancienne base de données conçues pour ArcGIS (Laszlo Maio, 2014). L'objectif ici était de voir quelles données de géomatique pouvaient être récupérées et potentiellement incorporées dans une base de données PostgreSQL par le biais de l'outil Post GIS.

Mon stage au sein d'Ad Terra Energy m'a aussi permis d'expérimenter d'autres tâches en lien avec les activités de la compagnie. J'ai notamment participé aux analyses mensuelles des balances de gaz et des processus de torchage sur le champ d'exploitation. J'ai aussi aidé à la réalisation de la complétion des puits. Ma contribution à cette activité s'est traduite par un soutien à la mise à jour des dessins de certains puits selon certaines informations disponibles.

2.2) Cadres théoriques

Procéder à une migration de base de données revient à transférer les informations à disposition d'un gestionnaire de répertoire à un autre. La vraie complexité du procédé apparaît lorsqu'il est nécessaire de jongler avec des jeux de données très nombreux et différents. PostgreSQL est un système de base de donnée relationnelle démarré en 1986. L'une des principales qualités de cet hébergeur est le support de différents types de données. Outre des types génériques, PostgreSQL supporte les caractères géographiques et les couches SIG. De plus, l'interface propose des facilités en termes de migration de base de données et aussi de nombreuses combinaisons possibles avec des outils tiers (devx.com, 2004).

L'intégration de données spatiales dans des bases de données classiques est fondamentale de nos jours dans de nombreux domaines. La gestion des risques naturels, l'agriculture ou l'aménagement

d'infrastructures industrielles en sont quelques exemples. Une base de données centralisant les différentes données géospatiales octroi la possibilité de conserver une cohérence et une maniabilité non-négligeable. Cela permet d'augmenter significativement l'efficacité des procédés dans la réalisation des différentes tâches de l'opérateur. Les systèmes d'information géographique sont composés de cinq étapes majeures : l'abstraction, l'acquisition, l'archivage, l'analyse et l'affichage des données. Concernant les enjeux des bases de données, l'archivage représente une étape clé qui permet par la suite l'exploitation des données pour différentes analyses (Solioz, 2006).

QGIS représente un bon compromis en tant que logiciel SIG open source, facile à installer et à utiliser. Le logiciel permet ainsi d'appliquer aisément les différentes tâches génériques et procédés des systèmes d'informations géographiques. En plus des formats classiques (raster et vecteur), d'autres formats de données peuvent être ajoutés et lus par la plateforme. Le logiciel est aussi un atout étant donné ses possibles connections complémentaires et efficaces avec les bases de données spatiales issues de PostgreSQL et Post GIS (Sadoun et al., 2022).

On peut considérer une extension comme un module complémentaire d'un logiciel. Il s'agit donc d'un programme additionnel développé dans l'optique de fournir de nouvelles fonctionnalités sans modifier le code source du programme original (oni-cif.com). QGIS offre la possibilité de produire des extensions en langage python avec le module PyQGIS. Il existe ainsi de nombreux « plugins » disponibles dans le dépôt officiel des extensions QGIS. Une extension sous QGIS est un répertoire regroupant différents fichiers nécessaires au bon fonctionnement de celui-ci. La façon la plus courante de construire un plugin consiste en la création à partir du module Plugin Builder 3, ce qui permet de générer une structure de base de l'extension conforme au logiciel (QGIS Project, 2020 ; Sherman et al., 2005).

III) Méthodes et Réalisations

3.1) Développement d'une base de données

Le choix de la base de données représente un paramètre non-négligeable, impactant fortement l'utilisation et le transfert des informations entre les collaborateurs. Lorsqu'un gestionnaire de donnée est bien implanté dans la structure organisationnelle d'une entreprise ou d'un institut, effectuer une migration vers un autre outil représente un vrai défi. Il s'agit d'une période durant laquelle il convient de continuer à maintenir la base de données actuelle tout en rendant opérationnel le nouveau système de stockage le plus rapidement possible.

3.1.1. Mise à jour des bases de données Access

Access est un système de gestion de bases de données rentrant dans la catégorie de l'informatique personnelle et individuelle. La base de données est représentée par un seul fichier sur disque dur et prend l'extension « .accdb » pour « Access Database » (Formation base de données, 2012).

La mise à jour de la base de données sous Access s'effectue par un procédé d'importation de données à partir de fichiers Excel ou PDF. Les données brutes sont reçues par différents moyens (mails, cloud, etc.) sous la forme de fichiers indépendants. Ces données sont ensuite traitées et mises en forme pour convenir au format des tables que l'on souhaite présenter dans Access. Pour certaines tables, le remplissage manuel des cellules dans la base de données suffisait. Cependant, d'autres tables plus conséquentes, notamment pour la production finale, nécessitaient de réimporter un nouveau fichier Excel pour chaque nouvelle mise à jour (Figure 1).

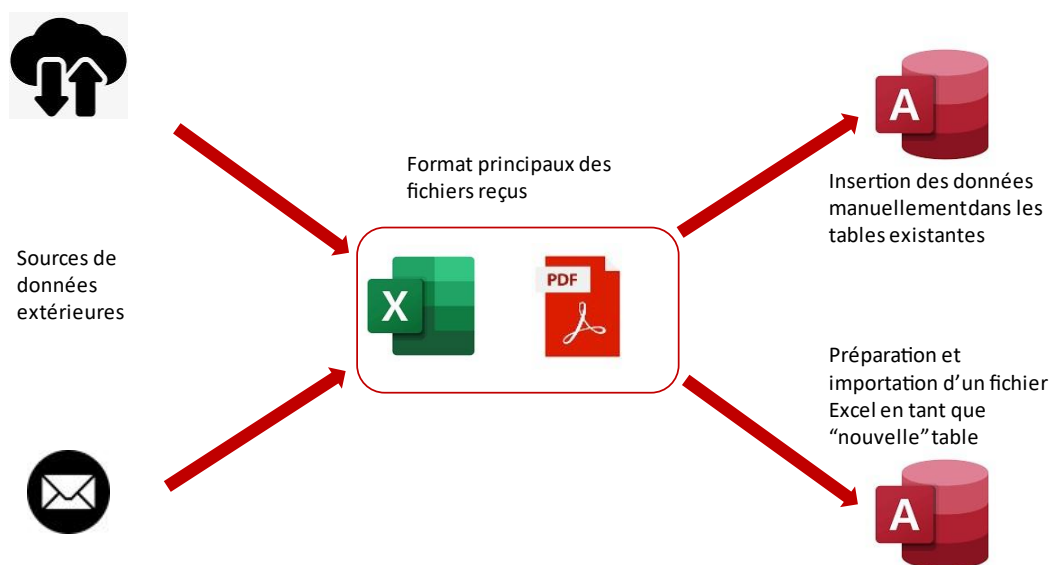


Figure 1. Schéma du procédé de mise à jour des données sous la base de données Access

La mise à jour de nouvelles informations sous Access nécessite plusieurs étapes et la plupart du temps une réimportation complète de la table concernée. Ce procédé peut s'avérer redondant et surtout chronophage, notamment concernant le chargement des données. Dans cette optique, la migration vers une base de données plus directe et plus souple peut s'avérer avantageuse.

La Figure 2 représente l'importation d'un fichier Excel contenant des données relatives aux statuts des puits dans la base de données Access « Well DB ». Comme évoqué précédemment, le fichier Excel doit être proprement formaté pour ne pas créer des soucis de conversion du fichier. À partir de ce point-

là, la mise à jour de la base de données peut se faire de deux manières : On peut choisir d'ajouter les données progressivement ou bien supprimer la table en question et réimporter la version mise à jour. Finalement, le choix de la procédure à suivre dépendra principalement du type de données à traiter, la fréquence et la taille des updates, et enfin des préférences de chacun. Il s'agit des différents facteurs qui participent à la conceptualisation d'une stratégie de maintenance des serveurs de stockages.

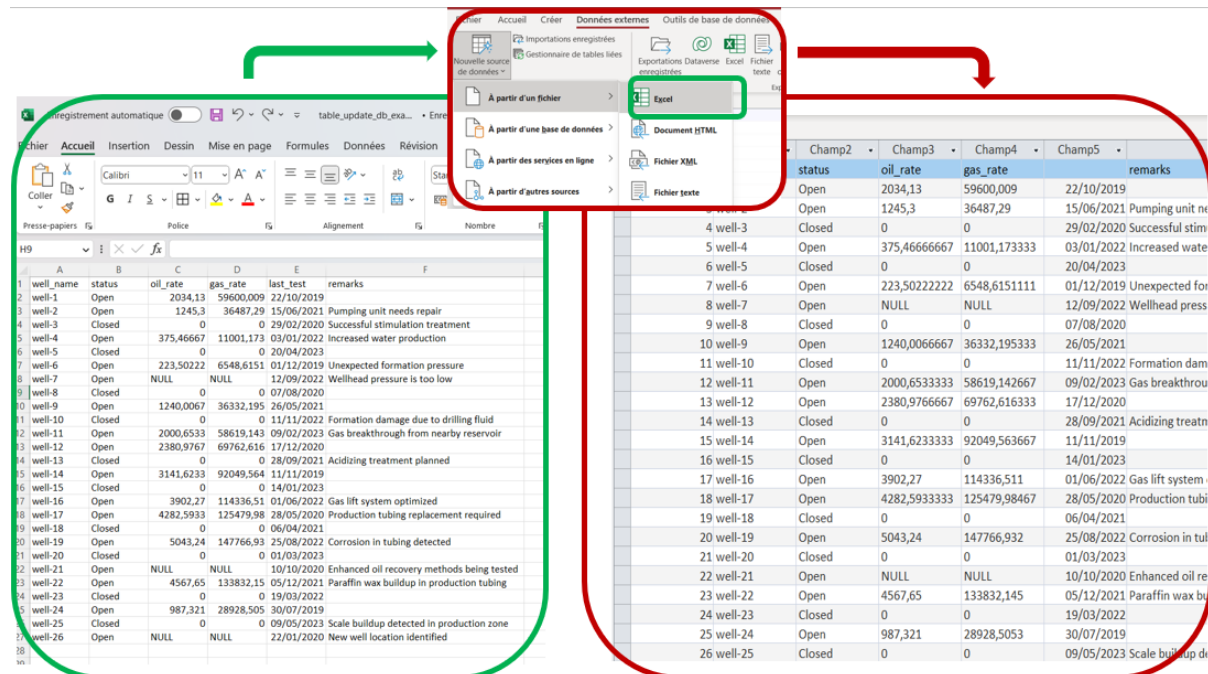


Figure 2. Import d'un fichier excel dans la base de donnée Access (exemple anonymisé)

3.1.2. Mise à jour et migration des bases de données PostgreSQL

PostgreSQL est aussi un gestionnaire de bases de données qui à la différence d'Access, est open source. L'outil suit une logique relationnelle entre client et serveur : il est question d'un gestionnaire des fichiers (PostgreSQL) contenus dans la base de données, qui gère les connexions des applications clientes (programme de l'utilisateur). Les applications clientes permettent d'exploiter les données gérées par le serveur de manières textuelle ou graphique par exemple. Une des principales forces de cet outil est la possibilité de traiter différentes connexions simultanément et permettre à plusieurs utilisateurs de procéder en même temps (The PostgreSQL Global Development Group, 1996-2022).

Le processus de mise à jour des données sur PostgreSQL s'avère plus uniforme compte tenu des types de données et le processus nécessite moins d'étapes que sur Access. Dans un premier temps, les fichiers sont reçus de l'extérieur de la même manière que précédemment. La principale différence réside dans la préparation des fichiers d'importation. Celle-ci s'effectue par l'utilisation d'un script VBA (Visual Basic for Application) qui permet d'appeler automatiquement toutes les nouvelles données dans un fichier Excel. Les informations importées sont automatiquement structurées et rangées dans des colonnes prédéfinies en fonction de la table qui doit être mise à jour. Ensuite, un script en python est exécuté pour alimenter la base de données PostgreSQL des nouvelles données à partir des fichiers au format « .csv » préparés en amont. La Figure 3 schématise le procédé d'actualisation des informations dans la base de données.

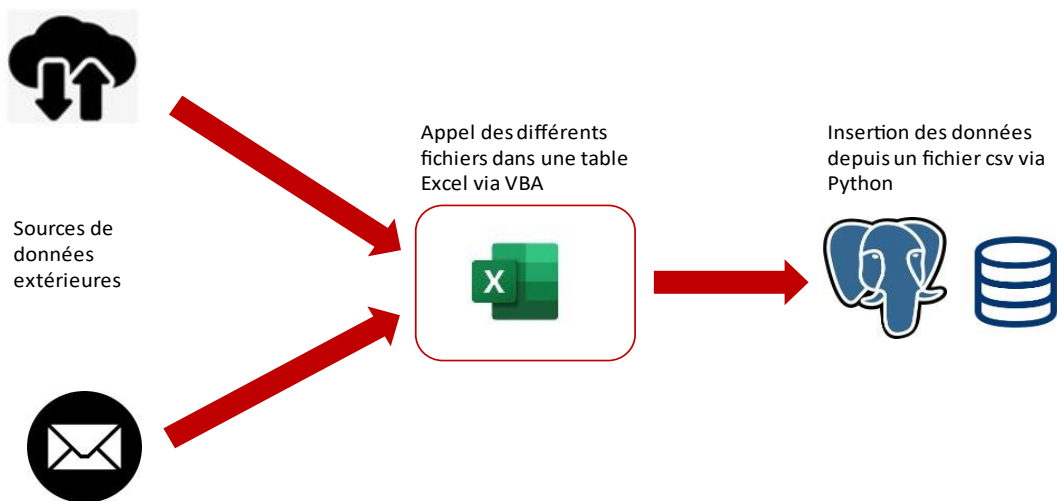


Figure 3. Schéma du procédé de mise à jour des données sous la base de données PostgreSQL

L'actualisation des différentes données sous PostgreSQL est sensiblement plus rapide en termes de temps que dans Access. Outre cet avantage, ce gestionnaire de données s'avère être une alternative judicieuse au vu de sa souplesse et de sa grande capacité de stockage. Les différents utilisateurs peuvent potentiellement obtenir un accès facilité aux données et pour une pluralité d'usage. L'utilisation de cet outil pourrait à l'avenir augmenter significativement la productivité et l'efficacité des différentes tâches nécessitant la mobilisation des informations stockées dans la base de données.

La Figure 4 illustre l'importation des données depuis un fichier CSV dans une base de données PostgreSQL. Ici, la mise à jour des informations passe par l'exécution d'un script python. Ce programme permet de sélectionner les tables que l'on souhaite actualiser. On peut procéder sur plusieurs jeux de données en même temps et il n'est pas nécessaire de réimporter toute la table systématiquement et l'importation peut se faire pendant que d'autres utilisateurs utilisent la base de données. Ce procédé permet de gagner un temps considérable pour la maintenance des informations

```

csv
Nom      Modifié   Type      Taille
field_pressure  12/05/2023 17:38  Fichier CSV Mono...  2 Ko
field_production 12/05/2023 17:38  Fichier CSV Mono...  2 Ko
well_locations   12/05/2023 17:38  Fichier CSV Mono...  2 Ko
well_pressure    12/05/2023 17:38  Fichier CSV Mono...  2 Ko
well_status      12/05/2023 17:38  Fichier CSV Mono...  2 Ko
well_test        12/05/2023 17:38  Fichier CSV Mono...  2 Ko

field_pressure.csv
field_production.csv
well_locations.csv
well_pressure.csv
well_status.csv
well_test.csv

# Open connection to PostgreSQL database and fill tables
#
conn = psycopg2.connect("host=111.222.33.444 port=5432 dbname=field_data user=robin password=passwor")
cur = conn.cursor()

# Store data by calling Functions
#
fill_data(field_pressure, conn, cur, "field_pressure", field_pressure_csv)
fill_data(field_production, conn, cur, "field_production", field_production_csv)
fill_data(well_locations, conn, cur, "well_locations", well_locations_csv)
fill_data(well_pressure, conn, cur, "well_pressure", well_pressure_csv)
fill_data(well_status, conn, cur, "well_status", well_status_csv)
fill_data(well_test, conn, cur, "well_test", well_test_csv)
cur.close()
conn.close()

```

PDC major	double precision	double precision	double precision	double precision	double precision	double precision
65	0	585.701	587.8	579		
66	65	0	582.4	588	605	
67	67	0	584	600	676	
68	68	-00000	678	603	702	
69	69	-00000	671.611	617	685	
70	70	0	614	624	634	
71	71	-00000	627.196	-00000	-00000	
72	72	0	607	668	682	
73	73	-00000	688.823	-00000	-00000	
74	74	-00000	587.847	-00000	-00000	
75	75	-00000	711	617	724	
76	76	0	716.8	728	748	
77	77	0	585.74	593	609	
78	78	-00000	703	740	758	
79	79	0	688.712	704	721	
80	80	-00000	693.92	-00000	-00000	
81	81	-00000	770	778	791	
82	82	-00000	688.4	674	689	
83	83	-00000	678	682		

Figure 4. Import et mise à jour des tables de données dans PostgreSQL via Python (exemple anonymisé)

3.2) Extensions « Plugins » QGIS

3.2.1. Analyse des données spatiales

L'analyse des données passe la plupart du temps par l'élaboration et l'utilisation de tables et graphiques. Cependant, la composante spatiale représente un autre pan très important des données qui est mis en valeur par l'exercice cartographique. Les logiciels de cartographie comme QGIS permettent d'importer des éléments spécifiques contenus dans la base de données. Les informations appelées peuvent être représentées dans une logique cartographique sous réserve de contenir des composantes spatiales.

Il existe différents moyens qui permettent de faire appel aux bases de données sous QGIS. La solution la plus courante consiste à utiliser le manager de base de données implémenté dans l'application. L'interface graphique de QGIS permet d'appeler les données depuis le gestionnaire grâce à l'utilisation de requêtes SQL. Si ces données contiennent des composantes spatiales dans leur table attributaire, elles peuvent être transformées en couche spatiale et cartographiées. Le module PyQGIS est la seconde option permettant d'importer des données depuis la base et de générer des couches fonctionnelles pour QGIS. Ce moyen s'effectue par l'exécution d'un script python effectuant les tâches.

Une autre méthode représente la base de la structure de l'élaboration des extensions. Les différentes fonctionnalités mises en place dans les outils développés s'exécutent à l'aide de scripts en python. Ces derniers définissent les processus et les tâches qui doivent s'effectuer entre la base de données et le logiciel de cartographie. L'utilisateur exécute les différents procédés à l'aide d'une interface graphique facilitant la prise en main des outils développés.

Ainsi, l'extension mise en place présente trois principales fonctionnalités permettant l'utilisation des données. Ces options ont été conçues spécifiquement pour la réalisation de tâches précises. Ces différents processus sont présentés un par un dans les sections suivantes.

Le développement d'un plugin QGIS peut s'effectuer de plusieurs manières. Il est tout à fait possible de programmer une extension à partir du début jusqu'à l'utilisation finale de façon totalement manuelle, c'est-à-dire écrire chaque ligne de code par soi-même et développer l'interface graphique de zéro, sans l'aide d'outils extérieurs. Cependant, une méthode beaucoup plus efficace, permettant de gagner un temps considérable, consiste à s'aider d'instruments et de programmes déjà existants et façonnés spécifiquement pour aider au développement de plugins. À cet effet, le développement d'un plugin sous QGIS nécessite au préalable l'assistance de deux éléments : Plugin Builder et Qt Designer. Plugin Builder est un support qui permet de générer une Template de base pour notre extension, de cette manière, il n'est plus nécessaire de programmer la compatibilité entre QGIS et notre code Python à la création de chaque nouvelle extension (qgis.plugins.com). Ensuite, Qt Designer est un outil très pratique dans la réalisation d'interfaces graphiques. Ce programme permet de facilement conceptualiser et de personnaliser différentes fenêtres comprenant des boutons de commandes, des zones de textes, des menus déroulants, et d'autres options qui permettent à l'utilisateur d'exécuter les différentes fonctionnalités du plugin QGIS (Willman, 2022). À partir de là, le développement d'un plugin ne nécessite plus qu'au créateur de programmer en langage python les différents processus et tâches à réaliser. Ainsi, une fois que le code est écrit et que l'interface graphique est prête, l'outil devrait être fonctionnel et prêt à l'utilisation.

3.2.2. Data Viewer

Data Viewer est la première extension développée. Cette extension a été élaborée dans l'optique d'optimiser la sélection des données au maximum. Il convenait de pouvoir dynamiser la sélection d'une information spécifique, à une date précise et par une symbologie efficace. Le but principal du

Data Viewer est de pouvoir couvrir la visualisation du plus grand nombre de données possible. Il était nécessaire dans un premier temps de permettre à l'utilisateur de subvenir à ses besoins de visualisation de la manière la plus efficace possible.

L'utilisation de ce premier utilitaire s'effectue en quelques étapes. Dans un premier temps, il est nécessaire de se connecter à la base de données directement depuis l'interface graphique pour s'assurer que les données privées ne soient pas accessibles à n'importe qui, par soucis de sécurité et confidentialité. La connexion engendre l'ouverture du projet de travail partagé QGIS et génère automatiquement une couche vecteur point des différents puits plus les informations concernant leur type et leur statut à la dernière date entrée dans la base de données SQL (Figure 5). Le collaborateur peut donc obtenir directement un état des lieux de la situation sur le champ.

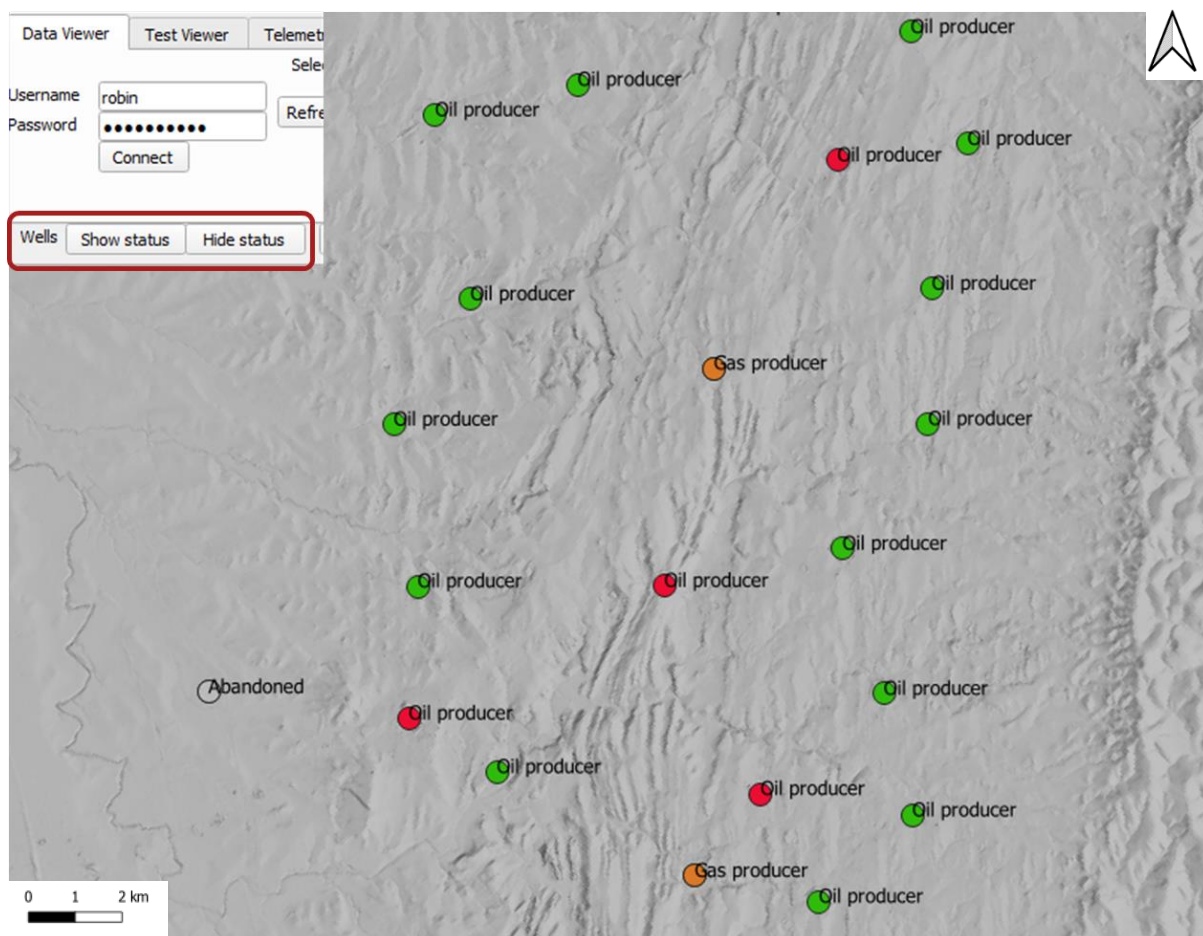


Figure 5. S statut des puits à la dernière date entrée dans la base de données et de leur type

Ensuite, la fenêtre est décomposée en différents boutons qui pourraient s'apparenter aux arguments que l'on pourrait utiliser dans une requête SQL. On retrouve deux boîtes de dialogues qui servent à sélectionner la table d'une part, et le champ à cartographier d'autre part. Les différents champs d'une table sont générés via un bouton qui exécute une première requête. Une fois la donnée d'intérêt choisie, il faut sélectionner la date à laquelle nous souhaitons analyser la donnée par le biais d'un sélecteur de date. Il est aussi possible de personnaliser la symbologie du rendu : la représentation cartographique s'apparente à des bulles de tailles variables selon la valeur pour les données numériques. De ce fait, l'utilisateur peut choisir la taille des bulles et le nombre de catégories de taille qu'il souhaite. Dans le cas de données textuelles, la couche générée affichera les éléments sous la

forme des labels. La Figure 6 représente la « Bubble Map » de production par puits au 1^{er} janvier 2023 générée depuis l'extension et l'interface graphique du Data Viewer.

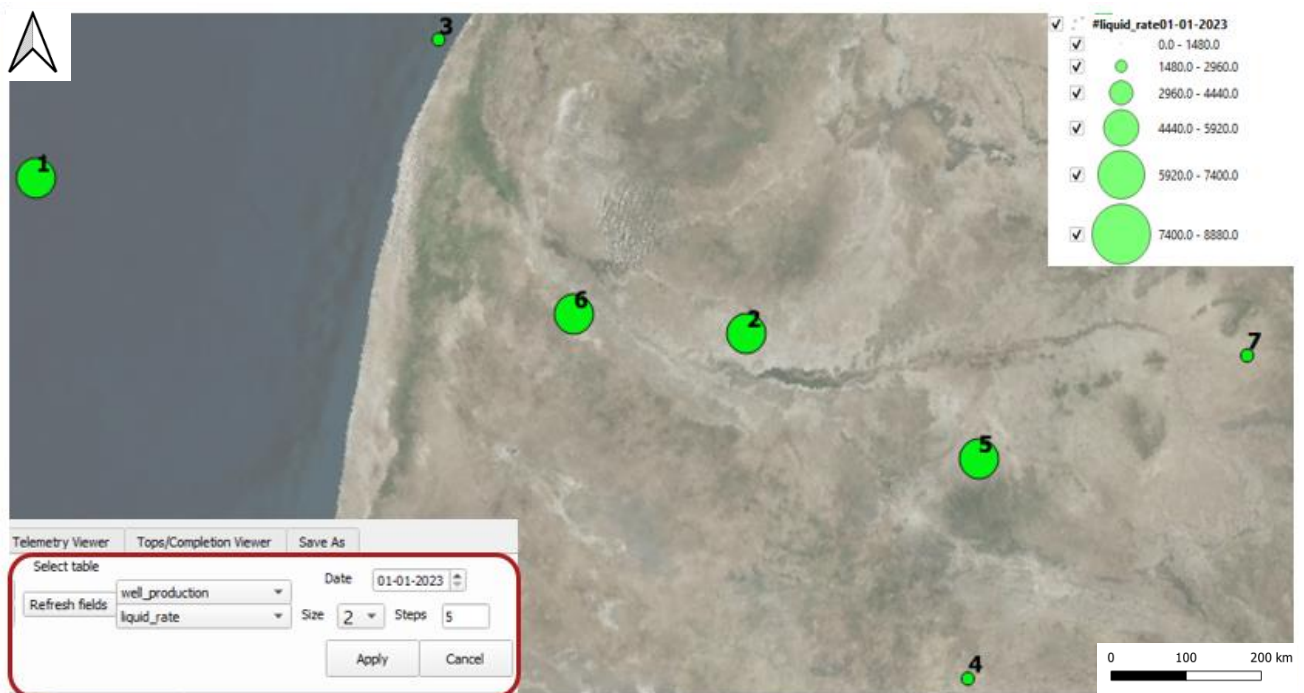


Figure 6. Cartes de bulles de la production au 01/01/2023 par puits avec une symbologie customisée

L'outil dispose aussi d'autres fonctionnalités annexes permettant de faciliter l'analyse pour quelqu'un de peu familier avec l'interface de QGIS notamment. On retrouve un bouton permettant de générer la table attributaire de la couche spatiale récemment créée. La composition de la table générée par cette option représente le même résultat qu'aurait conçu l'exécution d'une requête SQL classique. Une option pour la création de graphique est aussi présente, elle permet d'afficher depuis la donnée d'intérêt (Figure 7) :

- Un graphique en barres par puits pour une date donnée
- Un graphique d'aire par date pour un seul puit

Les graphiques sont directement produits depuis la donnée sélectionnée au préalable dans les menus déroulants. Ils donnent l'opportunité d'avoir un aperçu rapide des dynamiques liées au champ d'intérêt d'un point de vue comparatif ou temporel.

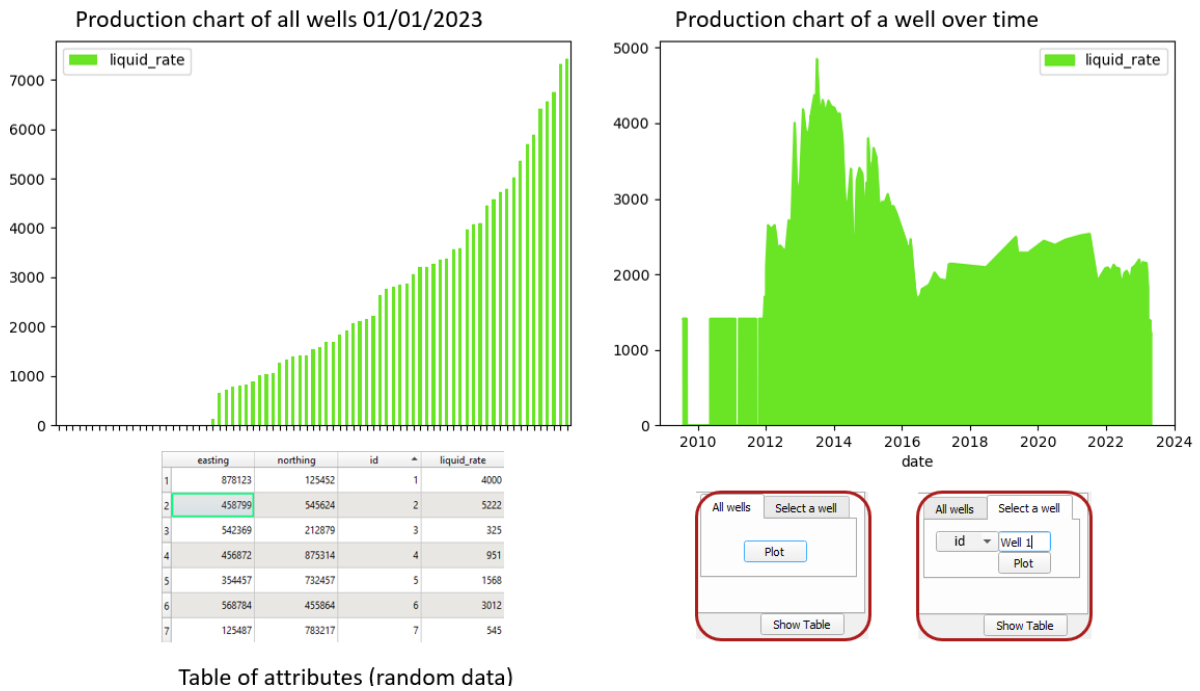


Figure 7. Graphiques et table caractéristiques des données de production (QGIS)

Il est aussi possible de générer la localisation d'un puits (couche point) à partir de l'identifiant du puits ou de générer la localisation de tous les puits en même temps. Cette option permet d'illustrer la position des différents puits de manière lisibles et épurée sans s'encombrer d'autres informations (Figure 8).

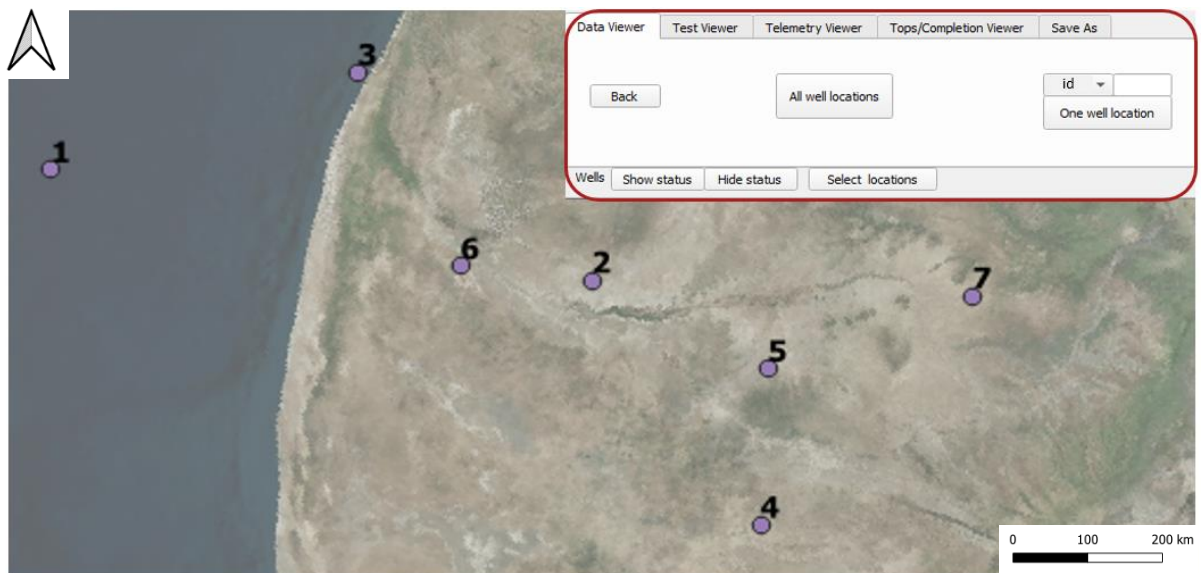


Figure 8. Couche vecteur point de la localisation des puits

Le Data Viewer est la principale fonction de l'extension QGIS développée. La majorité des informations présentes dans la base de données peuvent être mobilisées grâce à cet outil. Cependant, le format des éléments stockés peut varier du fait de la variété et de la complexité variable des informations. Il est probable que quelques tables ne puissent pas être activées via cet instrument. Les prochaines parties présentent les outils auxiliaires développés et leur rôle pour améliorer l'efficacité du plugin QGIS.

3.2.3. Test Viewer

Le Test Viewer est le second outil qui compose le plugin QGIS. À la différence du Data Viewer, il permet de mettre en valeur de façon cartographique un certain type de donnée. En effet, ce module se base sur la mise en avant des données en provenance des différents tests effectués sur les puits. Les tests ne sont pas effectués de façon journalière sur les puits et suivent une trajectoire relativement randomisée. Le problème pour l'utilisateur, à ce niveau, réside dans le fait de pouvoir sélectionner un test spécifique à afficher sans avoir connaissance de la date de réalisation de celui-ci au préalable. Il convenait donc de trouver une façon de faire apparaître l'information sans pour autant pouvoir prendre la date comme référence.

De ce fait, le Test Viewer propose deux options principales qui permettent de générer la valeur du dernier test en date par puit/pour tous les puits, sur une plage temporelle spécifiée. L'utilisateur peut définir la période de son choix avec deux boîtes de dialogue permettant d'entrer une date de début et de fin. Il doit aussi spécifier la valeur de quel type de test il souhaite représenter et choisir entre un puits ou tous les puits en même temps. La Figure 9 illustre le ratio pétrole-gaz du dernier test de chaque puits entre le 1^{er} janvier et le 31 décembre 2023.

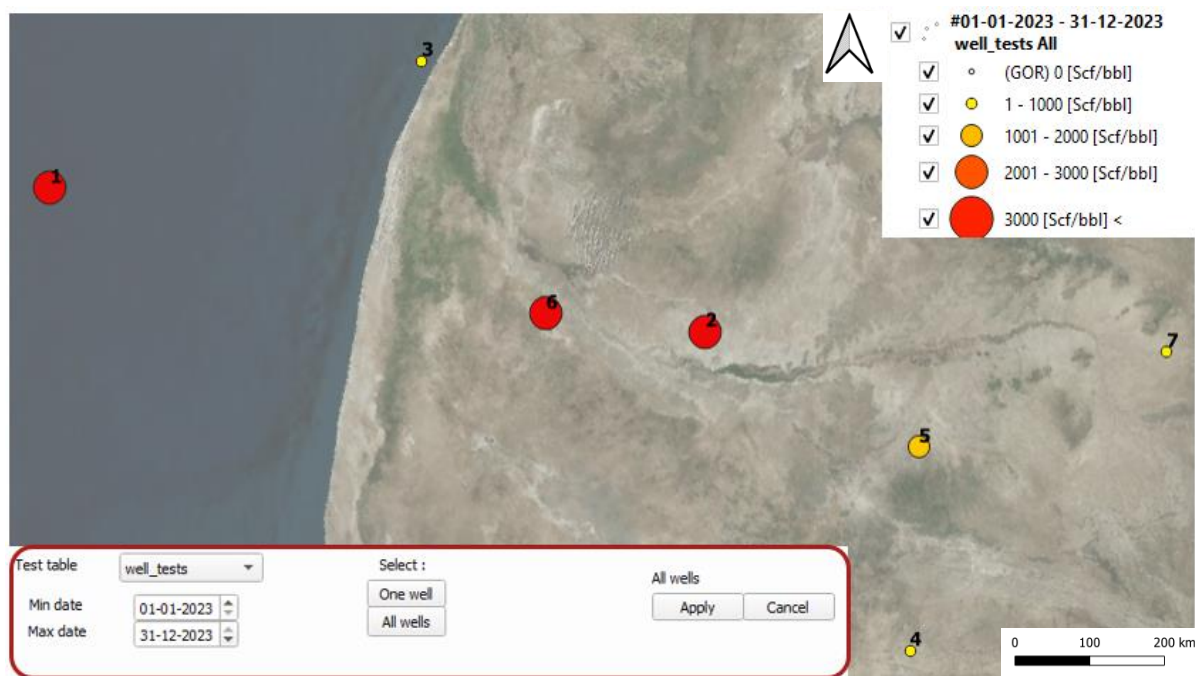


Figure 9. Carte en bulle du ratio pétrole-gaz du dernier test de chaque puit

Cet outil permet aussi de générer un graphique temporel en nuage de point combiné par puits. Le collaborateur peut choisir la plage temporelle de son choix et le graphique comprend toutes les valeurs des deux champs principaux provenant des mesures des tests : pourcentage d'eau et ratio gaz-pétrole (Figure 10).

Scatter plot of water percentage and oil to gas ratio

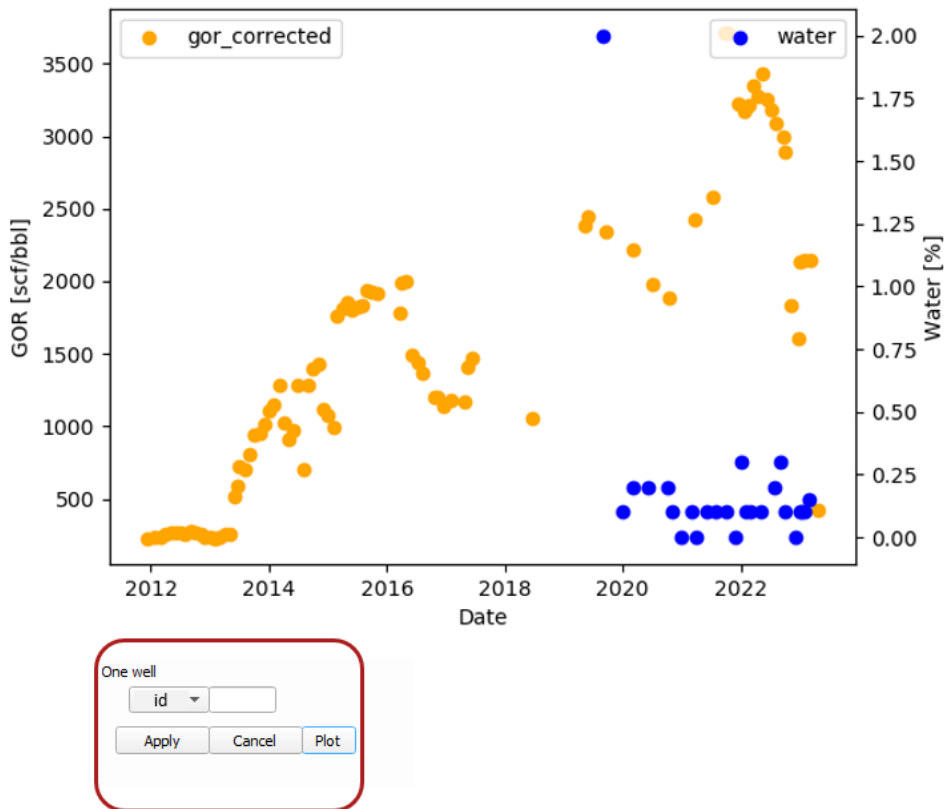


Figure 10. Nuage de point des pourcentages d'eaux et du ratio pétrole gaz pour un puit

Ce deuxième module à la différence du premier n'est pas développé dans une logique d'accès global aux données, mais il répond à un type d'information et un format spécifique. Le troisième outil du plugin rentre aussi dans cette logique.

3.2.4. Telemetry Viewer

Le dernier instrument qui compose l'extension est le Telemetry Viewer. Dans la continuité du module précédent, il a été élaboré de sorte à traiter les données possédant un format différent du reste de la base de données. Les données de télémétrie sont des mesures de pressions et de températures prises toutes les 30 minutes sur chaque puit. Elles représentent donc un nombre très abondant de données à stocker et nécessitent des moyens conséquents pour les analyser. Le jeu de donnée de télémétrie ne comporte pas une colonne date, mais un champs « date time » qui renvoie au format YYYY-MM-DD hh:mm:ss.

Pour optimiser la représentation cartographique des données de télémétrie, deux possibilités temporelles ont été mises en place. Dans un premier temps, il est possible de choisir d'afficher sur la carte la pression à la tête du puit, à la ligne d'écoulement ou la température. La première option est de visualiser une couche de la dernière mesure enregistrée dans le serveur par puit. Ce procédé permet de mettre en avant la situation quasi actuelle sur le champ, tant que la base de données est tenue à jour. La seconde méthode possible est de générer une « Bubble Map » des moyennes de pressions/températures sur une période données. De la même façon que le module précédent, il est possible de choisir entre deux dates et de générer une moyenne des différentes valeurs sur une échelle de temps personnalisée (Figure 11).

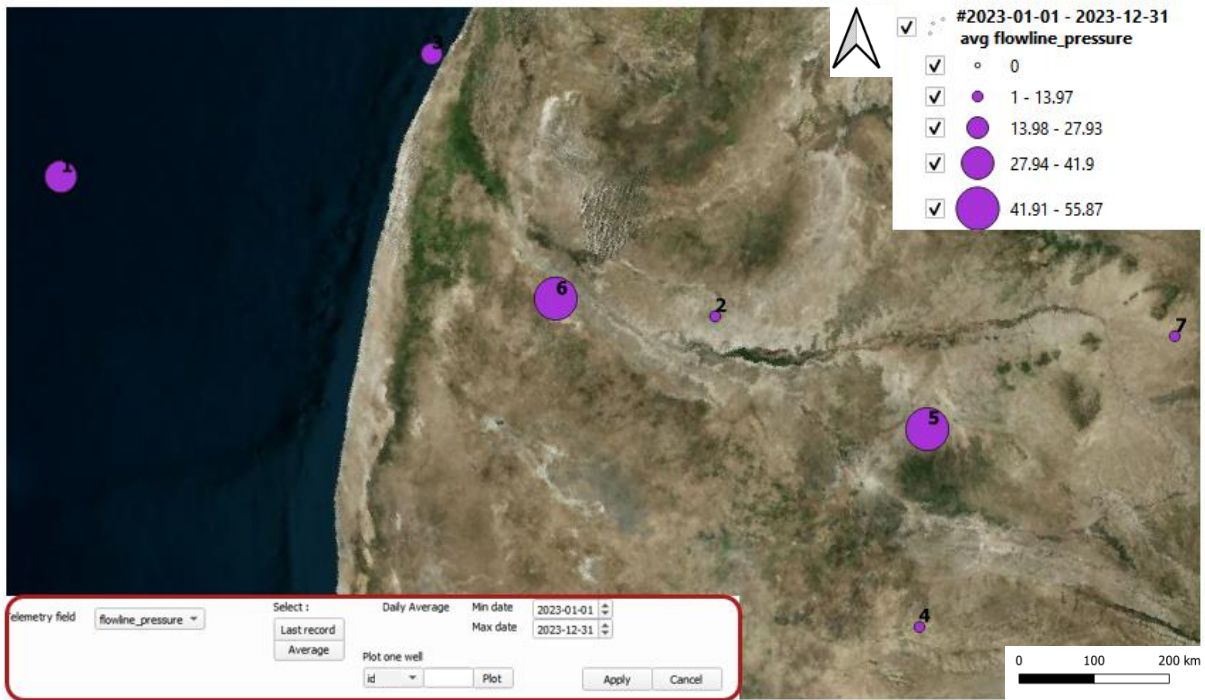


Figure 11. Carte en bulle de la pression moyenne par puits depuis le 1er Janvier 2023

Enfin, à l'image du Test Viewer, ce module propose aussi une fonction annexe permettant de générer un graphique en nuage de point. Le graphique correspond aux différentes mesures de pressions sur une échelle de temps customisée pour un puits défini par l'utilisateur. Le graphique est très important ici, car il permet de mettre en avant l'évolution temporelle des mesures et d'identifier les différentes dynamiques. Il complète en quelques sortes la représentation cartographique (Figure 12).

Scatter plot of flowline and wellhead pressure

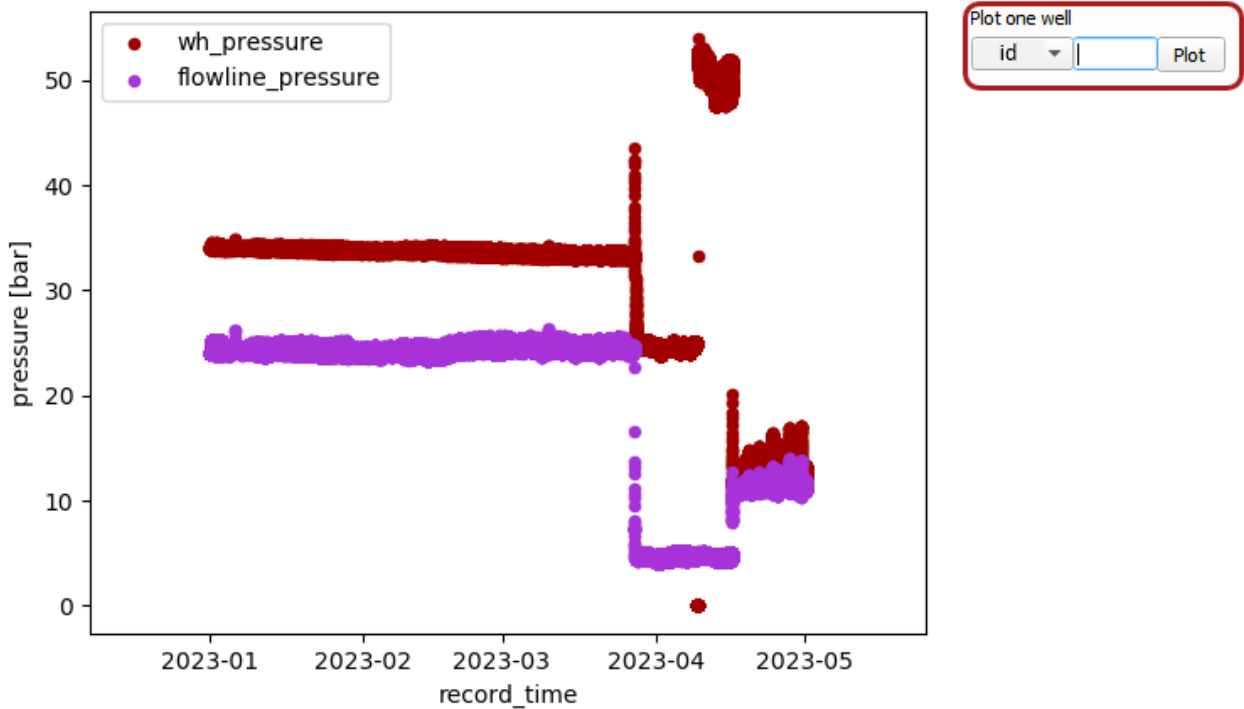


Figure 12. Nuage de point des mesures de pression pour un puit

Les trois modules présentés permettent de mobiliser et d’englober quasiment l’ensemble des jeux contenus dans la base de données PostgreSQL. L’interface graphique regroupe tous les outils à disposition du collaborateur. Centraliser et simplifier l’importation des informations facilitent grandement le processus de génération de couches cartographiques. Cependant, la finalisation d’un projet cartographique nécessite souvent la création d’une carte et d’une mise en page qui ne doivent pas être négligées.

3.2.5. Générations de cartes

Dans l’optique de rendre le plugin le plus complet possible, il était important de proposer une option de mise en page et d’exportation intéressante pour les utilisateurs. Ainsi, un dernier module a été élaboré pour cet objectif. Il s’agit d’un module très automatisé, il suffit de sélectionner un chemin d’accès dans lequel sauvegarder le résultat. À partir de là, le programme va automatiquement générer une mise en page « classique » comprenant une carte, une légende, une échelle et un titre en référence aux couches activées dans le projet QGIS. La carte est exportée au format PDF avec le nom de la principale couche spatiale comme identifiant. La Figure 13 présente une carte générée à partir de ce module.

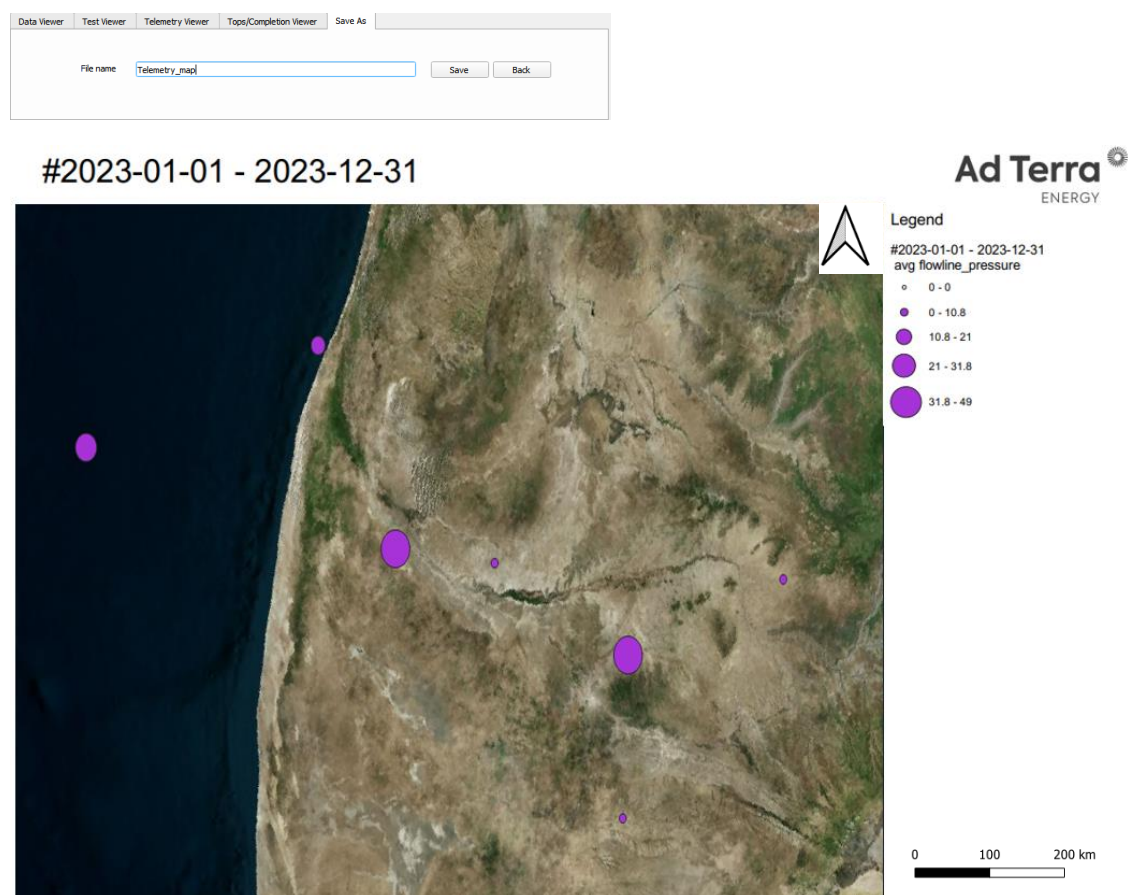


Figure 13. Carte de télémétrie et mise en page générée automatiquement

La possibilité de préparer de manière automatique des mises en page cartographique permet d’intégrer un plus grand nombre d’utilisateurs. Les collaborateurs, même ceux ayant moins d’aisance avec le logiciel QGIS, pourront préparer leurs cartes en fonction de leurs besoins sans au préalable nécessiter trop de connaissances spécifiques des SIG au préalable. Ceci aura pour effet de permettre d’une part un gain de temps considérable pour les usagers, et d’autre part une intégrité accrue du personnel. C’est-à-dire que la facilitation d’accès aux représentations spatiales pour le plus grand nombre ouvrira plus de perspective d’analyses et de présentations avec l’ajout de ce nouveau support.

3.3) Restructuration de l'ancienne géodatabase

3.3.1. La base de données spatiale existante

Le troisième projet effectué durant mon stage chez Ad Terra Energy se focalisait sur la restructuration et la mise à jour des différentes données spatiales à disposition. En effet, l'entreprise possédait une base de données conséquente d'information relatives aux différents clients stockées sur ses serveurs. Cependant, ce répertoire important de données comprenait beaucoup d'éléments qui commençaient à dater et qui pour certains s'avéraient obsolètes du fait qu'ils n'avaient pas été maintenus à jour depuis des années.

La mise à jour des données à disposition a représenté un enjeu important de cette restructuration. Cependant, l'objectif recherché n'était pas d'actualiser tous les jeux de données spatiaux présents dans les répertoires. En effet, beaucoup de données se sont avérées utiles et nécessaires dans la réalisation d'anciens projets, mais ces informations n'étaient pas nécessairement toutes pertinentes dans le cadre des nouveaux projets et de travaux actuels. C'est dans cette optique que le deuxième enjeu principal de ce travail a été de sélectionner principalement les éléments importants et pertinents pour la restructuration. Cette tâche est essentielle pour mettre en place une organisation des données qui répond aux besoins, mais qui reste claire et lisible pour permettre aux collaborateurs d'accéder à ce qu'ils cherchent plus efficacement.

3.3.2. Création d'une base de données spatiale

La nouvelle base de données a pour objectif de répondre aux besoins des usagers. En d'autres termes, elle doit être structurée de manière claire et logique, mais aussi permettre de stocker toutes les informations spatiales essentielles liées au projet en question. La base de données a été réalisée sous la forme d'un répertoire général contenant différents sous-répertoires qui stockent des informations triées en fonction des types de fichiers et de la thématique à laquelle ils renvoient. Le projet principal QGIS est aussi présent au début de l'arborescence des répertoires, ainsi que les informations générales du projet relevant du domaine des SIG.

On retrouve à la source quatre dossiers principaux contenant l'ensemble des informations. Le sous-répertoire « Archive » contient les anciens éléments généraux directement reliés au projet et qui ont connu une actualisation par la suite. Le sous-ensemble « Layer » représente le cœur des informations SIG stockées dans la base de données, il fera l'objet d'un développement plus approfondi ci-dessous. L'onglet « Plugins » contient les différentes extensions (QGIS principalement) implémentées et utiles au travail d'analyse spatial, l'extension présentée plus tôt est aussi disponible dans ce répertoire. Enfin, la dernière option « Temp » fait référence aux différentes couches temporaires qui ont été générées dans le cadre de processus et d'analyses des données spatiales. Il s'agit donc de couches pas nécessaires en soi, mais qui peuvent s'avérer utiles, notamment lors de géo-traitements.

Le corps principal de la base de données se situe donc dans le sous-répertoire « Layers ». Il est décomposé en quatre branches qui regroupent des informations du même type. On retrouve dans un premier temps un espace de stockage pour toutes les informations liées à la géologie. Différents types de fichiers tableurs (.csv, .xlsx), vecteurs (.shp + extensions de caractéristiques) liés à la géologie du terrain sont présents tels des données de sous-sols (cartes structurales, etc.) ou des cartes d'épaisseurs. Dans un registre similaire, la seconde sous-catégorie permet de stocker toutes les données relatives aux cartes paléontographiques/paléontologiques. Il s'agit de cartes illustrant des espaces et environnements du passé en prenant en compte les différents éléments associés comme la topographie, le climat ou la géologie (Scotese, 2016). Ce dossier comporte les mêmes types de fichier que pour la catégorie géologie. Un dossier est consacré au stockage des données liés aux différents puits, il contient les couches SIG au format vecteur des puits actuellement présents dans le

projet et de ceux en planification futurs. Ces données correspondent donc aux coordonnées des puits, mais comprennent aussi différentes caractéristiques de ces structures dans leur table attributaires. Enfin, la dernière et majeure sous-catégorie de cette arborescence est celle où l'on retrouve toutes les données relatives à la surface. Ces informations sont de natures diverses et variées. En effet, on retrouve des éléments relatifs à tout ce qui se passe sur le domaine du projet. De ce fait, les couches peuvent concerner les administrations, routes, infrastructures, populations, altitudes, cours d'eau, et d'autre. On retrouve donc des données aux formats SIG les plus courantes, de manière à être le plus accessible possible : des couches vecteurs et rasters accompagnés de leurs extensions caractéristiques. Enfin, c'est dans ce répertoire quel' on retrouvera un DEM (digital elevation model ou modèle numérique d'élévation) de la zone correspondante au projet. La Figure 14 schématise l'arborescence de la nouvelle géodatabase restructurée.

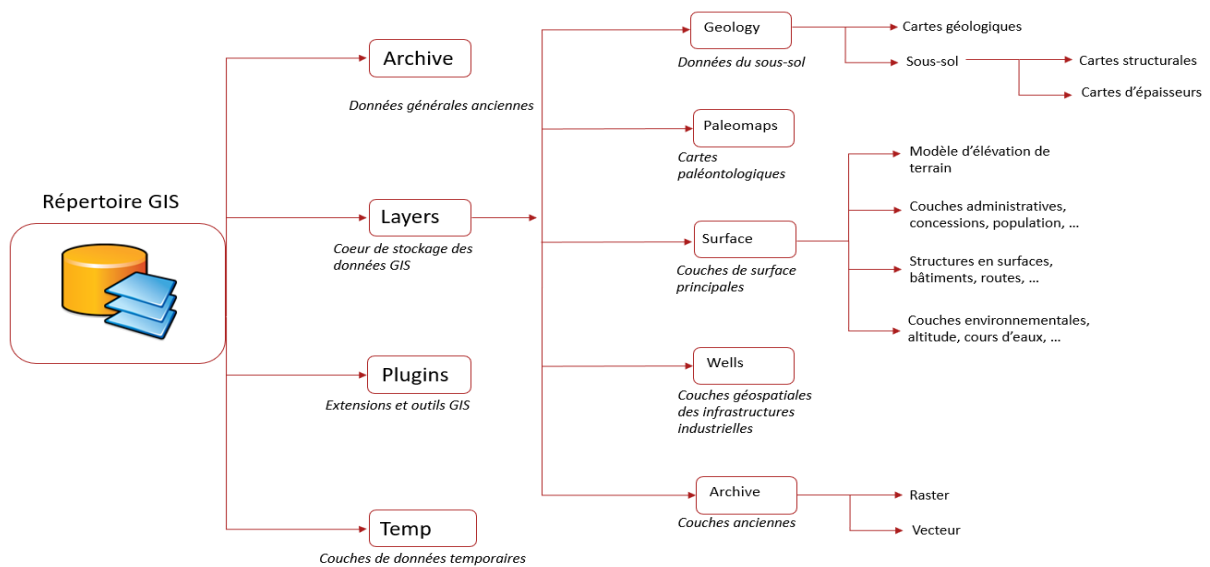


Figure 14. Arborescence de la base de données spatiale

La base de données spatiale est structurée en différents dossiers qui eux-mêmes, sont composés de sous-dossiers stockant les différents fichiers du projet. L'organisation des éléments est élaborée de sorte à rendre la recherche des informations intuitive pour les collaborateurs. L'arborescence est ainsi construite dans une logique similaire aux autres répertoires de données.

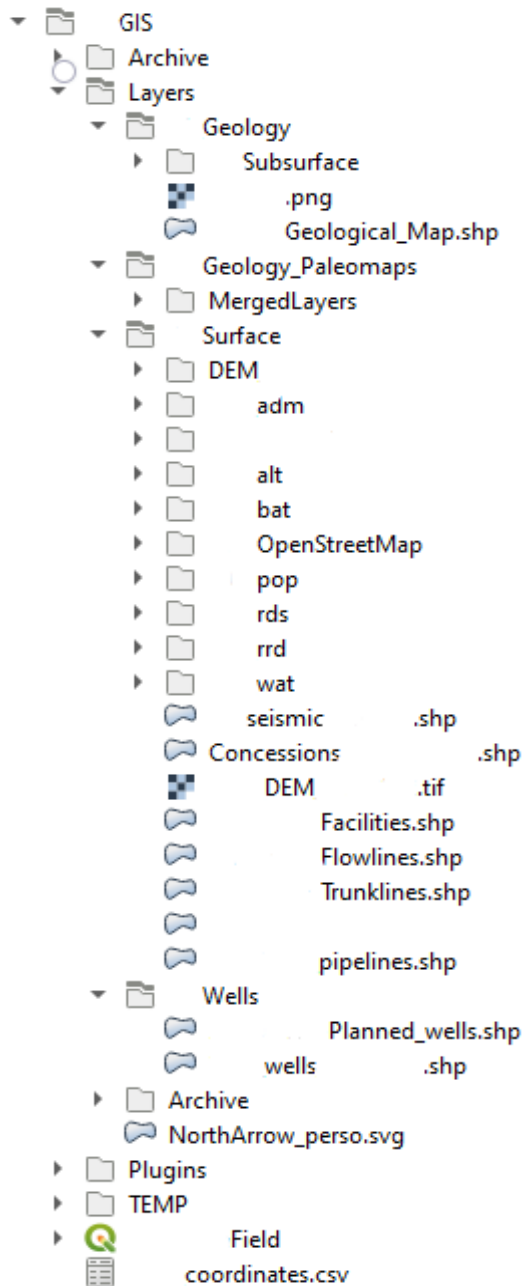


Figure 15. Arborescence de la base de données sous le catalogue QGIS (à noter que certains noms ont été masqués pour des raisons de confidentialités)

Conceptualiser la base des données spatiales de manière rigoureuse permet de faciliter son utilisation dans les logiciels SIG. La Figure 15 représente la structure des répertoires de stockages déployés sous le logiciel QGIS. En ouvrant un catalogue de recherche, on peut étendre les différentes sous-catégories et importer de façon rapide et efficace les éléments nécessaires. Une structure logique et compréhensible permet aux utilisateurs de s’y retrouver lorsqu’ils naviguent au travers les documents. Dans cette figure, on retrouve les différents dossiers présentés auparavant. Le chemin principal « Layers » est déployé, on peut apercevoir les différentes entités qui le composent. Ainsi, les dossiers mentionnés dans le paragraphe précédent sont bel et bien présents de la même manière qu’ils ont été créés dans le répertoire du projet. Mis à part QGIS, ce genre d’infrastructure peut être mis en place et mobiliser dans d’autres logiciels et systèmes d’information géographique (ex : ArcGIS). C’est aussi pourquoi la mise en place d’une architecture claire et logique est importante dans la création de répertoire de stockage de données spatiales. Il s’agit d’un procédé qui s’avère très fréquemment utile et nécessaire.

La restructuration de la base de données spatiale est la dernière tâche majeure qui a été réalisée durant ce stage. La prochaine partie de ce rapport se focalise sur une conclusion des réalisations effectuées. Les travaux présentés ici sont discutés et découlent sur une réflexion plus globale sur le déroulement du stage.

IV) Conclusion

4.1) Conclusion des réalisations

4.1.1. Bilan des réalisations

Dans la partie précédente, les différentes réalisations du stage ont été présentées et détaillées. Les différentes tâches ont été effectuées dans le cadre d'un projet plus important de migration d'une base de données spatiale. La totalité des travaux avaient pour but d'améliorer significativement l'efficacité et la facilité d'utilisation des ressources stockées pour les collaborateurs. À l'origine, les données étaient disponibles depuis une base de données Access. Cet outil présentait des avantages d'organisation et de centralisation certains, mais comprenait aussi certaines limites liées à la gestion des quantités d'informations importantes et de flexibilité d'utilisation. Dans un contexte où la compagnie développe de plus en plus de projets conséquents, il était nécessaire de pouvoir déployer des ressources permettant d'optimiser le flux croissant des informations. Il a été convenu que la migration de la base de données existante représentait un enjeu fondamental de cette transition. PostgreSQL semblait être un bon compromis entre l'utilisation d'un outil gratuit et l'automatisation des tâches de maintenance et de développement de la base de données. Ainsi, les travaux réalisés ont tourné autour de trois projets principaux. La première mission a été d'accompagner la migration des informations d'Access à PostgreSQL. Il a fallu commencer à remplir et à enrichir la nouvelle base de données tout en assurant la transition et la maintenance à jour de l'ancien outil de stockage. Même en considérant l'hypothèse qu'il aurait pu être possible de transférer les éléments d'un coup, les outils et méthodes de travail devait aussi être actualisés pour être compatibles avec les nouveaux logiciels utilisés. La plupart des applications et programmes de l'entreprise n'étaient pas confectionnées pour être fonctionnelles avec la nouvelle base de données relationnelles. D'autre part, la migration a demandé du temps pour aussi permettre aux collaborateurs de s'acclimater et d'apprendre à utiliser les nouveaux outils. Au terme de ce stage, on peut soulever quelques bilans. Aujourd'hui, la nouvelle organisation commence à être bien établie dans l'environnement de travail. Les informations principales sont bien présentes et leurs mises à jour quotidiennes sont assurées sur PostgreSQL. Les collaborateurs se détournent aussi petit à petit de l'ancien portail Access et viennent chercher les éléments qu'ils souhaitent sur la nouvelle plateforme de plus en plus, par le biais de différents outils. Cependant, on ne peut pas encore dire que la migration est totalement complétée. Il est fondamental de continuer à familiariser les utilisateurs aux nouveaux processus. De plus, ce nouveau centre de stockage est en perpétuelle expansion, on observe des nouveaux types d'information au fil du temps qu'il convient d'intégrer continuellement sur PostgreSQL, ce qui représente le nouvel enjeu pour les temps à venir.

Outre la migration de la base de données, le projet qui a nécessité le plus de temps fut la création d'une extension pour le logiciel QGIS. Il était fondamental pour ceux qui avaient besoin d'avoir accès aux informations de pouvoir leur fournir des solutions accessibles et fonctionnelles. Le développement de cette extension s'est présenté comme une option intéressante. QGIS en tant que logiciel libre et flexible, permet de générer des applications tierces fonctionnelles sur son interface. Ainsi, il a été possible de mettre en œuvre une solution gratuite et qui ne nécessitait pas la création d'un programme à partir de zéro. Les usagers avaient donc seulement besoin d'installer QGIS sur leur ordinateur pour récupérer les données par l'utilisation d'une interface graphique intuitive, qui ne nécessitent à aucun moment un contact direct avec l'outil PostgreSQL. La création du « Data Viewer » et ses sous-fonctionnalités a demandé beaucoup de temps et d'investissement. Il a fallu réfléchir longuement sur la façon de produire l'outil pour qu'il soit le plus efficace pour les utilisateurs et facile d'accès. C'est dans ce contexte que les différentes fonctionnalités présentées précédemment ont été mis en place en fonction des priorités et des besoins des employés. En seulement quelques manipulations, il est devenu très aisé d'accéder aux éléments que l'on recherche. L'avantage de QGIS

réside aussi dans sa composante spatiale. Les cartes en bulles, notamment, communiquent des informations et des indices des dynamiques quasi instantanément, d'un simple coup d'œil. Il est ensuite possible d'approfondir l'analyse par le biais des tableaux et des graphiques qui ont aussi été programmés pour se générer automatiquement d'un simple clic. L'outil a été adopté assez rapidement par quelques usagers peu de temps après sa présentation. Il tend à se déployer encore plus dans le temps après une certaine période de familiarisation avec son fonctionnement. L'extension développée ne prétend pas opérer comme une réelle interface de centralisation des données, à l'image des logiciels dbMap ou Sahara. En effet, il existe sur le marché des applications optimisées dans l'industrie pétrolière, qui ont pour but de centraliser, analyser et échanger les données de différents types, comprenant aussi les données de composantes spatiales (Petrosys, 2019 et Interfaces, 2020). L'instrument développé sous QGIS ne possède pas autant de ressources que ce genre de programmes, mais il permet tout de même d'effectuer d'obtenir un accès rapide et simple aux informations disponibles, ce qui constitue un outil efficace pour la réalisation des premières analyses. L'extension devrait connaître par la suite des modifications qui entraîneront l'ajout de nouvelles fonctionnalités selon le retour des utilisateurs. Les feedbacks seront essentiels pour permettre de continuer à déployer le potentiel de ce type d'outil.

Enfin, la restructuration de la base de données spatiale s'est avérée importante pour plusieurs raisons. Dans un premier temps et comme expliqué auparavant, beaucoup d'informations stockées étaient devenues obsolètes ou non pertinentes. L'ancien répertoire fournissait une quantité de données très conséquente et il était difficile de s'y retrouver. Ces deux raisons ont entraîné peu à peu la marginalisation du recours à cette base de données. Les employés qui souhaitaient utiliser des couches de données spatiales ont commencé à construire leur propres répertoires plus petits, spécifiques et compréhensifs en lien avec leur projet. Dans cette optique, il ne semblait pas cohérent de remettre à jour un immense stockage de données spatiales dont la plupart n'aurait été d'aucune utilité pour les nouveaux projets. De ce fait, le travail était focalisé sur la restructuration de la base de données en ne prenant en compte que les informations qui faisaient sens pour les projets actuels. Il a été nécessaire de parcourir les dossiers et de veiller à identifier les fichiers corrompus et les doublons. Ce travail, fastidieux au premier abord, est très utile pour permettre de se rendre compte et d'identifier l'état des éléments à disposition. Il reste alors à proposer une nouvelle architecture de dossier qui permet d'accéder le plus facilement possible aux dits éléments. Ce travail est aussi important pour se rendre compte de ce qui manque et qui serait essentiel d'avoir dans la base de données. Ainsi, la restructuration de l'instance de stockage a donné l'opportunité de se mettre à la recherche d'acquisition de nouvelles informations utiles qui ne sont pas présentes dans les données existantes. À ce jour, l'arborescence principale de la base de données est bien établie et les utilisateurs peuvent s'y retrouver facilement. Il conviendra par la suite de continuer à mettre en œuvre une stratégie d'acquisition des éléments pour permettre d'enrichir efficacement les renseignements à disposition.

4.1.2. Perspectives pour l'entreprise

La migration de la base de données effectuée dans la compagnie n'est pas simplement une question de préférence de logiciel d'exploitation. Ce projet se traduit dans une dynamique plus importante au sein de la compagnie elle-même. En effet, bien que ce projet ne représente qu'un faible élément d'un tout plus conséquent, il traduit une volonté de changement dans l'organisation d'Ad Terra Energy. L'entreprise commence peu à peu à acquérir plus de projets et continue de se développer considérablement. De ce fait, il est plus que primordial d'optimiser au maximum la gestion des projets, l'échange des informations et l'automatisation des tâches. Plus les flux de travaux sont importants et plus il est nécessaire d'avoir une organisation rigoureuse pour garder le contrôle des différentes dynamiques. C'est aussi dans cette optique que s'inscrit la migration de la base de données. Cette

transition permet de passer d'une structure plus rigide et fermée à une organisation où les données se veulent plus accessibles, compatibles avec des programmes tiers et faciles à maintenir à jour. De plus, malgré cette souplesse accordée par PostgreSQL, les informations restent rigoureusement protégées et ne sont accessibles à l'édition que par des employés spécifiques et les niveaux d'accès aux données par les personnes sont gérés par les administrateurs spécialisés. On se retrouve donc avec un outil performant et sécurisé qui est capable de supporter et d'accompagner la croissance des projets et des missions au sein de la compagnie. Il s'agit d'une première étape qui pourrait bien n'être que le début de la mise en place d'autres projets qui seraient tournés vers l'amélioration des processus de récoltes et de traitements de la donnée et vers le développement d'autres programmes et outils d'accès aux informations, et pas seulement dans le domaine des couches géospatiales.

L'amélioration des processus et des résultats de l'entreprise devra passer par une intégration massive de tous les types d'informations qui sont traités au sein même de la compagnie. Il est nécessaire d'arriver à centraliser le plus de chose possible pour faciliter les échanges entre les équipes et permettre de rendre plus efficace l'interaction pluridisciplinaire sur les différentes tâches. Chez Ad Terra Energy, la plupart des informations traitées concernent ce qui se passe en sous-sol. QGIS et l'extension présentée dans ce document sont optimisés pour la représentation des informations en surface, notamment concernant les données de production par puits, les élévations ou la localisation des infrastructures industrielles. Cependant, les informations de sous-sols tels que les diagraphes de puits, les tops de forage ou les modèles de réservoirs sont beaucoup moins évidentes à visualiser. Un des principaux enjeux en perspective pour l'avenir se situe ici. Des programmes spécifiques sont utilisés pour le traitement de ces éléments par du personnel qualifié. Il sera nécessaire dans le futur de réussir à centraliser la variété des informations disponibles dans une optique de visualisation et d'accès aux données par tous. Un des défis qui attend la compagnie sera de trouver un outil ou une suite de programme permettant de rendre les interactions et les échanges entre ces différents formats beaucoup plus fluides et évidents. Cette centralisation de l'accès aux informations donnera une structure de référence efficace et dynamique qui devrait s'avérer performante pour l'accompagnement du nombre de projets grandissants de l'entreprise.

4.2) Réflexions sur le déroulement du stage

4.2.1. *Evaluation de stage*

Le stage réalisé chez Ad Terra Energy a été très enrichissant sur de nombreux points. Mon expérience au sein de la structure a duré six mois, ce qui m'a permis d'approfondir au maximum mon investissement dans les projets auxquels j'ai été assigné. Durant cette période, j'ai pu voir l'avancée des différents projets et l'évolution de l'entreprise depuis l'intérieur. L'organisation du stage m'a laissé le temps d'assimiler les sujets traités dans la structure et de me familiariser avec les concepts fondamentaux des ressources minérales et énergétiques, qui sont au cœur des services proposés par Ad Terra Energy. En parallèle de cette première étape, j'ai été formé à la réalisation de divers travaux, plus en moins en autonomie et liés à mes domaines de formations. La diversité des missions m'a aidé à assimiler d'autant plus rapidement les thématiques et objets d'études. Dans un second temps, après cette familiarisation, j'ai été introduit à ce qui allait devenir mon projet principal : le développement de l'extension pour QGIS. Ce nouveau projet s'est avéré être un défi important d'un point de vue personnel, il s'agissait du premier outil que je développais et je découvrais encore les langages de programmation notamment python. Ce projet a demandé beaucoup d'investissement et de nombreux essais ont été nécessaires avant de trouver la formule qui me convenait. Mon travail a tout de même été facilité par un encadrement efficace et la mise à disposition de modèle qui m'a donné un aperçu de ce qui était attendu. Lorsque l'on débute une expérience comme celle-ci, il faut être préparé à s'adapter, le milieu de l'entreprise peut contenir des imprévus ou des changements de priorités de

dernières minutes qui peuvent être possiblement déroutant au début, spécifiquement si l'on vient directement du milieu académique où les choses peuvent être plus encadrées. Enfin, ce stage m'a aussi fait prendre conscience qu'il est tout aussi important de savoir faire preuve d'autonomie que de pouvoir collaborer en équipe. En effet, le développement de projets d'une certaine envergure nécessite de la communication avec les collaborateurs pour ainsi se mettre à l'écoute de ce qui est attendu et de ce que nous pouvons avoir besoin des autres pour la réalisation des missions. À l'inverse, il est tout aussi important de pouvoir travailler de manière autonome et de ne pas se reposer constamment sur les autres. De façon plus générale, il est plus efficace de se montrer proactif pour réussir à l'implémenter au mieux dans un ouvrage.

4.2.2. Développement personnel

D'un point de vue personnel, mon expérience chez Ad Terra Energy m'a apporté beaucoup de positif sur différents points. Ce stage a été la première opportunité que j'ai eue de pouvoir appliquer ce que j'avais appris durant mon cursus de Géomatique à l'Université de Genève. J'ai donc pu mettre en pratique différentes méthodes que je n'avais abordées que lors de cours théoriques jusque-là. J'ai notamment pu exercer différents procédés de géotraitement, de gestion de données spatiales et de télédétections notamment. J'ai aussi pu acquérir une certaine autonomie et des responsabilités dans la réalisation de mes travaux et j'ai pu assimiler de quelle manière se construisent les nouvelles idées au sein d'une entreprise. Cependant, mes projets principaux de migration de base de données et de développement de l'extension QGIS m'ont apporté le plus de compétence dans des domaines qui étaient nouveaux pour moi. J'ai pu dans un premier temps m'assimiler aux concepts d'automatisation des tâches avec l'utilisation de scripts VBA et Python dans le traitement des données brutes pour les rendre compatibles aux formats de stockages formatés pour nos tables stockées sur PostgreSQL. J'ai aussi pu me familiariser davantage avec l'utilisation de script et de codes de programmation en approfondissant mes connaissances en R ou en SQL. Mais j'ai surtout développé des connaissances fortes sur l'utilisation du langage Python pour le développement d'outil. Le fait de programmer un instrument depuis sa base m'a vraiment aidé à assimiler et à me rendre à l'aise avec un langage que je n'avais jamais utilisé auparavant. Il va donc de soi que ces six mois passés chez Ad Terra Energy m'ont beaucoup apporté en termes de compétences techniques et organisationnelles.

4.2.3. Perspectives professionnelles

À la fin de mon stage chez Ad Terra Energy, une opportunité m'a été offerte afin d'approfondir mon expérience en gestion de bases de données et de perfectionner mes compétences en géomatiques sous la forme d'un contrat de travail. Au fil des mois, j'ai découvert le milieu de la programmation qui m'a fortement passionné. Dans le futur, je souhaite continuer à travailler dans le cadre des interactions entre les bases de données relationnelles et le développement d'outils toujours plus optimisés capable de déployer tout le potentiel des ressources à disposition. La conceptualisation et la mise en service d'un instrument comme l'extension QGIS apporte déjà une très grande satisfaction en soi, mais le plus intéressant dans ce domaine est le fait de se dire que ces objets ne sont jamais des produits finis. Ils peuvent toujours être modifiés et améliorés sur la base de nouveaux besoins d'analyses, de nouvelles idées de représentation des données ou simplement en termes d'efficacité. Dans un monde où les flux de données sont de plus en plus nombreux et importants, la mise en service d'instruments facilitant leurs intégrations et leurs utilisations s'avère d'autant plus nécessaire.

V) Bibliographie

Auzoux, Sandrine, and Thierry Chapuset. 'Conception de base de données sous Access à des fins d'analyse et de modélisation : Formation bases de données - Décembre 2012 -Thiès'. Monograph, 2012. <https://agritrop.cirad.fr/570830/>.

Conrad, Tim. 'PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need'. *DevX*, 2004, 5.

Duhaut, Patrick. 'Qu'est-ce qu'un plugin ?' <https://www.oni-cif.com/> (blog). Accessed 9 May 2023. <https://www.oni-cif.com/quest-ce-quun-plugin/>.

'Manuel d'utilisation de QGIS — Documentation QGIS Documentation'. Accessed 9 May 2023. https://docs.qgis.org/3.10/fr/docs/user_manual/index.html.

Petrosys. 'Petrosys – Mapping, Surface Modeling & Data Management Software for Oil and Gas Professionals', 2009. <https://wwwsandbox.petrosys.com.au/>.

'PostgreSQL 11.19 Documentation'. *The PostgreSQL Global Development Group*, 2023, 2835.

'PyQGIS 3.16 Developer Cookbook', n.d.

Plugin Builder—QGIS Python Plugins Repository. (s. d.). Consulté 12 mai 2023, à l'adresse <https://plugins.qgis.org/plugins/pluginbuilder/>

Sadoun, Balqies, Omar Al-Bayari, and Suhaib Al-Tawara. "'Open Source GIS Solution: An Overview of the Architecture of Free Open Source Web GIS".'. 2022.

Sahara. 'VISUALIZATION AND ANALYSIS TOOLS'. Accessed 9 May 2023. <https://interfaces.com.ar/en/resources/brochures.php>.

Sherman, Gary E, Tim Sutton, Radim Blazek, and Lars Luthman. 'Quantum GIS User Guide', November 2005, 70.

Soloz, Nathalie. 'Intégration des données spatiales dans les applications de gestion'. Haute école valaisanne, 2006. Swiss Open Access Repository.

University of Texas at Arlington. 'Paleomap Project'. Scotese, 2016. <http://www.scotese.com/>.

Willman, J. M. (2022). Creating GUIs with Qt Designer. In J. M. Willman (Éd.), *Beginning PyQt : A Hands-on Approach to GUI Programming with PyQt6* (p. 217-258). Apress. https://doi.org/10.1007/978-1-4842-7999-1_8

VI) Annexes

```
1 # -*- coding: utf-8 -*-
2
3 #
4 #
5 #
6 #
7 #
8 #
9 #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 #
18 #
19 #
20 #
21 #
22 #
23 #
24 #
25 #
26 #
27 #
28 #
29 #
30 #
31 #
32 #
33 #
34 #
35 #
36 #
37 #
38 #
39 #
40 #
41 #
42 #
43 #
44 #
45 #
46 #
47 #
48 #
49 #
50 #
51 #
52 #
53 #
54 #
55 #
56 #
57 #
58 #
59 #
60 #
61 #
62 #
63 #
64 #
65 #
66 #
67 #
68 #
69 #
70 #
71 #
72 #
73 #
74 #
75 #
76 #
77 #
78 #
79 #
80 #
81 #
82 #
83 #
84 #
85 #
86 #
87 #
88 #
89 #
90 #
91 #
92 #
93 #
94 #
95 #
96 #
97 #
98 #
99 #
100 #
101 #
102 #
103 #
104 #
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #
```

```
617         or selected_table == "watsapp_well_tests"
618         selected_table = "w11_tests"
619         field_panel = [not available]
620
621     elif "lemetry" in selected_table:
622         field_panel = [use telemetry viewer]
623
624     else:
625         conn = psycopg2.connect(host="111.222.12.333", port="1234", dbname="project_data", user=self.dockwidget.lineEdit().text(),
626                               with conn.cursor() as cur:
627             cur.execute("SELECT * FROM " + str(selected_table) + " ;")
628             field_panel = [0] for i in cur.description]
629         self.dockwidget.comboBox.addItem(field_panel)
630
631 #mapping process
632 def mapping():
633     global date
634     global target_field
635     global selected_table
636     global range_list
637     global symbolize
638     global mySteps
639     global lyr
640
641 #parametering of mapping process for the selected data and widget tools - query connection
642 steps = int(self.dockwidget.mySteps.currentText())
643 mySteps = 1
644 lyr = 0
645 symbolize = int(self.dockwidget.symbolize.currentText() + 1) * 5
646 selected_table = self.dockwidget.comboBox.currentText()
647 print(selected_table)
648 target_field = self.dockwidget.comboBox.currentText()
649 date = "" + self.dockwidget.dateTimeEdit.date().text() + ""
650
651 #data of interest recognition query with data selection
652 conn = psycopg2.connect(host="111.222.12.333", port="1234", dbname="project_data", user=self.dockwidget.lineEdit().text(),
653                       with conn.cursor() as cur:
654             cur.execute("SELECT a.easting, a.northing, b.id_1, b.id_2, b.id_3 + target_field + ' FROM well_collection
655                       ON a.id_1 = b.id_1
656                       WHERE b.date = '" + date + "'
657                       EXCEPT SELECT a.easting, a.northing, b.id_1, b.id_2, b.id_3 + target_field + ' FROM well_collection
658                       ON a.id_1 = b.id_1
659                       WHERE easting = '999999'
660                       ")
661             global qry
662             qry = cur.fetchall()
663             view = [0] for i in cur.description]
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
641 QStandardItem *vlayer_item;
642 QStandardItem *vlayer_status;
643 QStandardItem *vlayer_name;
644 QStandardItem *vlayer_double;
645 QStandardItem *vlayer_double2;
646
647 // vlayer update fields
648 for row in qrys:
649     point = QPoint(row[5], row[6])
650     feat = QFeature()
651     feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(point)))
652     feat.setAttribute(QDate(row[9], row[1], row[2], row[3], row[4]))
653     prov.addFeatures([feat])
654
655 // vlayer update extents
656
657 #vlayer and labeling for renderer
658
659 mySymbol = QgsSymbol.defaultSymbol(vlayer_geometryType())
660 mySymbol.setColor(QGuiColor('#808080'))
661 mySymbol.setSize(4)
662
663 mySymbol2 = QgsSymbol.defaultSymbol(vlayer_geometryType())
664 mySymbol2.setColor(QGuiColor('#E0E0E0'))
665 mySymbol2.setSize(4)
666
667 cat1 = QgsRenderCategory('Open', QgsMarkerSymbol(mySymbol), 'Open')
668 cat2 = QgsRenderCategory('Closed', QgsMarkerSymbol(mySymbol2), 'Closed')
669
670 name_fields = 'concat("id_1", "v", "id_2") || "v" || "well_type"'
671
672 test_format = QgsTextFormat()
673 label = QgsPalayerSettings()
674 label.fieldName = name_fields
675 label.isExpression = True
676 label.placement = QgsPalayerSettings.Line
677 label.enabled = True
678 label.setTextFormat(test_format)
679
680
681 labeler = QgsVectorLayerSimpleLabeling(label)
682 lyr.setLabelEnabled(True)
683 lyr.setLabeling(labeler)
684 lyr.triggerRepaint()
685
686 categorized_renderer = QgsCategorizedSymbolRenderer()
687 categorized_renderer.setClassificationField('well_status')
688 categorized_renderer.addCategory(cat1)
689 categorized_renderer.addCategory(cat2)
690 lyr.setRenderer(categorized_renderer)
691
692 QgsProject.instance().addMapLayer(lyr)
693
694
695 def hide():
696     #hide the last situation layer
697     for id, lyr in QgsProject.instance().mapLayers().items():
698         if 'status' in lyr.name():
699             QgsProject.instance().removeMapLayer(lyr.id())
700
701
702 @uiSlot('id_loc')
703 def id_loc_clicked():
704     id_loc = self.dockwidget_id_select.currentText()
705     well_name_loc = ' ' + self.dockwidget_well_name.text() + ' '
706
707 #query of interest for data recollection - query graphical connection
708 with conn as cursor:
709     cur.execute("SELECT * FROM well_collection WHERE " + id_loc + " = '" + well_name_loc + "' ;")
710     qrys = cur.fetchall()
711
712 vlayer = QgsVectorLayer('Point?crs=epsg:32638', 'label' + str(well_name_loc), 'memory')
713 prov = vlayer.dataProvider()
714
715 prov.addAttributes([ QgsField("id_1", QVariant.String),
716                     QgsField("id_2", QVariant.String),
717                     QgsField("vname", QVariant.Double),
718                     QgsField("vdouble", QVariant.Double)])
719
720 vlayer.updateFields()
721 for row in qrys:
722     point = QPoint(row[4], row[5])
723     feat = QFeature()
724     feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(point)))
725     feat.setAttribute(row[1], row[2])
726     prov.addFeatures([feat])
727
728 vlayer.updateExtents()
729
730 mySymbol = QgsSymbol.defaultSymbol(vlayer_geometryType())
731 mySymbol.setSize(1)
732 mySymbol.setColor(QGuiColor('black'))
733
734 test_format = QgsTextFormat()
735 label = QgsPalayerSettings()
736 label.fieldName = str(id_loc)
737 label.enabled = True
738 label.setTextFormat(test_format)
739
740 label.placement = QgsPalayerSettings.Line
741
742 labeler = QgsVectorLayerSimpleLabeling(label)
743 vlayer.setLabelEnabled(True)
744 vlayer.setLabeling(labeler)
745 vlayer.triggerRepaint()
746
747 myRenderer = QgsSingleSymbolRenderer(QgsMarkerSymbol(mySymbol))
748 vlayer.setRenderer(myRenderer)
749 QgsProject.instance().addMapLayer(vlayer)
750
751 def cancel():
752     #hide the table of wells
753     for id, lyr in QgsProject.instance().mapLayers().items():
754         if 'label' in lyr.name():
755             QgsProject.instance().removeMapLayer(lyr.id())
756
757
758 prov.addAttributes([QgsFieldNames, QVariant.String,
759                   QgsFieldNames, QVariant.Double])
760
761 vlayer.updateFields()
762
763 for row in qrys:
764     point = QPoint(row[0], row[1])
765     feat = QFeature()
766     feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(point)))
767     if table_test == "well_tests":
768         feat.setAttribute(row[8], row[1], row[2], row[3], row[4], QDate(row[5]), row[6], row[7], row[8],
769                          row[9], row[10], row[11], row[12], row[13], row[14], round(row[15], 0), row[16])
770     elif table_test == "int_well_tests":
771         feat.setAttribute(row[8], row[1], row[2], row[3], row[4], QDate(row[5]), row[6], round(row[7]*100, 2), row[8])
772     else:
773         feat.setAttribute(row[8], row[1], row[2], row[3], row[4], QDate(row[5]), row[6], round(row[7]*100, 2), row[8])
774
775     prov.addFeatures([feat])
776     vlayer.updateExtents()
777
778 #feat settings parametrizing
779 myRangeList = []
780 myOpacity = 1
781
782 vlayer_field = vlayer.fields()
783 if table_test == "well_tests":
784     field_id = vlayer_field.indexFromName("pnr")
785     elif table_test == "int_well_tests":
786         field_id = vlayer_field.indexFromName("pnr_corrected")
787     else:
788         field_id = vlayer_field.indexFromName("water")
789
790 for i in range(1, 1000):
791     row_feats = [feat[field_id] for feat in vlayer.getFeatures()]
792     i = 0
793     while row_feats[i] == NULL:
794         i = i + 1
795     feats = []
796     for f in row_feats:
797         if f == NULL:
798             f = 0
799         feats.append(f)
800
801 [round(item, 0) for item in row_feats]
802 except:
803     dialog_exec()
804
805 #vlayer processing referring of the test table
806 if table_test == "well_tests":
807     myLabel = (000) @ [Scf/001]
808 else:
809
810
811 self.dockwidget_show_status.clicked.connect(current)
812 self.dockwidget_show_status.clicked.connect(hide)
813 self.dockwidget_hide_status.clicked.connect(hide)
814
815 self.dockwidget_fill_data.clicked.connect(field_fill)
816
817 #display of well logs
818
819 def loading_all():
820     id_loc = self.dockwidget_id_select.currentText()
821     #layer creation for well_logs display
822     conn = psycopg2.connect("host='111.222.12.333', port='1234', dbname='project_data', user=self.dockwidget_lineedit_text)
823     with conn.cursor() as cur:
824         cur.execute("SELECT * FROM well_collection")
825         qrys = cur.fetchall()
826
827     vlayer = QgsVectorLayer('Point?crs=epsg:32638', 'label' + str(well_name_loc), 'memory')
828     prov = vlayer.dataProvider()
829
830     prov.addAttributes([ QgsField("id_1", QVariant.String),
831                         QgsField("id_2", QVariant.String),
832                         QgsField("vname", QVariant.Double),
833                         QgsField("vdouble", QVariant.Double)])
834
835     vlayer.updateFields()
836     for row in qrys:
837         point = QPoint(row[4], row[5])
838         feat = QFeature()
839         feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(point)))
840         feat.setAttribute(row[1], row[2])
841         prov.addFeatures([feat])
842
843     vlayer.updateExtents()
844
845     mySymbol = QgsSymbol.defaultSymbol(vlayer_geometryType())
846     mySymbol.setSize(1)
847
848     test_format = QgsTextFormat()
849     label = QgsPalayerSettings()
850     label.fieldName = str(id_loc)
851     label.enabled = True
852     label.setTextFormat(test_format)
853
854     labeler = QgsVectorLayerSimpleLabeling(label)
855     vlayer.setLabelEnabled(True)
856     vlayer.setLabeling(labeler)
857     vlayer.triggerRepaint()
858
859     myRenderer = QgsSingleSymbolRenderer(QgsMarkerSymbol(mySymbol))
860     vlayer.setRenderer(myRenderer)
861     QgsProject.instance().addMapLayer(vlayer)
862
863 def labeling_one():
864     QgsProject.instance().removeMapLayer(vlayer_id())
865
866 self.dockwidget_show_id_data.clicked.connect(labeling_all)
867 self.dockwidget_show_one_clicked.connect(labeling_one)
868 self.dockwidget_hide_id_data.clicked.connect(cancel)
869
870 #Change stacked dockwidget
871 def id_loc_2():
872     self.dockwidget_location_wells.setCurrentWidget(self.dockwidget_position_well)
873     self.dockwidget_location_clicked.connect(id_loc)
874
875 def back_data():
876     self.dockwidget_location_wells.setCurrentWidget(self.dockwidget_data_select)
877     self.dockwidget_location_2.clicked.connect(back_data)
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

1110 == int('well_tests')
1111 myRange1 = QgsRendererRange(2001, 3000, mySymbol1, myLabel1)
1112
1113 #elif
1114 myRange2 = QgsRendererRange(3001, 4000, mySymbol2, myLabel2)
1115 myRangeList.append(myRange2)
1116
1117 #elif
1118 if table_test == "int_well_tests":
1119     table_test = "int_well_tests"
1120     mySymbol1 = QgsSymbol.defaultSymbol(QgsVectorLayerGeometryType())
1121     mySymbol2 = QgsSymbol.defaultSymbol(QgsVectorLayerGeometryType())
1122     mySymbol3.setSize(4)
1123     mySymbol4.setSize(8)
1124     myRange3 = QgsRendererRange(2001, 3000, mySymbol3, myLabel3)
1125     myRange4 = QgsRendererRange(3001, 4000, mySymbol4, myLabel4)
1126     myRangeList.append(myRange4)
1127
1128 #elif
1129 if table_test == "well_tests":
1130     table_test = "well_tests"
1131     name_fields = "concat(gpr|| 'w' ||'id_2')'"
1132     elif table_test == "int_well_tests":
1133         name_fields = "concat(gpr_corrected)|| 'w' ||'id_2')'"
1134     else:
1135         name_fields = "concat(water)|| 'w' ||'id_2')'"
1136
1137 test_format = QgsTextFormat()
1138 label = QgsTextRendererSettings()
1139 label.fieldName = name_fields
1140 label.isExpression = True
1141 label.placement = QgsTextRendererSettings.Line
1142 label.enabled = True
1143 label.setFormat(test_format)
1144
1145 #elif
1146 labeler = QgsVectorLayerSimpleLabelingLabeler()
1147 vlayer.setMetadataLabeler(labeler)
1148 vlayer.triggerRepaint()
1149
1150 #Renderer production for mapping process
1151 myRenderer = QgsRendererManager.createRenderer("", myRangeList)
1152 if table_test == "well_tests":
1153     myRenderer.setClassAttribute("gpr")
1154     myRenderer.setClassAttribute("gpr_corrected")
1155     myRenderer.setClassAttribute("water")
1156 else:
1157     myRenderer.setClassAttribute("water")
1158
1159 vlayer.setRenderer(myRenderer)
1160 QgsProject.instance().addMapLayer(vlayer)
1161 except:
1162     dialog.exec_()
1163
1164 #Categorized production process
1165
1166 def plot():
1167     id = self.dockWidget.test_id_type.currentText()
1168     date_time = self.dockWidget.date_time_currentText()
1169     date_time = self.dockWidget.date_time_currentText()
1170     date_time2 = self.dockWidget.date_time2_currentText()
1171
1172 conn = psycopg.connect(host='131.222.12.333', port='1234', dbname='project_data', user=self.dockWidget.lineEdit().text())
1173 with conn.cursor() as cur:
1174     cur.execute("SELECT a.setting, b.northing, b"
1175 FROM wells_collection a
1176 LEFT JOIN " + table_test + " b
1177 ON a.id_2 = b.id_2
1178 WHERE date = " + date_time + "
1179 SELECT ROWID(a)
1180 FROM " + table_test + " WHERE id_2 = b.id_2 AND date BETWEEN '" + date_time + "' AND '" + date_time2 + "'")
1181
1182 qry = cur.fetchall()
1183 view = [0] for i in cur.description)
1184
1185 #GIS layer production and QGIS field parametrizing
1186 vlayer = QgsVectorLayer("Point[crs=epsg:3000]", "id", self.dockWidget.date_time_currentText() + " " + self.dockWidget.date_time2_currentText())
1187 prov = vlayer.dataProvider()
1188
1189 for names in view:
1190     prov.setAttribute(QgsFieldNames.QVariant.String, QgsFieldNames.QVariant.Double))
1191
1192 vlayer.updateFields()
1193
1194 for row in qry:
1195     point = QgsPoint(row[0], row[1])
1196     feat = QgsFeature()
1197     feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(point)))
1198     if table_test == "well_tests":
1199         feat.setAttributes([row[1], row[2], row[3], row[4], QgsPoint(row[5], row[6], row[7]), row[8], row[9], row[10], row[11], row[12], row[13], row[14], row[15], row[16]])
1200     elif table_test == "int_well_tests":
1201         feat.setAttributes([row[1], row[2], row[3], row[4], QgsPoint(row[5], row[6], row[7]), row[8], row[9], row[10], row[11], row[12], row[13], row[14], row[15], row[16], row[17], row[18], row[19]])
1202     else:
1203         feat.setAttributes([row[1], row[2], row[3], row[4], QgsPoint(row[5], row[6], row[7]), row[8], row[9], row[10], row[11], row[12], row[13], row[14], row[15], row[16], row[17], row[18], row[19], row[20], row[21]])
1204
1205 except:
1206     dialog.exec_()
1207
1208 elif table_test == "field_id_well_tests":
1209     feat.setAttributes([row[1], row[2], row[3], row[4], QgsPoint(row[5], row[6], row[7]), row[8], row[9], row[10], row[11], row[12], row[13], row[14], row[15]])
1210     else:
1211         feat.setAttributes([row[1], row[2], row[3], row[4], QgsPoint(row[5], row[6], row[7]), row[8], row[9], row[10], row[11], row[12], row[13], row[14], row[15], row[16], row[17], row[18], row[19]])
1212
1213 prov.setAttributeNames(feat)
1214 vlayer.updateExtent()
1215
1216 #Test settings parametrizing
1217 myRangeList = []
1218 myOpacity = 1
1219
1220 layer_fields = vlayer.fields()
1221 if table_test == "well_tests":
1222     field_id = layer_fields.indexFromName("gpr")
1223     elif table_test == "int_well_tests":
1224         field_id = layer_fields.indexFromName("gpr_corrected")
1225     else:
1226         field_id = layer_fields.indexFromName("water")
1227     try:
1228         row_feats = [feat[field_id] for feat in vlayer.getFeatures()]
1229     except:
1230         i = 0
1231
1232 #elif
1233 myRange2 = QgsRendererRange(10, 30, mySymbol2, myLabel2)
1234 myRangeList.append(myRange2)
1235
1236 #elif
1237 if table_test == "well_tests":
1238     table_test = "int_well_tests"
1239     myLabel3 = "2001 - 3000 [Scf/m3]"
1240     myColor3 = QgsColor(QgsColor.fromString("#FF0000"))
1241     else:
1242         myLabel3 = "30 x 30"
1243         myColor3 = QgsColor(QgsColor.fromString("#808080"))
1244     mySymbol3 = QgsSymbol.defaultSymbol(QgsVectorLayerGeometryType())
1245     mySymbol3.setSize(4)
1246     mySymbol3.setOpacity(myOpacity)
1247     mySymbol3.setSize(4)
1248     if table_test == "well_tests":
1249         myRange3 = QgsRendererRange(2001, 3000, mySymbol3, myLabel3)
1250     else:
1251         myRange3 = QgsRendererRange(30, 61, 100, mySymbol3, myLabel3)
1252     myRangeList.append(myRange3)
1253
1254 #elif
1255 if table_test == "well_tests":
1256     table_test = "int_well_tests"
1257     myLabel4 = "3000 [Scf/m3]"
1258     myColor4 = QgsColor(QgsColor.fromString("#FF0000"))
1259     mySymbol4 = QgsSymbol.defaultSymbol(QgsVectorLayerGeometryType())
1260     mySymbol4.setSize(8)
1261     mySymbol4.setOpacity(myOpacity)
1262     mySymbol4.setSize(8)
1263     mySymbol4.setSize(8)
1264     myRange4 = QgsRendererRange(3000, 9999999, mySymbol4, myLabel4)
1265     myRangeList.append(myRange4)
1266
1267 #elif
1268 if table_test == "well_tests":
1269     name_fields = "concat(gpr)|| 'w' ||'id_2')'"
1270     elif table_test == "int_well_tests":
1271         name_fields = "concat(gpr_corrected)|| 'w' ||'id_2')'"
1272     else:
1273         name_fields = "concat(water)|| 'w' ||'id_2')'"
1274
1275 test_format = QgsTextFormat()
1276 label = QgsTextRendererSettings()
1277 label.fieldName = name_fields
1278 label.isExpression = True
1279 label.placement = QgsTextRendererSettings.Line
1280 label.enabled = True
1281 label.setFormat(test_format)
1282
1283 #elif
1284 labeler = QgsVectorLayerSimpleLabelingLabeler()
1285 vlayer.setMetadataLabeler(labeler)
1286 vlayer.triggerRepaint()
1287
1288 #Renderer production for mapping process
1289 myRenderer = QgsRendererManager.createRenderer("", myRangeList)
1290 if table_test == "well_tests":
1291     myRenderer.setClassAttribute("gpr")
1292     elif table_test == "int_well_tests":
1293         myRenderer.setClassAttribute("gpr_corrected")
1294     else:
1295         myRenderer.setClassAttribute("water")
1296
1297

```



```

2223 mySymbol = QgsSymbol.defaultSymbol(vlayer.geometryType())
2224 mySymbol.setColor(QGuiColor('#0000FF'))
2225 mySymbol.setSize(4)
2226 myRange3 = QgsRenderRange(prod_mid3, prod_max3, mySymbol, myLabel3)
2227 range_list.append(myRange3)
2228
2229 myLabel4 = str(round(prod_mid4, 2)) + " " + str(round(prod_max4, 2))
2230 mySymbol4 = QgsSymbol.defaultSymbol(vlayer.geometryType())
2231 mySymbol4.setColor(QGuiColor('#0000FF'))
2232 mySymbol4.setSize(5)
2233 myRange4 = QgsRenderRange(prod_mid4, prod_max4, mySymbol4, myLabel4)
2234 range_list.append(myRange4)
2235
2236 name_field = "concat('' + completion_field + '' | 'w' | '14_23')'"
2237 text_format = QgsTextFormat()
2238 label = QgsPalLayerSettings()
2239 label.fieldName = name_field
2240 label.isPrecision = True
2241 label.placement = QgsPalLayerSettings.Line
2242 label.enabled = True
2243 label.setTextFormat(text_format)
2244
2245 labeler = QgsVectorLayerSimpleLabeling(label)
2246 vlayer.setMinimalRenderFrom()
2247 vlayer.setLabeling(labeler)
2248 vlayer.triggerRepaint()
2249
2250 #render production for mapping process
2251 myRenderer = QgsDefaultSymbolRenderer('' + range_list)
2252 myRenderer.setClassAttribute(completion_field)
2253 vlayer.setRenderer(myRenderer)
2254 QgsProject.instance().addMapLayer(vlayer)
2255
2256 QgsMapCanvas().refresh()
2257
2258 except:
2259     dialog_exec()
2260
2261 def remove_lastIntervention():
2262     for id, vlayer in QgsProject.instance().mapLayers().items():
2263         if "last_" in vlayer.name():
2264             QgsProject.instance().removeMapLayer(vlayer.id())
2265
2266 self.dockwidget.tableView_apply_3.clicked.connect(remove_lastIntervention)
2267 self.dockwidget.tableView_cancel_3.clicked.connect(remove_lastIntervention)
2268 self.dockwidget.tableView_cancel_2.clicked.connect(remove_lastIntervention)
2269 self.dockwidget.tableView_cancel_1.clicked.connect(remove_lastIntervention)
2270
2271 #####
2272
2273
2274
2275
2276
2277
2278
2279 #####
2280
2281 #save map
2282 def save():
2283     layers = self.iface.mapCanvas().layers()
2284     layer = layers[0]
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386

```