



PROJET TUTORÉ

Data Lake, Data Warehouse : principes, comparaisons, utilisations en Santé

Recherches et rédactions réalisées par, Denoëla Guennoc, Clémence Delestre, Camille Henry et Gérald Escolano, dans le cadre du projet tutoré de recherche bibliographique du 1er semestre 2020

SOMMAIRE

INTRODUCTION	3
I/ Data Warehouse	4
Data Warehouse : qu'est-ce que c'est ?	4
Architecture d'un Data Warehouse	5
Mise en place d'un Data Warehouse	10
II/ Data Lake	14
Définition	14
Architecture d'un Data Lake	16
Mise en place	17
III/ Comparaison et utilisations en santé	20
Utilisation, avantages, inconvénients d'un Data Warehouse	20
Utilisation, avantages, inconvénients d'un Data Lake	21
Quelle solution pour quelle utilisation (en santé) ?	22
Qu'utilise le secteur de la santé de nos jours ?	24
IV/ Secouriste de poche	26
Objectifs du projet	26
Modèle conceptuel de données	27
Maquette du site	29
CONCLUSION	31
BIBLIOGRAPHIE	32
ANNEXES	35
Annexe 1 : Amazon Redshift Manuel du développeur de base de données; Architecture système de l'entrepôt de données	35
Annexe 2 : Dremel: Interactive Analysis of Web-Scale Datasets	38
Annexe 3 : Data Warehouse Tools: Faster Time-to-Value for Your Healthcare Data Warehouse	48

INTRODUCTION

Depuis la nuit des temps, l'Humain a besoin de connaissances pour se développer, de les stocker, de les analyser etc.. Que ce soit sur les murs de grottes, sur des tablettes en pierre ou encore sur des livres de papyrus, la connaissance se transmet de génération en génération depuis des temps immémoriaux via de nombreux supports. Cependant depuis la fin des années 90 et l'apparition d'internet, le besoin en matière de stockage se fait de plus en plus grand et ces quantités pharaoniques d'informations doivent ensuite être analysées, étudiées, décortiquées pour que L'Humain puisse continuer à s'adapter à l'environnement avec succès. C'est pourquoi il fallait trouver un moyen simple et rapide pour pouvoir prendre en main toutes ces informations dont nous disposons. Le "Big Data" est né.

De nos jours on agence donc nos données selon plusieurs modèles comme par exemple en Data Warehouses qui sont, " un ensemble de données orientées sujet, intégrées, variables dans le temps et non volatiles." (Oracle.com. Data Warehouse : Qu'est-ce que c'est ?). C'est selon William H. Inmon, l'inventeur du terme, les quatres caractéristiques propres au Data Warehouse. On considère donc que ce type de schéma basé sur un schéma statique peut accueillir des pétaoctets de données, notamment.

Cependant les pétaoctets ne sont peut être pas suffisant pour certains besoin, ou alors peut être que certains enjeux exigent une lisibilité des données en temps réel ou encore d'autres paramètres empêchent l'utilisation d'un Data Warehouse. C'est pour cela que nous avons besoin de transformer une nouvelle fois notre façon d'utiliser nos données. Par conséquent depuis 2010, certaines personnes parlent de Data Lakes permettant une plus grande flexibilité et un plus grand stockage que les Data Warehouse, en effet ceux-ci peuvent atteindre l'exaoctet en termes de stockage.

Nous verrons par la suite les avantages sur ces deux structures, que ce soit en termes d'efficacité pour certaines tâches, économique ou encore en termes de complexité d'utilisation. Après cela nous verrons des exemples portant sur ces deux structures en santé. Comme nous savons que la santé est sujette à l'utilisation de grandes quantités de données nous pouvons d'ores et déjà supposer que leur utilisation est importante dans ce secteur.

Dans une dernière partie nous allons décrire le projet sur lequel nous allons travailler le semestre prochain. Nous définirons le but du projet que nous avons nommé "Secourisme de poche" puis avec une visée un peu plus technique nous aborderons le modèle conceptuel de donnée de notre projet. Enfin on vous présentera la maquette que nous avons faite et qui sera la base de notre futur projet, même s'il est possible que des changements soient opérés.

I/ Data Warehouse

A. Data Warehouse : qu'est-ce que c'est ?

Les Data Warehouse, ou entrepôts de données en français, sont apparus progressivement dans les années 1960 et 1970. Avec l'apparition du numérique, il était nécessaire que de nouveaux systèmes de stockage de données se mettent en place. Les entreprises de l'époque possédaient chacune des systèmes de données opérationnels et transactionnels. Cependant, ceci n'étaient conçus que pour héberger une quantité de données limitées et d'actualité. Ils n'étaient donc pas en capacité de stocker des données dites historiques. A chaque fin de cycle (mois, trimestre, année ou autre), ces entreprises sont forcées de supprimer toutes leurs données opérationnelles et/ou transactionnelles pour faire de la place aux nouvelles sur leurs disques. Elles n'ont alors plus accès aux données précédentes et il leur est impossible de faire un certain nombre de requêtes pourtant indispensables à leurs bases de données. Il est par exemple impossible pour une entreprise d'analyser l'évolution de ses ventes sur les dernières années. Il est par conséquent également impossible de se servir de ces chiffres pour faire des prédictions sur les ventes à venir ou encore de voir quels sont les produits qui ont le mieux fonctionné. Ce sont autant de problèmes qui ont poussé les ingénieurs à penser de nouveaux systèmes de stockage. Il leur fallait un système de données dans lequel ils pourraient copier l'intégralité des informations présentent sur leurs bases de données avant de vider ces dernières. Ce nouveau système devrait conserver toutes les données copiées en les ordonnant dans le temps. (Diyotta.com. Data Warehouse: what it is, history and why it matters.)

C'est donc ainsi que les Data Warehouse ont fait leur apparition. Le nom d'entrepôt qui leur a été attribué donne une image très parlante de leur mode de fonctionnement. Comme dans un entrepôt, les données sont triées, harmonisées et rangées sur des étagères où tout est accessible à tout moment. Le système sait où trouver chaque donnée, qu'elle date de la première ou de la dernière sauvegarde. De cette façon, les informations sont accessibles et par conséquent utilisables. Ces Data Warehouse se sont énormément développés par la suite à partir de la seconde moitié des années 1980 lorsque les entreprises ont commencé à utiliser des systèmes d'aide à la décision. Aujourd'hui, posséder un système de données stable et efficace, qu'il s'agisse d'un Data Warehouse ou d'une autre solution, est une priorité pour la majorité des entreprises quel que soit leur domaines d'action.

Comme défini par William H. Inmon, un Data Warehouse est donc "un ensemble de données orientées sujet, intégrées, variables dans le temps et non volatiles." (Oracle.com. Data Warehouse : Qu'est-ce que c'est ?)

- **orientées sujet** : Afin que chaque donnée puisse être accessible facilement, elles sont rangées par thème, par sujet. A la manière d'un entrepôt, les marchandises, ou ici informations, de même nature sont stockées au même endroit.
- **intégrées** : Les données présentes dans un Data Warehouse sont de provenances multiples. Celles-ci peuvent avoir pour origine un nombre de bases de données infini et donc de constructions diverses. Il est par conséquent nécessaire que chaque donnée soit intégrée individuellement

pour que chaque donnée de l'entrepôt soit sous la même forme. Il s'agit de créer puis de maintenir une cohérence entre les informations. Il est également courant de stocker différentes versions d'une même donnée dans ces entrepôts ce qui ajoute à l'importance de l'intégration des données.

- **variables dans le temps** : Comme vous l'aurez compris, le Data Warehouse sert en quelque sortes d'archives à l'entreprise. Il contient des données sur une période de temps étendue et auxquelles il faut pouvoir accéder en fonction de leur date de création. Qu'il faut également pouvoir mettre en regard les unes des autres selon ces mêmes dates, etc. Il est donc nécessaire qu'un cadre temporel soit établi entre les informations présentes dans l'entrepôt.
- **non volatiles** : Dans le but de maintenir l'intégrité de l'entrepôt, une fois qu'une donnée y est ajoutée, elle n'est plus altérable. Celle-ci ne pourra en effet plus être modifiée ou encore supprimée.

Depuis quelques années les Data Warehouse connaissent une nouvelle évolution dans leur histoire puisque de nombreuses entreprises se tournent vers une solution dématérialisée. Les entrepôts de données se déclinent dorénavant également dans le cloud. Ils offrent alors trois principaux avantages à ceux qui préféreront cette solution. Tout d'abord, les coûts de ces systèmes sont très différents. Ici, il n'est plus nécessaire d'acheter le matériel hardware pour accueillir ses données. De plus, la mise en place d'un tel entrepôt et sa gestion sont plus simples et plus rapides. Enfin, les requêtes complexes faites sur cette base de données sont traitées beaucoup plus vite grâce au MPP (Massively Parallel Processing)(365 Data Science. What Is a Data Warehouse?). Autant de raisons qui tendent vers une plus grande utilisation de la solution cloud du Data Warehouse. Quoi qu'il en soit, ce modèle de données semble encore avoir de beaux jours devant lui.

B. Architecture d'un Data Warehouse

L'Architecture d'un système est un modèle conceptuel qui décrit la structure et le comportement d'un système.

Comme de nombreux autres systèmes, le DataWarehouse a subi quelques améliorations dans son architecture. En effet, on peut distinguer une architecture dite traditionnelle et une architecture basée sur le Cloud.

1) Architecture traditionnelle

L'architecture traditionnelle d'un Data Warehouse est appelée "architecture en trois tiers" c'est-à-dire qu'elle est composée de 3 niveaux comme on peut le voir sur la figure 1.

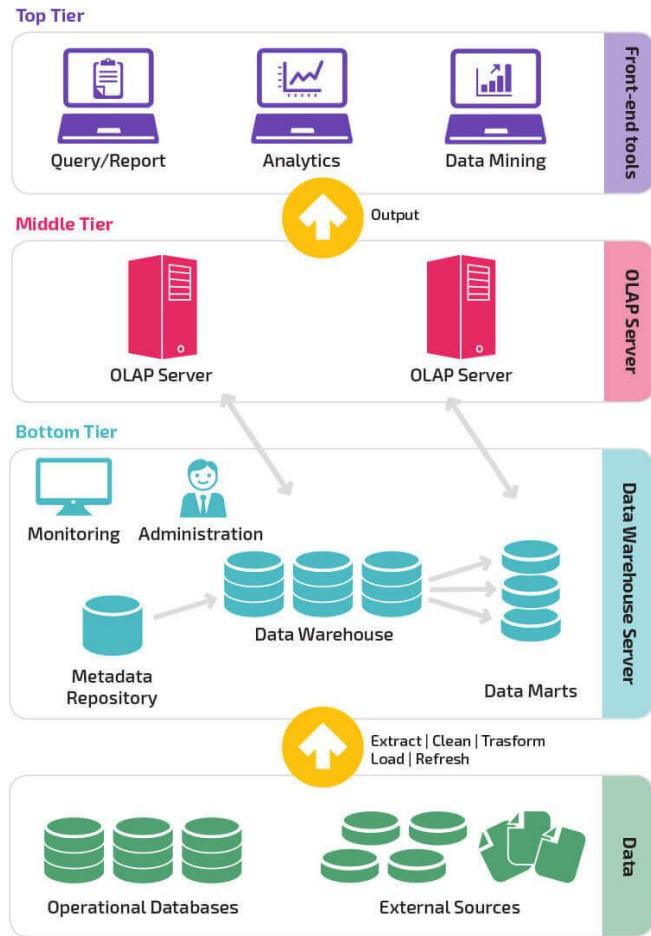


Figure 1 : architecture en trois tiers

Le niveau supérieur, qu'on peut appeler couche client, est l'endroit où le client accède et interagit avec les données. C'est dans ce niveau qu'on peut utiliser des outils de reporting, de requête, d'analyse et d'exploration de données.

Le niveau intermédiaire arrange et transforme les données via un serveur OLAP (Pour plus d'informations se référer à : Lebigdata.com. OLAP : définition d'une technologie d'analyse multidimensionnelle) afin que leur analyse soit plus facile.

Le niveau inférieur est la base de données du Data Warehouse. En effet, il contient un serveur base de données permettant d'extraire les données de sources différentes. C'est ici que les données qui ont été nettoyées et transformées au niveau intermédiaire sont chargées.

2) Chargement de données d'un DataWarehouse : méthode ETL ou ELT

Comme son nom l'indique, l'ETL (Extract Transform Load) extrait d'abord les données brutes qui sont stockées dans une base, le plus souvent transactionnelle afin de les mettre dans une base de données temporaire. C'est dans cette base temporaire que vont avoir lieu les opérations de transformation qui comprennent des actions de nettoyage, standardisation, enrichissement et qualification. Tout cela permet de rendre les données

brutes exploitables. Vient ensuite la phase de chargement : les données transformées vont être chargées vers le système centralisé.

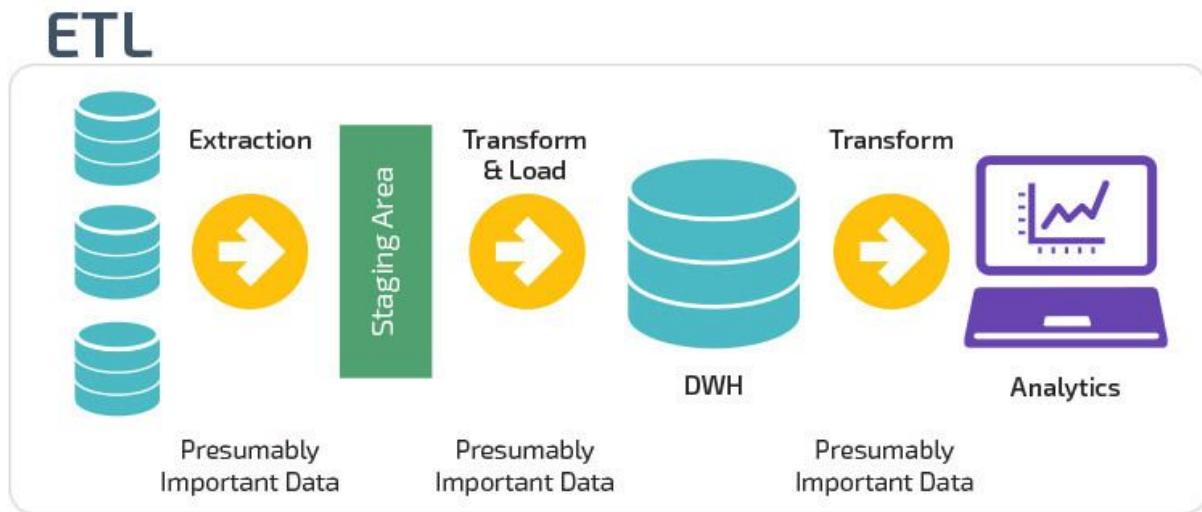


Figure 2 : ETL

Comme la méthode ETL, l'ELT (Extract Load Transform) extrait d'abord des données brutes. Mais contrairement au ETL, ici les données ne sont pas entreposées dans une base temporaire. Elles sont directement chargées dans le référentiel unique et centralisé. C'est là qu'a lieu la transformation des données. Néanmoins, les données brutes sont quand même gardées.

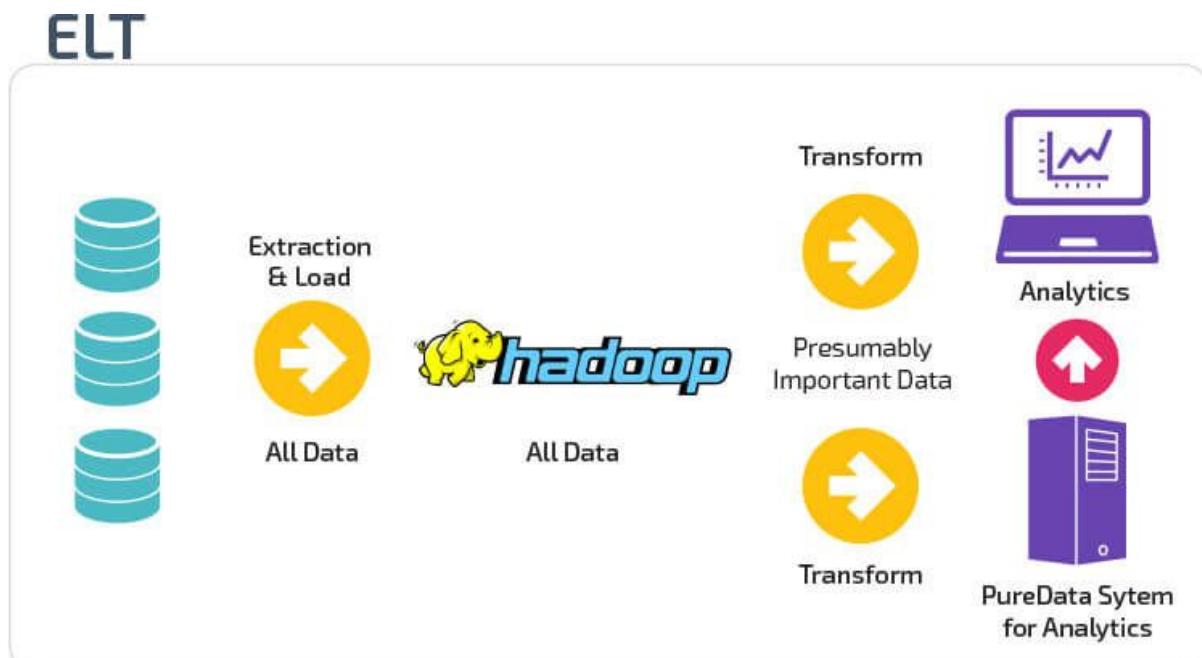


Figure 3: ELT

Ces deux méthodes atteignent le même but. Elles sont toutes les deux utilisées pour la production de données décisionnelles à partir de données brutes. Néanmoins, leur usage

diffère. Tandis que l'ETL est impliqué dans des transformations complexes, l'ELT entre en jeu lorsque d'énormes volumes de données sont impliqués. De plus, étant donné que les données sont directement chargées dans le système cible, l'ELT demande moins de temps que l'ETL. (toujours à cause des deux étapes différentes qu'implique son processus, la maintenance ainsi que le coût sont plus élevés.).

3) Ajout possible d'un Data Mart au DataWarehouse

On a vu qu'il est possible d'ajouter une structure dite de staging (comme dans l'ETL) afin de faciliter l'utilisation d'outils d'analyse et de reporting en convertissant les données dans un format structuré résumé. En plus de cette structure de staging, il est possible d'ajouter des DataMarts. Ceux-ci rassemblent un ensemble de données organisées, ciblées, agrégées et regroupées dans le but de répondre aux besoins de l'utilisateur final. (axysweb.com. ETL vs ELT : comment faire son choix entre ces 2 processus ?)

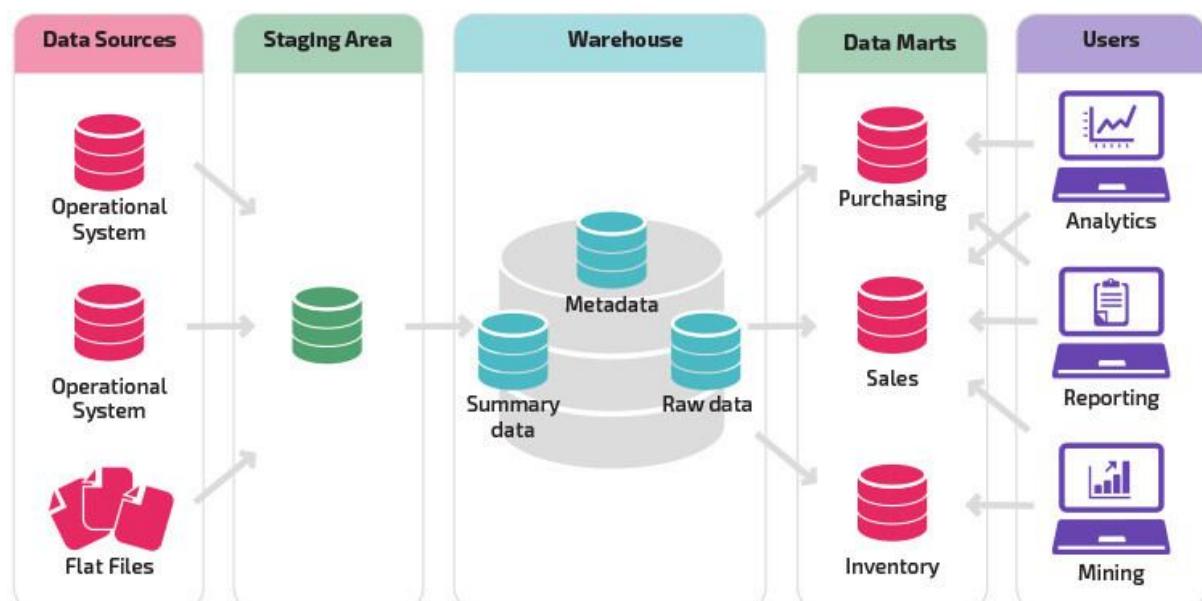


Figure 4: Fonctionnement avec un Data Mart ajouté

4) Avantages et Enjeux d'un DataWarehouse Cloud

Le DataWarehouse Cloud devient de plus en plus prisé pour diverses raisons. Premièrement, le coût. En effet, le calcul et le stockage étant séparé, si vous avez besoin de plus d'espace de stockage, vous n'avez qu'à acheter ce qui est essentiel. Deuxièmement, un DataWarehouse Cloud est rapide à déployer : en quelques minutes seulement, des professionnels peuvent construire leur Data Warehouse. Troisièmement, étant donné que la disponibilité, l'évolutivité et la performance sont meilleures sur Cloud, un Data Warehouse Cloud permet d'avoir les informations plus rapidement et plus précisément. Quatrièmement, la gestion pouvant être automatisée, les équipes informatiques n'ont plus besoin de faire de la gestion de l'entrepôt de données une priorité. Et enfin, un Data Warehouse Cloud permet d'avoir des données plus sécurisées.

Néanmoins, la mise en place d'un Data Warehouse Cloud présente encore quelques défis. Tout d'abord, le chargement des données est généralement réalisé à l'aide d'un autre outil. De plus, la sauvegarde des données nécessite une surveillance et une attention particulière. Enfin, l'optimisation du cluster et des requêtes doit être continuellement ajustée. (Oracle.com. Qu'est-ce que le cloud? | Définition du cloud | Cloudflare)

Les datawarehouses basés sur le cloud ont des architectures uniques. En voici deux exemples :

a) Amazon Redshift

L'exemple d'Amazon Redshift est pertinent car nous avons ici un exemple de DataWarehouse sur le Cloud utilisant une architecture traditionnelle.

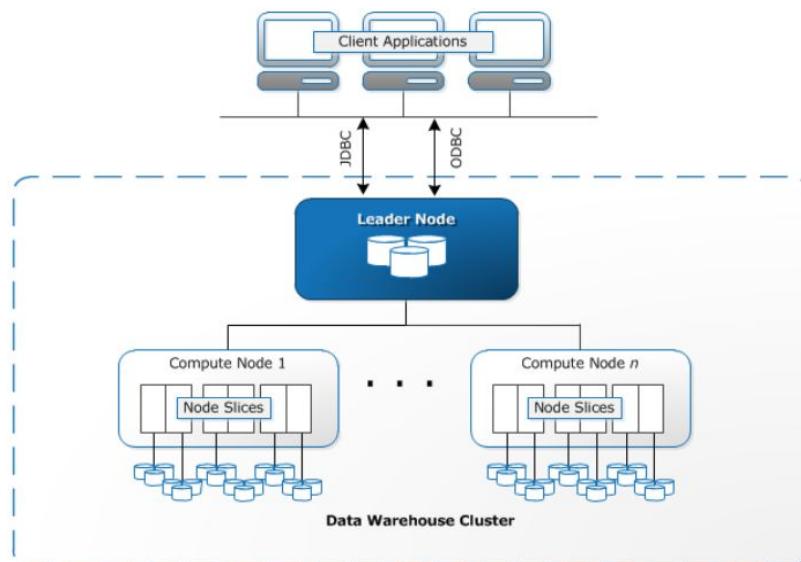


Figure 5 : Data Warehouse Cluster

En effet, comme on peut le voir sur la figure 5 , les ressources sont mises en place dans un cluster composé de plusieurs nœuds. Chaque nœud est découpé en tranches, dans lesquelles les données sont stockées. On observe également un nœud principal qui permet de compiler les requêtes afin de les faire exécuter par les nœuds de calculs. Les requêtes sont plus rapides car les nœuds de calculs traitent les requêtes dans chaque tranche simultanément. Le nœud principal renvoie ensuite les résultats à l'application client. Les applications clients qui peuvent être des outils de BI (business intelligence) et d'analyse peuvent se connecter à Redshift via des pilotes Open Source PostgreSQL JDBC et ODBC. Amazon Redshift utilise un processus ETL. Pour plus de renseignements, aller voir "Architecture système de l'entrepôt des données" d'Amazon en annexe (Annexe 1).

(Aws.Amazon.com. Data warehouse system architecture)

b) Google BigQuery

Google BigQuery est construit autour de la technologie Dremel (cf Annexe 2). L'architecture de BigQuery est représentée dans la figure ci-dessous:

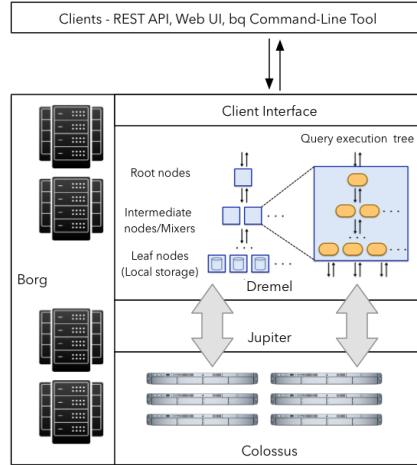


Figure 6: Architecture de Big Query

BigQuery repose sur plusieurs composants. Tout d'abord, Dremel est ici le moteur d'exécution de requêtes SQL : il est capable de scanner des milliards de lignes en très peu de temps. Colossus est un système de gestion de fichier sous-jacent, Jupiter un réseau permettant à Dremel de lire les données de Colossus et Borg un système de gestions de clusters à grande échelles : il permet d'assigner automatiquement les ressources de calcul et de stockage des serveurs à des tâches individuelles, plutôt que d'avoir à le faire manuellement.

On remarque qu'ici BigQuery sépare les concepts de stockage, représenté ici par Colossus, et de calcul, représenté ici par Borg.

(Panoply.io. A Deep Dive Into Google BigQuery Architecture) (Lebigdata.com. Google BigQuery : tout savoir sur la plateforme Cloud de Big Data)

C. Mise en place d'un Data Warehouse

Un Data Warehouse est donc une base de données regroupant des informations provenant de sources multiples. Les données sont alors transformées et organisées dans le seul but de permettre à l'entreprise d'avoir une aide à la décision et de produire facilement des rapports depuis ces données. Avant de commencer à créer son Data Warehouse il est donc nécessaire de définir les besoins de l'entreprise concernée. Ces besoins doivent servir de guides tout au long de la conception puis de la création de l'entrepôt.

Après avoir identifié les besoins, il est important d'identifier toutes les sources différentes de données qui devront téléverser dans notre Data Warehouse. Ainsi, nous pourrons toutes les y intégrer sans surprise. Plus cette liste sera complète, plus la suite de notre travail sera simple. En effet, ajouter à postériori une source de données qui n'avait pas été prévue dans notre modèle est une étape plus complexe.

Une fois ces informations regroupées et analysées, plusieurs solutions s'offrent à l'entreprise. Le développeur peut créer son Data Warehouse "from scratch", soit en le codant à partir de zéro. Cette solution est la plus lourde et pour qu'elle soit appliquée sans accrocs, une certaine expérience de développement de ce genre de systèmes est nécessaire. Sinon, de multiples solutions logicielles existent aujourd'hui aussi bien pour créer un Data Warehouse en local ou sur le cloud.

Ici, nous allons présenter brièvement un certain nombre des solutions web et logicielles permettant la mise en place pas à pas d'un entrepôt de données. Plusieurs entreprises importantes telles que Oracle, Amazon, IBM et bien d'autres proposent aujourd'hui chacune des outils dans ce sens.

1/ Des solutions pour le cloud

Aujourd'hui, pour des questions de rapidité du calcul et de budget, les solutions cloud sont les plus sollicitées. Elles offrent une grande flexibilité aux entreprises et se mettent en place facilement. Beaucoup d'acteurs de l'informatique et du web se sont donc lancés sur ce marché depuis plusieurs années. En voici trois , couramment choisis pour leurs fonctionnalités. Néanmoins, il existe de nombreux autres fournisseurs dans ce domaine aux solutions toutes aussi performantes.

Autonomous Data Warehouses par Oracle :

Créé en 1977 Oracle est aujourd'hui un acteur du web incontournable. Leur solution, Autonomous Data Warehouse offre ainsi un certain nombre d'avantages.

Elle est tout d'abord reconnue pour sa grande simplicité. Faisant appel à l'intelligence artificielle pour gérer le Data Warehouse, Oracle permet à l'administrateur de cette base de données de se dégager d'une quantité non négligeable de tâches. C'est ainsi l'IA qui se charge de piloter la base de données, de sa sécurité et de sa réparation le cas échéant.

Cette solution est également rapide, aussi bien à mettre en place que lors de son utilisation. En effet, pour créer un entrepôt de données avec Oracle, très peu d'informations sont nécessaires à fournir : le data center choisit, le nombre d'OCPU (Oracle Compute Units) et de Terabytes souhaités, ainsi que le nom d'utilisateur et le mot de passe de l'administrateur de la base. Suite à quoi il ne reste qu'à charger ses données dans l'entrepôt créé et nous pouvons commencer les requêtes. Par la suite, c'est l'utilisation dans le Data Warehouse de Oracle Exadata qui le rend plus rapide que la moyenne au quotidien.

Enfin, cette solution offre une grande liberté d'adaptation. En effet, la quantité des ressources utilisées ou la taille de l'entrepôt peuvent être augmentées ou diminuées à tout moment par l'administrateur sans que cela ne ralentisse le système.

(Oracle.com. Présentation de la solution Autonomous Data Warehouse)

Amazon Redshift par Amazon :

Connu pour la vente par correspondance, Amazon n'était pas forcément attendu sur le créneau des Data Warehouse. Cependant, il s'agit d'une solution couramment utilisée par les entreprises et fiable. Pour pouvoir être utilisé, Amazon Redshift ne demande que peu de

prérequis : posséder un compte AWS et une maîtrise, somme toute logique, du langage SQL.

Tout comme la solution Oracle, Amazon Redshift offre une réponse simple et rapide et sécurisée pour la création d'un Data Warehouse. Celle-ci peut être mise en place en quelques minutes et sa présence sur le cloud permet la diminution des temps de calculs lors du requêtage.

Amazon permet ici de créer son entrepôt de données pas à pas en suivant des étapes claires accompagnées de nombreuses aides. Il suffit alors de se laisser porter en suivant les instructions pour obtenir un Data Warehouse complet et stable sur lequel il sera facile de travailler.

(Aws.Amazon.com. Déploiement d'un entrepôt de données)

IBM Db2 Warehouse par IBM :

Également incontournable dans le secteur de l'informatique, il est logique de retrouver IBM parmi les fournisseurs de solution pour la création de Data Warehouse les plus sollicités.

Une fois encore, il s'agit ici d'une solution cloud fiable offrant une grande flexibilité pour les entreprises. La taille de l'entrepôt et les ressources disponibles au calcul peuvent être gérées en temps réel pour répondre au mieux au besoin du client. Les réponses aux requêtes sont rapides même pour des analyses plus complexes et lourdes. Une intelligence artificielle est là encore présente et peut être en partie contrôlée par l'utilisateur qui peut créer des modèles d'apprentissage. Enfin, cette solution permet également de posséder jusqu'à sept sauvegardes de son entrepôt de données et de planifier des sauvegardes dans le temps.

(Ibm.com. IBM Db2 Warehouse on Cloud)

2/ Des solutions sur site

Contrairement aux solutions cloud, les solutions dites sur site, qui permettent de créer un Data Warehouse sur un serveur local de l'entreprise, sont plus longues à mettre en place. Cependant, ils restent de loin avantageux sur ce point en comparaison avec le développement de zéro d'un entrepôt de données. Dans ce domaine, il existe à la fois des logiciels payants et des solutions open source diverses. Voici pour exemple l'un de ces logiciels :

Azure Synapse Analytics par Microsoft :

Microsoft propose ici une solution logicielle payante combinant un entreposage des données de qualité et la puissance de l'analyse Big Data. Synapse Analytics est l'un des logiciels de la gamme Azure de Microsoft dont plusieurs sont en lien avec l'entreposage de données ou leur utilisation. Il s'agit d'une solution professionnelle et sûre qui apporte aux entreprises de nombreux avantages tels que la mise à disposition d'outils d'analyse, de gestion ou encore de sécurité variés et l'interopérabilité de ces outils.

Ici encore, les données sont intégrées automatiquement par le logiciel et des options de machine learning sont disponibles pour gagner du temps lors des analyses et créer des modèles.

(Microsoft.com. Azure Synapse Analytics)

Une fois de plus, cette liste est loin d'être exhaustive et quelques recherches plus approfondies permettront à chaque entreprise de trouver la solution, cloud ou sur site, correspondant le mieux à ses besoins.

III/ Data Lake

A. Définition

C'est Dorian Pyle qui en 1999, parla pour la première fois de Data Lake dans son livre "Data preparation and data mining". Quant à James Dixon de l'entreprise américaine Pentaho, c'est lui qui a mis en avant le concept en 2010.

Le Data Lake est utile aujourd'hui pour faire face au big data. Le concept de big data apparu fin du 20ème siècle et désignait un nouveau groupe de données de très grand volume, très différentes. On associe souvent le big data aux trois "v" : Volume, Vitesse, Variété. En effet, les données sont reçues et traitées en temps réel, ou quasi-réel. De plus, les jeux de données qui composent le big data sont très complexes. Le big data est très utile pour le commerce, la recherche ou encore pour la sécurité (fraudes etc) et permet d'avoir des réponses, des études plus précises grâce au grand volume d'informations disponible.

En français, Data Lake se traduit par lac de données. En effet on pourra voir que cette métaphore permet d'expliquer le concept de Data Lake qui va permettre de stocker énormément de données sans se soucier de la limite de stockage, grâce aux clusters qui peuvent être déployés sur site ou dans un cloud. Un cluster est un agrégat, un groupe de ressources qui permettent ici de stocker les données. Ces ressources forment une unité qui se comporte comme un seul système de stockage et permet donc l'exploitation et le stockage massif de données.

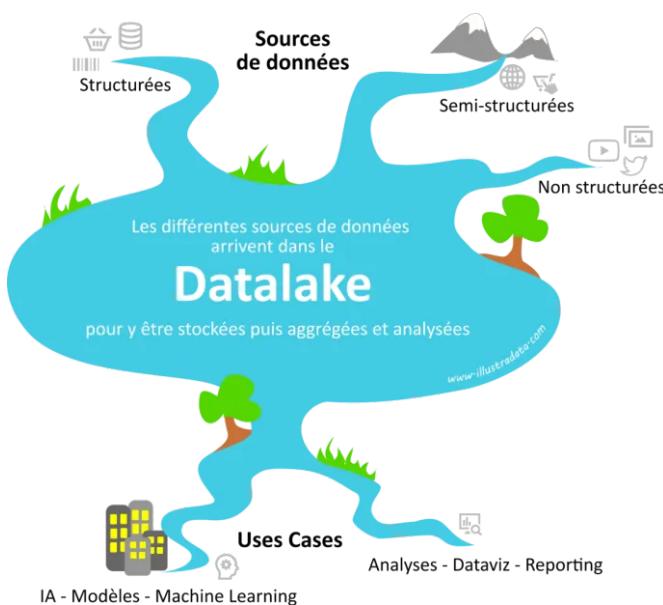


Figure 7 : La métaphore du Lac

L'image du lac et de l'eau qu'il contient permet de comprendre que les informations sont stockées sans ordre particulier et sans contrainte. Comme le montre la figure 7, les différentes façons d'alimenter le lac représentent les différents moyens d'approvisionner le Data Lake. Les données peuvent en effet être stockées sous forme brutes, transformées ou semi-transformées. C'est à dire que des données de type base relationnelles, en ligne ou en colonnes, des données de type log (données de type 'journal' qui représentent tous les

événements, comme l'heure de consultation, la date, l'adresse ip, d'un ordinateur ou d'un serveur) peuvent cohabiter avec des données type e-mails, documents pdf, et même des données binaires comme des photos ou autre. Ce moyen de stockage est très différent des autres. Les données sont classiquement structurées puis stockées, seulement, la structuration de données avant stockage peut être coûteuse et complexe et les bugs de structuration peuvent amener à la perte de données. Cependant, contrairement au Data Lake, la structure des informations avant stockage facilite la navigation entre elles. Ce qui est plus compliqué dans le référentiel du Data Lake. Cependant, le but de celui-ci n'est pas de connaître l'utilisation des données stockées et donc n'a nul besoin de les structurer avant stockage. Les informations sont physiquement stockées dans l'entreprise ou dans le cloud dans des bases de données distribuées (comme avec HDFS, Hadoop Distributed File Storage), dans des bases NoSQL ou dans des bases objets telles qu'Amazon S3 ou Azure Blog Storage.

Ces données peuvent venir d'un système interne ou externe. Ceci permet de gagner du temps car le coût habituel de transformation des données est très long et le Data Lake permet d'éviter cela. De plus, ces informations peuvent être lues et utilisées comme un plongeur qui pourrait observer le fond du lac où en sortir des échantillons pour les analyser. La figure 7 montre le parallèle avec le lac, et la possibilité de récupérer les données, représenté par les deux rivières sortant du lac et allant vers les plaines. Il s'agit en effet de comprendre qu'un Data Lake permet non seulement de stocker des données massivement, mais aussi de les récupérer afin, soit de les utiliser soit de les analyser. Les usages les plus courants de Data Lake sont le machine learning, l'analyse des données, la création de rapports ou de tableaux de bord. Il peut aussi être associé à d'autres outils comme l'intelligence artificielle ou l'analyse en temps réel.

Le Data Lake a des avantages certains : il ne demande pas de structurer les données avant de les stocker, mais juste de les nettoyer et de les structurer avant l'utilisation, ce qui permet de pouvoir garder au maximum le potentiel des données stockées. Les données sont donc gardées "pures" et permettent un plus grand panel d'analyses, différentes, exhaustives et transversales puisque les informations seront éventuellement structurées une fois que l'on saura ce que l'on en fera, ce qui fait la flexibilité du Data Lake. Il permet donc pour les entreprises par exemple, une meilleure exploitation des données. Pour la collecte d'informations et leur analyse sur des clients cela peut devenir très avantageux, vu la quantité de données possiblement stockées. On peut voir également que le Data Lake peut être configuré ou reconfiguré à loisir que ce soit par les développeurs, les analystes ou les data scientists.

Cependant, il y a des revers au Data Lake. En effet, celui-ci demande des outils beaucoup plus poussés et des compétences plus élevées pour le manipuler. On peut remarquer aussi que le Data Lake peut vite devenir un frein si il n'est pas correctement architecturé, puisqu'il n'y a ni catégorisation ni hiérarchisation en son sein. Certaines entreprises en sont revenues, après avoir essuyé des problèmes de récupération de données sans aucune règle de "rangement". Le Data Lake peut vite devenir une "poubelle" où les données sont juste entassées "au cas où". En outre, si le Data Lake a un coût de stockage de données dix à vingt fois plus faible que le Data Warehouse par exemple, il a besoin d'être alimenté en données, et la créations des chemins permettant cela est souvent coûteuse en temps et en ressources. Sans compter que le Data Lake permet de stocker une quantité monstrueuse de données, et donc les entreprises peuvent stocker beaucoup d'informations, ce qui pose aussi une question d'éthique sur la place de la confidentialité des données.

B. Architecture d'un Data Lake

Dans un premier temps nous verrons dans quel but sont créés les Data Lakes pour mieux comprendre son architecture propre..

Les Data Lakes donc, sont construits dans un but de gérer un très grand nombre de données, qui arrivent rapidement. Cela peut être des données structurées venant des bases de données de l'entreprise ou des données non structurées venant de médias nouveaux. C'est pour ces raisons que le Data Lake doit être flexible c'est-à-dire qu'il est munie de nombreuses applications dynamiques. Une application dynamique est une application qui n'a pas besoin d'un tiers pour se déclencher et s'adapter, en effet grâce aux algorithmes de machine learning implantés par les data scientists, elles apprennent à réagir. Par exemple lors d'une affluence conséquente et récurrente elle est en mesure de créer d'elle-même des instances dans le back end pour que le serveur tienne le coup. Enfin à terme, une entreprise espère pouvoir économiser du temps et des ressources sur des travaux de maintenance. En effet la plupart du temps de ces data scientist est occupé à l'administration de leur base de données et non à l'élaboration d'algorithme.

Le Data Lake est construit selon une architecture plate. Les données ne sont pas ordonnées dans une arborescence par exemple, elles ont juste chacune un identifiant propre, et des tags lui sont attachés pour la caractériser plus facilement. En tout cas dans un Data Lake le plus important est d'enregistrer la date d'arrivée de chaque donnée. Tout ceci est basé sur le concept du Data Lake, le modèle de données schema on read, le rendant ainsi très flexible. Les données sont donc souvent découpées en 3 parties, par exemple les données de moins 6 mois, les données plus vieilles mais encore utilisées et enfin les données archivées qui ne sont plus utilisées mais qui peuvent encore être utiles. Cela sert à simplifier l'analyse des données, on peut par exemple utiliser des petits paquets de données ou alors de grands paquets de données pour avoir une analyse plus complète. C'est ce que l'on appelle respectivement, le stream processing et le batch processing. L'un est basé sur la rapidité de l'analyse, rendant ainsi possible l'analyse en temps réel, tandis que l'autre est plus axé sur la qualité de celle-ci. L'architecture d'un Data Lake est donc en constante augmentation de sa capacité en même temps qu'elle accueille de nouvelles données.

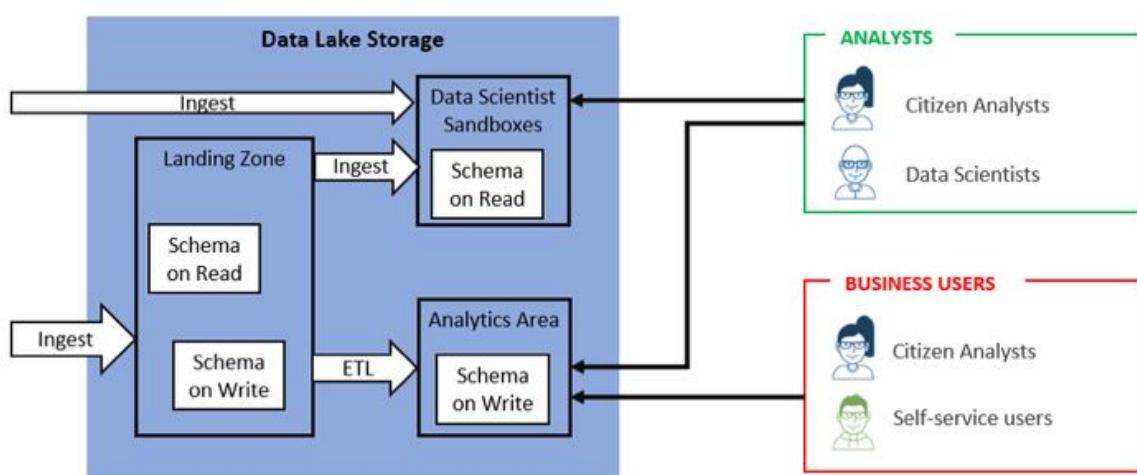


Figure 8 : Fonctionnement simplifié du Data Lake

Comme nous pouvons le voir le Data Lake stock différentes sortes de données certaines sont traitées par le schéma en lecture, celles-ci sont souvent utilisées pour faire des interfaces dont les données s'actualisent dynamiquement. C'est donc dans la les mains des data scientists qu'elles tombent pour qu'elles soient analysées par leurs algorithmes. Les autres sont redirigées vers une analyse plus classique.

(Journaldunet.com. Application dynamique : anatomie d'une technologie au service de la compétitivité des entreprises) (Oracle.com. Data Lake : Définition) (Miloslavskaya and Tolstoy)

C. Mise en place

Le Data Lake a besoin d'une stratégie bien définie pour être bénéfique à l'entreprise qui va l'utiliser. En effet, pour une meilleure utilisation d'un Data Lake, il est nécessaire d'identifier l'utilisation des futures données sélectionnées. Et si l'usage des données qui vont être stockées est important, il faut aussi se pencher sur la qualité des données stockées.

Trois points principaux doivent être étudiés avant de faire appel à un Data Lake dans l'entreprise : d'abord, il va falloir que l'entreprise définitse des cas d'usages qui vont devoir être hiérarchisés en fonction de leur apport économique, leur coup, leur complexité. Ils peuvent être une amélioration des cas d'usages précédents, tels qu'une amélioration de la rapidité, de la précision, etc et ont besoin de nouvelles sources d'informations, qu'elles soient internes ou non. Cependant il peuvent aussi viser l'innovation. Ensuite, l'entreprise a besoin d'un plan d'action précis pour pouvoir avoir la direction dans laquelle l'entreprise a besoin d'emmener le Data Lake. Pour finir, cela nécessite de faire un choix sur l'infrastructure. On peut remarquer que les entreprises sont de plus en plus friandes du cloud computing, qui désigne l'utilisation de cloud au lieu d'un stockage local. Ainsi, les entreprises ont donc le choix entre des stockages internes ou externes, managés ou non et toutes les infrastructures ont un intérêt propre qui pourra correspondre ou non aux besoins de l'entreprise, mais le coût et la complexité vont être aussi très différents selon les infrastructures.

Pour avoir un Data Lake fiable, il est nécessaire d'avoir des outils qui s'adaptent à l'itératif et à l'interactif pour avoir un lien stable et fiable entre utilisateurs et données. En effet, le Data Lake, lorsqu'il est utilisé en entreprise, doit permettre aux utilisateurs de comprendre et de pouvoir interagir simplement avec celui-ci. La simplicité de l'interface doit donc être prise en compte également.

Le déploiement d'un Data Lake n'est donc pas quelque chose de très simple, et nécessite du recul et de la réflexion sur l'entreprise. Il est alors presque indispensable de mettre une équipe sur pied pour gérer un Data Lake. Ces équipes doivent en général contenir des data scientists, des data engineers des data architects, des experts de la visualisation de données et des administrateurs systèmes. Toutes ces spécialisations apportent respectivement des compétences en mathématiques et statistiques, en connaissance des technologies du big data, en architectures informatique, en design de l'information et de sa récupération, et le maintien technique des réseaux, machine etc. Sans oublier un besoin de personnes compétentes en droit afin de traiter toutes les questions éthiques et réglementaires que pose le Data Lake (droit de récupération des données, transparence, utilité, temps de stockage, ...).

Un Data Lake peut être géré par différents environnements, comme vu plus haut, et ceux-ci sont souvent basés sur la technologie hadoop. Ils peuvent permettre de gérer le Data Lake localement ou alors dans le cloud.

Nous pouvons par exemple parler de certains outils utilisés pour faire un Data Lake, par exemple Amazon Web Services qui a créé tout un écosystème d'outils permettant la mise en place de Data Lake. Nous allons donc essayer de décrire comment ça marche.

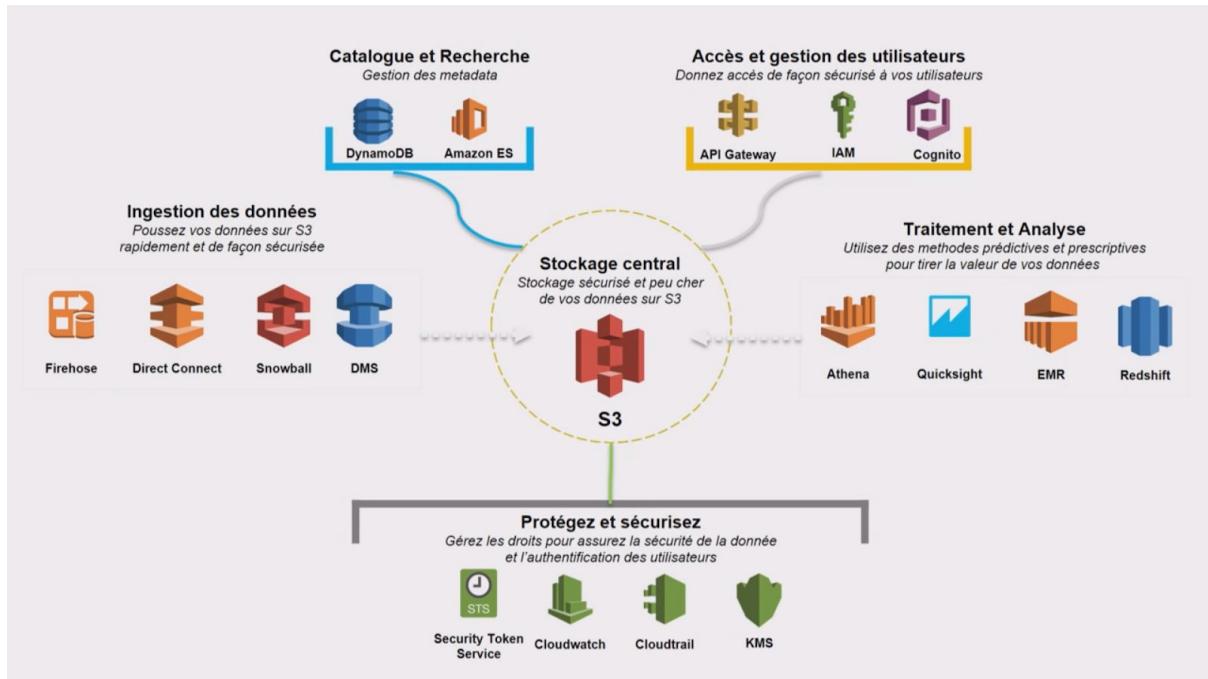


Figure 9 : Écosystème AWS (Amazon Web Service France. Créez son Data Lake avec AWS - AWS Summit Paris 2018)

En premier lieu nous avons le stockage central qui est le point de concours de tous les autres outils. Celui-ci est nommé Amazon Simple Storage Service, abrégé en Amazon S3. Il permet de stocker toutes les données d'un Data Lake. D'après Amazon, leur qualité de conservation des données est exceptionnelle (99.9999999999% de conservation selon le site d'Amazon). Cela se traduit par le fait qu'en mettant 10 000 fichiers sur le S3, il faudra attendre 10 millions d'années pour qu'un fichier soit perdu. La qualité de sécurité des données est également mise en avant par la marque américaine, ce qui semble en effet primordial pour un outil qui stocke autant de données.

Deux outils vont directement se greffer à S3 pour utiliser les données de celui-ci. En premier lieu nous allons parler des outils permettant l'ajout de données dans S3.

Firehose sert à charger les données de manière sécurisée et rapide dans S3 mais surtout de préparer ces données au format demandé par le Data Lake, facilitant ainsi le traitement de celles-ci plus tard.

Direct Connect quant à lui met plus l'accent sur l'optimisation des coûts en réseau, il permet par exemple de paramétriser facilement la connexion demandée de 1 à 10 Gbits/s et plusieurs connexions peuvent être programmées. Il permet de connecter les serveurs de l'entreprise et ceux d'AWS.

La gamme snowball peut-être utilisée dans des cas de stockage à la périphérie dans des endroits un peu éloignés des serveurs AWS comme des opérations militaires. Ce sont des appareils permettant la collecte, le stockage temporaire puis le transfert des données vers le serveur.

Enfin AWS DMS (Data Migration Service) permet comme le suggère son nom de faire migrer ses base de données vers AWS par exemple. L'efficacité semble au rendez-vous car le temps d'interruption durant la migration est très faible, simple d'utilisation avec une configuration rapide et aisée.

Ces quatres outils peuvent être utilisés selon les objectifs de l'entreprise ou du projet mais tous prennent des données extérieures pour les ajouter à S3 nous allons maintenant parler de quatres autres outils mais ceux ci permettant le traitement et l'analyse de Data Lake.

Premièrement nous allons parler d' Amazon Quicksight qui est en fin de compte l'interface homme machine et représente le résultat des requêtes. Il sera donc souvent associé avec les 3 autres outils utilisés pour l'analyse et le traitement des données.

Ensuite, le prochain de ces outils dont nous allons parler s'appelle Athena. Il permet de faire des requêtes liées à S3 dans le cas d'un Data Lake. Ce qui semble être mis en avant dans cet outil est la rapidité des requêtes, en effet la puissance de calcul mise à disposition est énorme.

De même EMR permet une solution de requêtage avec pourtant quelques différences, elle est basée sur une dizaine de projets open source dont Apache Hadoop. Il ne permet pas le lancement de requête sans serveur comme Athena mais est compatible avec bon nombres d'autres outils

Enfin Redshift quant à lui est une solution de Data Warehouse compatible avec les autres outils.

Il a fallu également ajouter des outils de catalogue et de recherche pour pouvoir repérer cite les données pertinentes dans un si grand volume. La gestion des utilisateurs ainsi que les quatres outils verts permettent eux d'améliorer la sécurité des données, ce qui semble être la pierre angulaire dans les arguments marketing d'Amazon.

III/ Comparaison et utilisations en santé

A. Utilisation, avantages, inconvénients d'un Data Warehouse

1/ Utilisation du DataWarehouse

On observe l'utilisation de DataWarehouse dans différents secteurs tels que les compagnies aériennes, le secteur bancaire, le secteur public , le secteur de l'investissement et de l'assurance, le secteur des télécommunications, de l'industrie hôtelière ou encore le domaine de la santé.

Chez les compagnies aériennes, le DataWarehouse est utilisé à des fins opérationnelles. On peut notamment citer l'affectation de l'équipage, les analyses de la rentabilité des itinéraires ou les promotions de programmes de voyageurs fréquents. Dans le secteur bancaire, l'utilisation des DataWarehouse sert essentiellement à gérer efficacement les ressources disponibles. Le secteur public, lui, utilise le DataWarehouse afin de collecter des informations. En effet, il aide les organismes gouvernementaux à tenir et à analyser les dossiers fiscaux pour chaque individu par exemple. Le DataWarehouse peut également être utilisé pour analyser les modèles de données, les tendances des clients et suivre les mouvements du marché surtout dans le secteur de l'investissement et de l'assurance. L'industrie hôtelière utilise le DataWarehouse pour concevoir et estimer leurs campagnes de publicité afin de cibler les clients en fonction de leurs commentaires et de leurs habitudes de déplacement. Enfin, nous verrons plus loin que le DataWarehouse peut également être utilisé dans le domaine de la santé.

2/ Avantages et inconvénients du DataWarehouse

Le DataWarehouse possède de nombreux avantages tels que :

- l'accessibilité rapide aux données
- enregistrement des modifications pour construire l'historique : suivi en temps réel tendances et changements (ex: commande annulée, changement statut du client,...)
- réduction du délai total d'exécution pour l'analyse et la production de rapports
- économie de temps : le DataWarehouse permet aux utilisateurs d'accéder aux données critiques à partir d'un nombre de sources en un seul endroit donc le DataWarehouse permet à l'utilisateur d'économiser le temps qu'il aurait passer à récupérer les données à partir de sources multiples.
- réduction du délai total d'exécution pour l'analyse et la production de rapports.

Le DataWarehouse possède également des inconvénients qui sont :

- assez complexe d'utilisation pour les utilisateurs moyens
- pas forcément l'option idéale pour les données non structurées
- difficile d'apporter des modifications dans les types et les plages de données, les index et les requêtes.
- les organisations doivent consacrer une grande partie de leurs ressources à la formation et à la mise en œuvre.

(Guru99.com. What is Data Warehouse? Types, Definition & Example)

B. Utilisation, avantages, inconvénients d'un Data Lake

1/ Utilisation du Data Lake

Le Data Lake est utilisé dans de nombreux secteurs, comme la santé, l'enseignement, les transports, le marketing, ... En effet, le Data Lake permet de stocker des données non structurées ou structurées, et ce rapidement et efficacement. Ainsi, pour le secteur de la santé, le Data Lake va permettre de stocker des données très différentes comme des rapports, notes des médecins, données cliniques. Et cela dans le but de pouvoir les consulter en temps réel ce qui est permis par le Data Lake. On peut également citer les données biologiques adn qui peuvent amener plus de mille téraoctets de stockage en une année d'après Astrazeneca.

L'utilisation du Data Lake se retrouve aussi dans l'enseignement, où les données souvent volumineuses et brutes qui viennent du secteur de l'enseignement vont être facilement stockées et gérées grâce à un Data Lake. Ces données peuvent être les notes et appréciations des élèves, mais aussi les coûts des établissements par exemple. Dans le secteur des transports, le Data Lake est également d'une grande utilité de par sa capacité de stockage en grand volume, mais également par toutes les informations qui en seront extraites et donc cela permettra de faire des prédictions sûres et efficaces. Cela permet de gérer, par exemple, les coûts et les approvisionnements que les entreprises de transports doivent gérer.

En ce qui concerne le marketing, les Data Lake permettent de stocker diverses données sur les clients afin de cibler au mieux leurs intérêts et leurs besoins.

Ils commencent également à être utilisés dans les secteurs de la fabrication, où ils vont permettre un meilleur retour par rapport aux produits et à leur utilisation ce qui peut permettre de les rendre plus efficaces et optimaux.

De plus, les Data Lakes sont certainement utilisés pour l'apprentissage des algorithmes du deep learning du fait du volume des données stockées.

(Talend.com. Différences entre lac de données et entrepôt de données)

2/ Avantages et inconvénients du Data Lake

Les avantages du Data Lake sont :

- Stockage rapide des données
- Coût peu cher
- Permet de stocker des données structurées, non structurés ou semi-structurées ou même des données brutes
- Permet de gérer et stocker massivement les données
- Permet à tous les types de données de cohabiter (au format natif et/ou transformées)
- les données sont plus facilement transformables ce qui fait que le Data Lake apporte de la flexibilité
- Le Data Lake permet une utilisation en temps réel
- Permet de mélanger données externes et internes

Les inconvénients du Data Lake :

- Pas de catégorisation ou hiérarchisation
- Avec la quantité de données stockées l'extraction et l'exploitation de celles-ci peuvent être compliquée
- La facilité de stockage des données créer des fuites de données, voir un stockage de données trop systématique qui peut être contre les règles de confidentialité les concernant
- Latence pour l'entreprise car les données sont stockées massivement et sont peut être physiquement éloignées

C. Quelle solution pour quelle utilisation (en santé) ?

Avant de voir les différentes utilisations en santé, il semble important d'abord de montrer en quoi les Data Lake et Data Warehouse diffèrent. En effet, même s'ils permettent tout deux le stockage de Big Data, ils ont des caractéristiques les différenciant, ceci permettant de pouvoir choisir entre les deux, celui semblant le plus adapté.

Le premier facteur les différenciant est la structure des données. En effet, les Data Lakes stockent généralement des données brutes non transformées. De ce fait, ils exigent généralement une capacité de stockage importante. De plus, les données brutes non transformées sont dites "malleables" c'est -à -dire qu'elles peuvent être analysées très rapidement et sont donc idéales pour le machine learning. De leur côté, les Data Warehouse stockent des données transformées et nettoyées donc utilisent nettement moins de stockage.

Le deuxième facteur les différenciant est la nature des données. En effet, les Data Warehouse stockent des données transformées c'est-à-dire qui ont déjà été utilisées à des fins spécifiques dans l'entreprise. De ce fait, l'espace de stockage ne sera pas pollué par des données qui ne serviront jamais. Au contraire, les Data Lakes stockent des données brutes qui n'ont pas forcément été utilisées par l'entreprise. Par conséquent, les Data Lakes sont bien moins équipés en fonctionnalités de structuration et de filtration des données que les Data Warehouses.

Le troisième facteur les différenciant est l'expérience utilisateur. En effet, l'exploration des Data Lakes est plus compliquée pour un utilisateur n'ayant pas l'expérience des données non transformées. Un data scientist ou des outils spécialisés sont souvent nécessaires pour comprendre et traduire les données brutes non structurées à des fins commerciales. De leur côté, les données transformées présentes dans les Data Warehouses sont généralement utilisées dans des graphiques, feuilles de calculs ou tables, afin de permettre à quiconque étant familier avec le sujet concerné de pouvoir les lire.

Le dernier facteur les différenciant est l'accessibilité à la manipulation. Tandis que les consultations ou les modifications des données peuvent être faites assez facilement dans les Data Lakes dans la mesure où ceux-ci n'ont pas de structures, et donc peu de restrictions, les modifications des données sont bien plus compliquées dans les Data Warehouses. En effet, les Data Warehouses sont bien plus structurés ce qui les rend plus difficiles et coûteux à manipuler dans la mesure où ils seront soumis à plus de restrictions sur la structure.

Ces deux méthodes de stockage sont bien différentes. Néanmoins, les entreprises ont souvent besoin des deux : les Data Lakes permettent d'exploiter des données brutes avec le machine learning tandis qu'un Data Warehouse est souvent nécessaire pour l'exploitation analytique par des spécialistes internes à l'entreprise.

En santé, les Data Warehouses sont utilisés depuis pas mal de temps. En effet, ils sont utilisés afin d'élaborer des stratégies et prévoir les résultats, produire des rapports sur les traitements des patients par exemple, partager les données avec les compagnies d'assurance,... Néanmoins, dans le secteur de la santé, les données sont souvent de type non structuré. On peut prendre comme exemple les notes des médecins ou les données cliniques par exemple. En sachant en plus que le personnel de santé a besoin de connaissances en temps réel, les Data Lakes semblent mieux correspondre au modèle idéal. En effet, vu qu'ils permettent de combiner données structurées et non structurées, ils conviennent généralement mieux aux prestataires de santé.

On peut également mentionner que l'introduction et la mise en place d'un Data Warehouse est un processus long (entre 12 et 24 mois) et que, pour les professionnels de santé, il faut attendre que le Data Warehouse soit entièrement terminé afin de pouvoir l'utiliser. Ce délai est donc également un frein. Néanmoins, il existe des entreprises telles que Health Catalyst, qui ont mis en place des solutions afin de réduire ce temps de mise en place. (cf Annexe 3)

Enfin, voyons dans quelles mesures légales il est possible de stocker des données de santé dans un Data Warehouse en France (CNIL. Traitements de données de santé : comment faire la distinction entre un entrepôt et une recherche et quelles conséquences ?). Tout d'abord, on demande le consentement explicite des personnes concernées par la collecte, l'enregistrement et la conservation des données. Si cela est possible, le responsable de traitement (professionnel de santé, structures de soins,...) doit informer de la manière la plus transparente et claire possible le patient concerné par la collecte de données. Le responsable de traitement n'a pas d'obligation quant au support d'information. En effet, l'information peut être dite à l'oral, écrite, affichée dans les lieux de soins ou dans les secrétariats par exemple.

Il existe donc deux hypothèses :

- Le consentement explicite de la personne concernée ,préalable à la collecte d'information, est obtenu via un formulaire en ligne par exemple. Dans ce cas, aucune formalité est nécessaire mais le responsable de traitement devra être en mesure de démontrer la conformité de son traitement au RGPD (Règlement Général sur la Protection des Données)
- Le consentement explicite de la personne n'a pas pu être obtenue donc une "demande d'autorisation santé" est formulée et la finalité de l'entrepôt de donnée doit avoir un intérêt public

Dans ces deux cas, une analyse d'impact sur la protection des données (AIPD) devra être réalisée par le responsable de traitement. Cette analyse devra alors être transmise avec le dossier de demande d'autorisation adressé à la CNIL.

Le Data Lake est plus controversé par rapport à la confidentialité des données. En effet, certains disent que le Data Lake est en contradiction complète avec les mesures légales qui lui sont imposées, car le Data Lake permet de stocker toutes sortes de données, délicates et confidentielles ou non, sans se soucier directement de leur future utilité ni de la date de leur utilisation. Le règlement européen veut que les données enregistrées dans un Data Lake soient spécifiques au traitement du patient. Il impose aussi une suppression des

données enregistrées pour un patient si elles n'ont plus d'utilité. Cependant, le data lake peut permettre au contraire de mieux respecter certaines règles de confidentialité des données, le droit d'opposition au profilage, le droit à l'oubli ou le droit à la portabilité des données. Si cela est possible, c'est aussi car les données sont plus rassemblées dans un Data Lake. Ces règles sont notamment explicitées dans le règlement général sur la protection des données (RGPD) et la loi française informatique et libertés. (Conseil National de l'ordre des médecins)

D. Qu'utilise le secteur de la santé de nos jours ?

Il semble également important de préciser qu'il existe de nos jours ce qu'on appelle les EDS (Entrepôts de Données de Santé). Ces EDS sont des outils informatiques permettant la collecte, l'intégration puis l'exploitation de données de santé provenant d'un grand nombre de sources d'information comme les dossiers patient ou les prescriptions informatisées par exemple. Un EDS a donc pour but de faciliter la réalisation de recherches dans le domaine de la santé et d'études relatives au pilotage hospitalier en regroupant dans une base unique l'ensemble des données recueillies dans les établissements." En effet , beaucoup d'établissements de santé ont compris que standardiser, expertiser, organiser et rendre accessibles les données est extrêmement bénéfique afin d'optimiser des recherches, détecter des profils de santé particuliers, renforcer la prise en compte des indicateurs de qualité ou encore développer des algorithmes d'aide à la décision. De ce fait, beaucoup de CHU revendique disposer d'un EDS mais seulement 3 d'entre eux ont obtenu l'autorisation de la CNIL à mettre en oeuvre un traitement automatisé de données à caractère personnel ayant pour finalité un EDS : Assistance Publique - Hôpitaux de Paris (AP-HP) depuis janvier 2017, le CHU de Nantes depuis juillet 2018 et le CHU de Lille depuis septembre 2019. Pour obtenir cette autorisation, il suffit de faire une demande d'autorisation à la CNIL comme lors de la création d'un Data Warehouse (en effet, EDS et Data Warehouse sont assez semblables). De plus, les EDS doivent définir des règles d'accès et de protection des données puisque , en plus du respect des règles liées à l'hébergement des données de santé, les EDS sont également soumis au cadre du RGPD. En effet, la préservation de l'anonymat des patients est primordiale. Mais tandis que l'anonymisation est relativement facile pour les données de santé structurées, les technologies de l'intelligence artificielle permettent elles l'anonymisation les données non structurées (compte rendus médicaux, images médicales,...) dont l'anonymisation est nettement plus difficile.

Les EDS ont néanmoins un gros problème : le manque d'interopérabilité. L'interopérabilité est la capacité d'un système informatique à communiquer, exécuter des programmes ou transférer des données avec d'autres produits ou systèmes informatiques, existants ou futurs, sans contrainte pour l'utilisateur dans l'accès ou la mise en œuvre, et sans multiplier les efforts de développements. Contrairement aux Etats-Unis ou au Royaume-Unis, la France a encore de gros efforts à déployer. En effet, le manque d'interopérabilité des logiciels, services et outils numériques des hôpitaux constitue un frein à l'amélioration de la qualité des parcours de soins. On peut, en effet, remarquer que certains établissements de santé possèdent jusqu'à 300 applicatifs différents ce qui conduit à des ruptures dans le parcours de soins.

L'autre problème rencontré est la difficulté pour le secteur public à attirer des profils qualifiés comme des data scientists de part le salaire moins bon que dans le privé. En effet, des compétences en ingénierie, informatique et sciences des données sont nécessaires pour gérer, analyser et exploiter des données massives.

De ce fait, les EDS avec un encadrement clair permettant la mise en place d'une organisation accordant une interopérabilité entre EDS seraient une grande opportunité pour les patients, les professionnels de santé et la santé publique. (HOUDART & Associés) Ci-joint, le site web de l'entrepôt de données de santé de l'AP-HP : <https://eds.aphp.fr/>

IV/ Secouriste de poche

A. Objectifs du projet

Le projet du *Secouriste de poche* intervient en réponse à un constat simple et tristement affligeant : il n'existe aucun site internet dédié aux secours d'urgence. Plus précisément, internet répertorie un grand nombre de sites dédiés à la formation aux gestes qui sauvent mais pas un seul, facile d'accès qui disent précisément, sans ambiguïté et surtout sans fioriture, que faire dans une situation d'urgence lorsque l'on s'y trouve.

En effet, si une personne s'étouffe devant vous, vous n'avez pas vraiment le temps de regarder une vidéo de dix minutes précédée de deux publicités ou de lire des paragraphes de textes sur trois pages internet différentes et des dessins peu précis dont plusieurs étapes manquent pour comprendre comment aider cette personne. Or, aujourd'hui c'est ce qu'il faut faire pour obtenir une réponse.

De plus, nous avons également fait le constat que les sites regroupant des informations sur les gestes de secours sont rarement complets. Il s'intéressent bien souvent à une ou deux situations particulières et ne traitent pas l'ensemble des situations d'urgence les plus fréquentes. Certaines pages ne traitent ainsi que de l'arrêt cardiaque, d'autres uniquement de l'hémorragie...

C'est donc de cette analyse qu'est née l'idée du *Secouriste de poche*. Développé sous la forme d'un site web compatible avec un affichage sur smartphone et si possible à terme d'une application mobile, il s'agit d'y regrouper des tutoriels simples et rapides des gestes à réaliser en situation d'urgence. Sans longs textes d'explication et avec des illustrations précises de chaque action pour ne pas laisser de place à l'incertain, ces fiches seraient le plan de sauvetage pas à pas pour répondre à chaque cas particulier.

Se trouvant devant une victime, l'utilisateur pourrait ainsi sélectionner rapidement grâce à un système de filtres la fiche qui répond à sa situation. Il lui suffirait par exemple de cliquer sur les réponses "non" aux questions : la personne est-elle consciente et la personne respire-t-elle pour avoir directement accès à une fiche sur le massage cardiaque.

Enfin, puisqu'il reste tout de même très important de former la population à ces gestes de premiers secours, un onglet formation serait également accessible. Dans cet onglet, l'utilisateur pourrait retrouver l'ensemble des fiches de secours mais aussi des informations plus complètes et des explications sur les différentes situations de secours. Une fois de plus, il pourrait filtrer les chapitres qu'il souhaite consulter, en placer certains en favoris ou encore y ajouter des commentaires pour poser des questions, proposer des modifications ou apporter des précisions pour les autres utilisateurs. Pour finir, des situations d'exercice seraient proposées sous forme de QCM pour s'entraîner après avoir lu les différents chapitres d'un thème. Elles permettraient à l'utilisateur de s'assurer qu'il a retenu les informations importantes ou bien qu'il est toujours à jour de ses connaissances. En effet, le PSC1 (Prévention et Secours Civique de niveau 1) étant comme d'autres un diplôme dont le bénéfice est acquis à vie, peu sont les personnes qui décident de le repasser régulièrement. Cependant, même les secouristes aguerris recyclent leurs connaissances de façon récurrente car ils oublient les gestes qu'ils ne réalisent pas souvent. Il est donc tout aussi important pour l'utilisateur qui réalise ces gestes encore moins souvent

de les réviser régulièrement. De même, les procédures de secours étant amenées à changer, il est important de lire de temps en temps les nouvelles marches à suivre pour se tenir à jour.

B. Modèle conceptuel de données

Le site que nous sommes en train de concevoir devra regrouper une quantité importante de données de natures diverses. De plus, il faudra que l'utilisateur soit en mesure d'accéder aux données qui correspondent aux différents choix qu'il fera. Il va donc être nécessaire de créer une base de données pour les regrouper, les organiser et les rendre accessibles.

D'après les objectifs que nous avons fixés dans la partie précédente, nous pouvons dès lors et déjà définir l'ensemble des entités et des associations qui constitueront notre base de données. En effet, après avoir analysé notre projet, nous obtenons les observations suivantes :

- Le site contiendra des fiches caractérisées par au moins un guide.
- Les guides sont définis par un titre et une illustration. Ils sont la marche à suivre dans une situation d'urgence précise.
- Il existera également des cours, qui possèderont un titre, les guides associés, des explications, et potentiellement des médias (vidéos, audios...).
- Chaque cours est abordé dans un chapitre, également décrit par un titre. Un chapitre est constitué d'au moins un cours.
- De même, chaque chapitre devra appartenir à un thème possédant une fois de plus un titre. Un thème peut contenir un ou plusieurs chapitres.
- Les médias possèdent un titre, et une ressource audio ou vidéo.
- Le site contiendra également des QCM. Un QCM est caractérisé par un numéro et concerne au moins un thème. Il possède également un fichier de questions, un fichier contenant les réponses possibles à chaque question et un fichier de correction contenant les réponses vraies aux différentes questions.

Nous pouvons donc proposer le modèle conceptuel de données suivant pour ce projet :

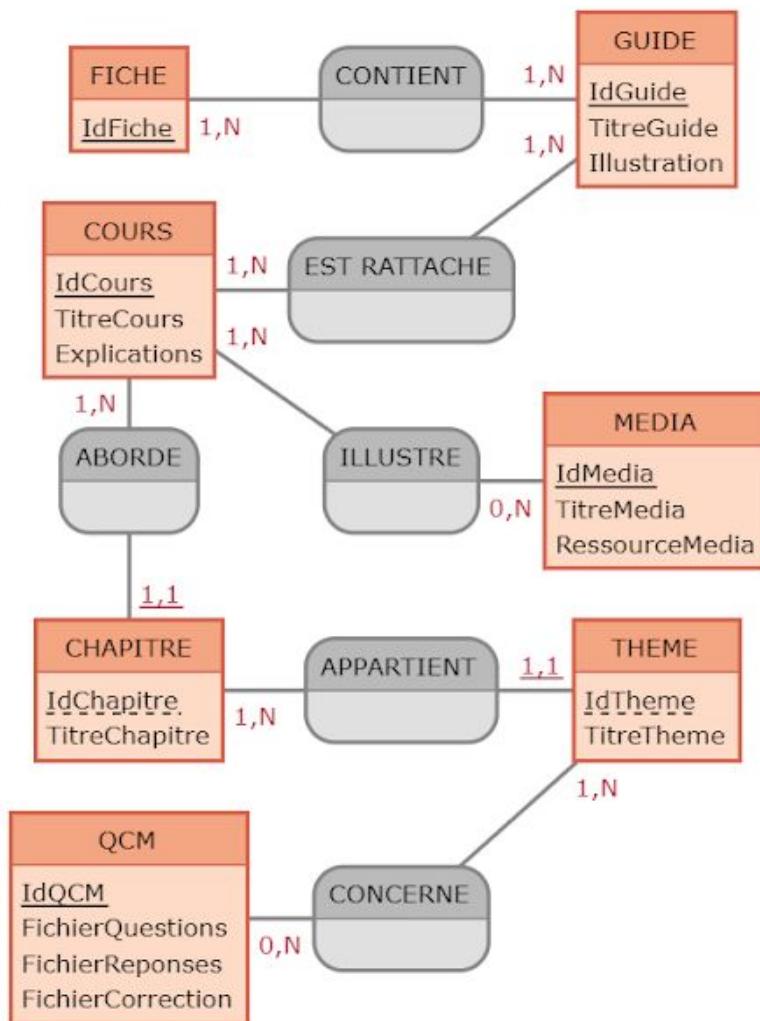


Figure 10 : Modèle Conceptuel de Données du secouriste de poche

Ce qui correspond au modèle relationnel suivant :

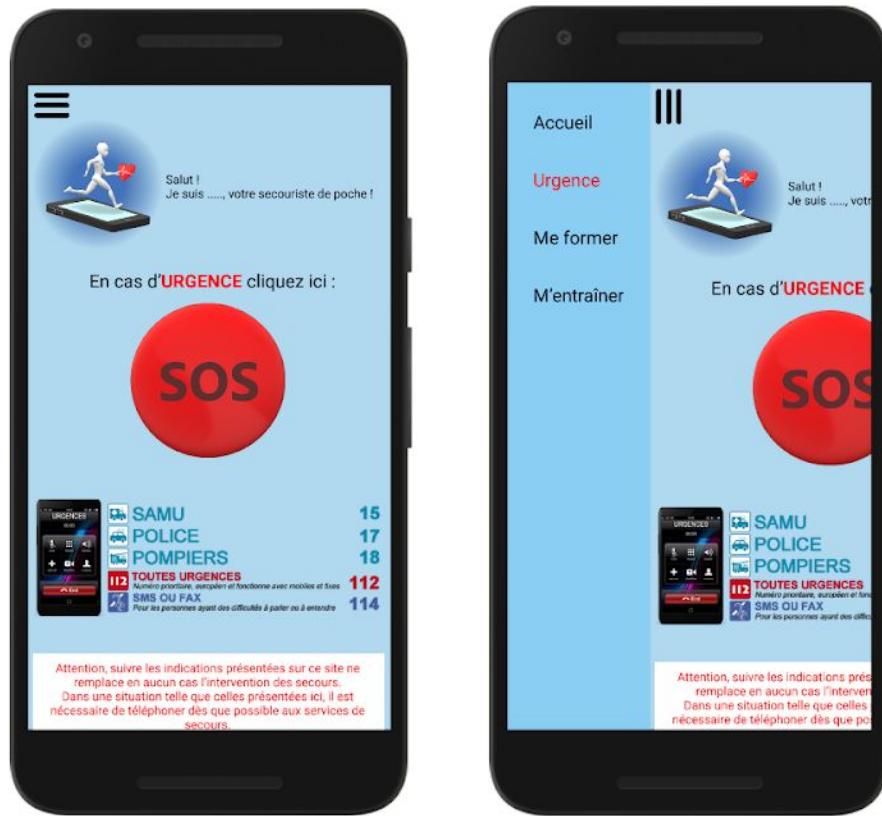
```

    CONTIENT ( IdFiche, IdGuide )
    GUIDE ( IdGuide, TitreGuide, Illustration )
    COURS ( IdCours, TitreCours, Explications )
    EST_RATTACHE ( IdCours, IdGuide )
    ILLUSTRE ( IdMedia, IdCours )
    MEDIA ( IdMedia, TitreMedia, RessourceMedia )
    CHAPITRE ( IdCours, IdChapitre, TitreChapitre )
    THEME ( IdCours, IdChapitre, IdTheme, TitreTheme )
    QCM ( IdQCM, FichierQuestions, FichierReponses, FichierCorrection )
    CONCERNE ( IdQCM, IdCours, IdChapitre, IdTheme )
    
```

Figure 11 : Modèle Relationnel du secouriste de poche

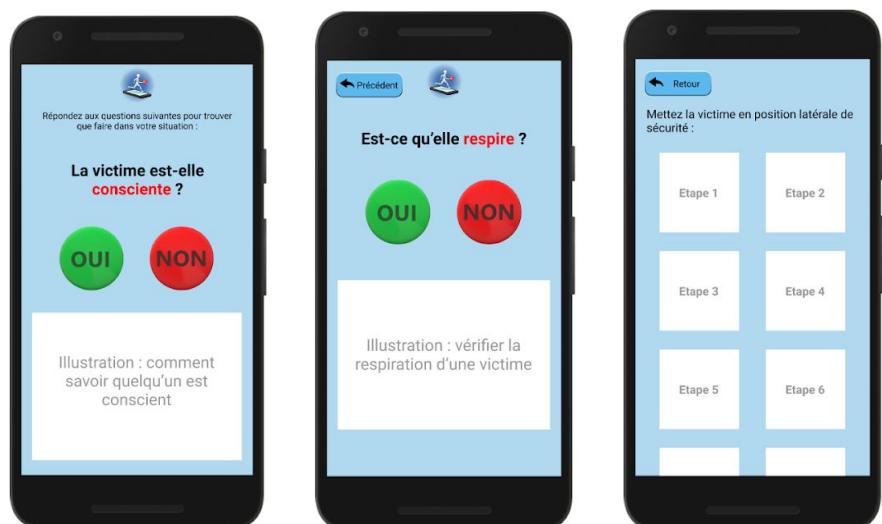
C. Maquette du site

Nous avons ainsi réalisé une première maquette du site que nous souhaitons concevoir :

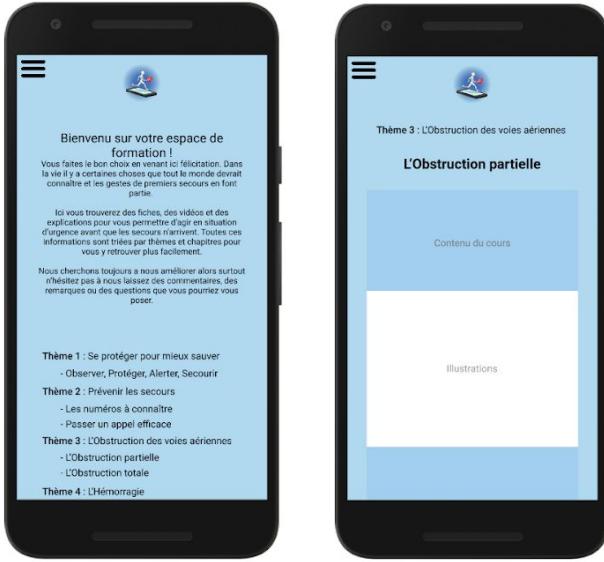


Ce site web contiendra donc trois onglets : Urgence, Me former et M'entraîner, sur lesquels l'utilisateur pourra trouver les informations qu'il recherche aussi simplement que possible.

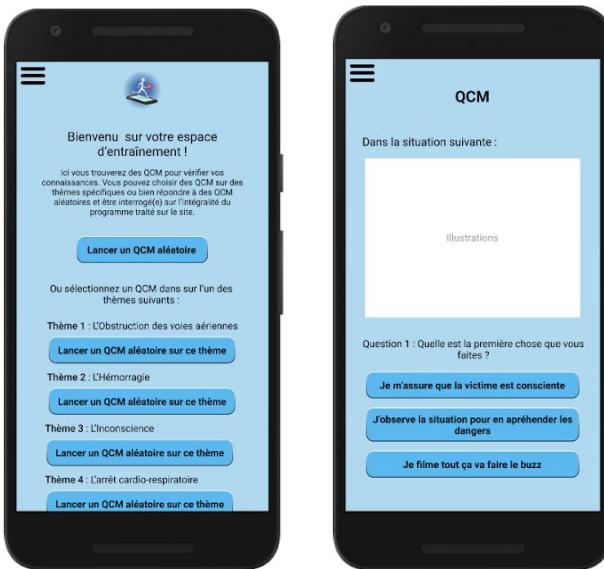
Sur l'onglet Urgence, l'utilisateur sera guidé par des questions pour trouver la fiche qui correspondra le mieux à sa situation.



L'onglet Me former regroupe des cours organisés par thèmes et chapitres.



Enfin, l'onglet M'entraîner regroupe des QCM sur les différents thèmes abordés.



Cette maquette sera bien sûr amenée à connaître des modifications mais elle nous donne une première idée concrète de ce qui sera à réaliser pour la suite du projet.

La maquette pseudo-cliquable est visualisable sur le lien suivant :

<https://www.figma.com/proto/OWatq6LR6jALp7SOQDMkjv/Secouriste-de-poche-v1?node-id=1%3A2&scaling=scale-down>

CONCLUSION

Pour conclure, Data Lake et Data Warehouse sont tous deux utilisés pour stocker des big datas. Mais ce seul point commun ne fait pas d'eux des systèmes interchangeables. En effet, leur architecture étant différente, nous avons exposé leurs différences afin de mieux comprendre pourquoi ils sont utilisés à des fins non similaires. En effet, nous avons bien vu qu'un Data Lake peut convenir à une entreprise là où un Data Warehouse ne lui conviendrait pas. Nos recherches se sont néanmoins plus centrées sur le secteur de la santé afin de potentiellement voir si Data Lake ou Data Warehouse pourraient nous servir dans notre projet tutoré. Le début des recherches sur notre projet intitulé "Secourisme de poche" a également été développé dans la partie IV. Ce développement ainsi que toutes nos recherches nous ont amené à penser que la création d'un Data Warehouse ou un Data Lake ne serait pas forcément utile à notre projet.

BIBLIOGRAPHIE

Amazon Web Service France. "Créer son Data Lake avec AWS - AWS Summit Paris 2018."

Créer son Data Lake avec AWS - AWS Summit Paris 2018, 2018.

https://www.youtube.com/watch?v=AU_ORwLjivY&feature=emb_logo. Accessed 2020.

Aws.Amazon.com. "Data warehouse system architecture." *Data warehouse system architecture*, ?.

https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html. Accessed 2020.

Aws.Amazon.com. "Déploiement d'un entrepôt de données." *Déploiement d'un entrepôt de données*, ?.

<https://aws.amazon.com/fr/getting-started/hands-on/deploy-data-warehouse/>. Accessed 2020.

axysweb.com. "ETL vs ELT : comment faire son choix entre ces 2 processus ?" *ETL vs ELT : comment faire son choix entre ces 2 processus* ?, ?, axysweb.com. Accessed 2020.

CNIL. "Traitements de données de santé : comment faire la distinction entre un entrepôt et une recherche et quelles conséquences ?" *Traitements de données de santé : comment faire la distinction entre un entrepôt et une recherche et quelles conséquences* ?, 2019.

<https://www.cnil.fr/fr/traitements-de-donnees-de-sante-comment-faire-la-distinction-entre-un-entrepot-et-une-recherche-et>. Accessed 2020.

Conseil National de l'ordre des médecins. "Protéger les données de santé." *Protéger les données de santé*, 2019.

<https://www.conseil-national.medecin.fr/medecin/devoirs-droits/proteger-donnees-sa>

nte#:~:text=Le%20r%C3%A8glement%20g%C3%A9n%C3%A9ral%20sur%20la,secr
et%20des%20informations%20la%20concernant. Accessed 2020.

Diyotta.com. “Data Warehouse: what it is, history and why it matters.” Diyotta.com, ?.
<https://www.diyotta.com/data-warehouse-definition-history-and-evolution>. Accessed 2020.

Guru99.com. “What is Data Warehouse? Types, Definition & Example.” What is Data
Warehouse? Types, Definition & Example, ?.
<https://www.guru99.com/data-warehousing.html#7>. Accessed 2020.

HOUDART & Associés. “LES ENTREPÔTS HOSPITALIERS DE DONNÉES, DU MYTHE À
LA RÉALITÉ.” LES ENTREPÔTS HOSPITALIERS DE DONNÉES, DU MYTHE À LA
RÉALITÉ, 2019.

<https://www.houdart.org/les-entrepos-hospitaliers-de-donnees-du-mythe-a-la-realite/>.
Accessed 2020.

Ibm.com. “IBM Db2 Warehouse on Cloud.” IBM Db2 Warehouse on Cloud, ?.
<https://www.ibm.com/fr-fr/cloud/db2-warehouse-on-cloud>. Accessed 2020.

Journaldunet.com. “Application dynamique : anatomie d'une technologie au service de la
compétitivité des entreprises.” Application dynamique : anatomie d'une technologie
au service de la compétitivité des entreprises, 2018.
<https://www.journaldunet.com/solutions/dsi/1209591-application-dynamique-anatomie-dune-technologie-au-service-de-la-competitivite-des-entreprises/>. Accessed 2020.

Lebigdata.com. Google BigQuery : tout savoir sur la plateforme Cloud de Big Data, Google
BigQuery : tout savoir sur la plateforme Cloud de Big Data, 2019.
<https://www.lebigdata.fr/google-bigquery-tout-savoir>. Accessed 2020.

Lebigdata.com. “OLAP : définition d'une technologie d'analyse multidimensionnelle.”
lebigdata.fr, 18 03 2018, <https://www.lebigdata.fr/olap-online-analytical-processing>.
Accessed - - 2020.

Microsoft.com. “Azure Synapse Analytics.” Azure Synapse Analytics, ?.
<https://docs.microsoft.com/fr-fr/azure/synapse-analytics/>. Accessed 2020.

Miloslavskaya, Natalia, and Alexander Tolstoy. *Big Data, Fast Data and Data Lake*

Concepts. ?, ?

Oracle.com. "Data Lake : Définition." *Data Lake : Définition*, ?,

<https://www.oracle.com/fr/database/data-lake-definition.html>. Accessed 2020.

Oracle.com. "Data Warehouse : Qu'est-ce que c'est ?" oracle.com, ?,

<https://www.oracle.com/fr/database/data-warehouse-definition.html>. Accessed 2020.

Oracle.com. "Présentation de la solution Autonomous Data Warehouse." Présentation de la solution Autonomous Data Warehouse, 2018.

<https://www.oracle.com/fr/database/data-warehouse.html>. Accessed 2020.

[Oracle.com](#). “Qu'est-ce que le cloud? | Définition du cloud | Cloudflare.” *Qu'est-ce que le cloud? | Définition du cloud | Cloudflare*, ?,

<https://www.oracle.com/fr/database/benefices-data-warehouse-cloud.html>. Accessed 2020.

Panoply.io. "A Deep Dive Into Google BigQuery Architecture." A Deep Dive Into Google BigQuery Architecture, ?,

<https://panoply.io/data-warehouse-guide/bigquery-architecture/>. Accessed 2020.

Talend.com. "Différences entre lac de données et entrepôt de données." Différences entre lac de données et entrepôt de données, ?.

<https://www.talend.com/fr/resources/data-lake-vs-data-warehouse/>. Accessed 2020.

365 Data Science. "What Is a Data Warehouse?" [Youtube.com](https://www.youtube.com/watch?v=KJyfjwvXWU), 2020.

https://www.youtube.com/watch?v=AHR_7jFCMeY&feature=youtu.be. Accessed 2020.

ANNEXES

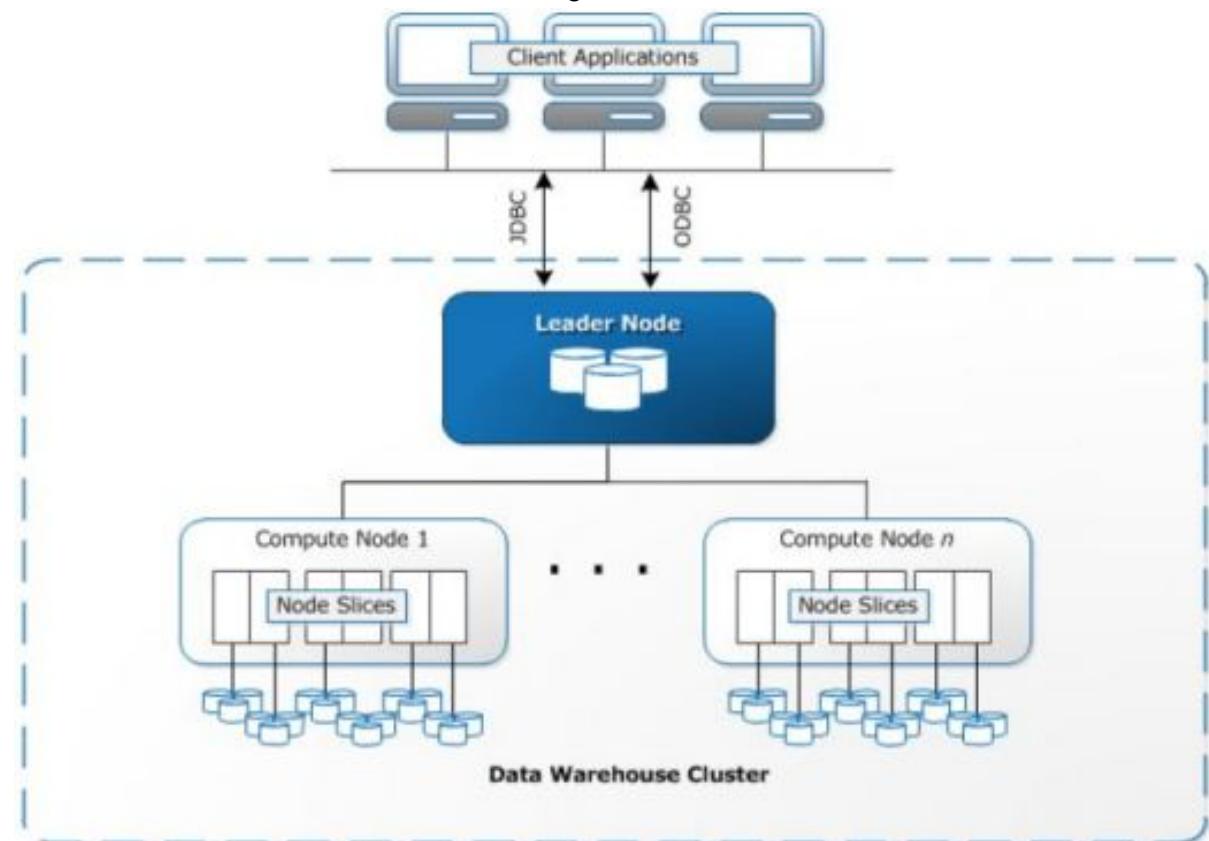
Annexe 1 : Amazon Redshift Manuel du développeur de base de données; Architecture système de l'entrepôt de données

Lorsque vous exécutez des requêtes analytiques, vous extrayez, comparez et évaluez de grandes quantités de données dans le cadre d'opérations à plusieurs étapes afin d'obtenir un résultat final.

Amazon Redshift permet un stockage efficace et des performances de requête optimales via la combinaison d'un traitement hautement parallèle, d'un stockage des données en colonnes et des schémas d'encodage de compression des données ciblés et très efficaces. Cette section offre une présentation de l'architecture système de Amazon Redshift.

Architecture système de l'entrepôt de données

Cette section présente les éléments de l'architecture système de l'entrepôt de données Amazon Redshift comme illustré dans la figure suivante.



Applications clientes

Amazon Redshift s'intègre à différents outils de chargement de données et outils

d'extraction, de transformation et de chargement (ETL), ainsi qu'à divers outils de création de rapports d'aide à la décision, d'exploration de données et d'analyse. Comme Amazon Redshift s'appuie sur PostgreSQL, standard de l'industrie, la plupart des applications clientes SQL existantes fonctionnent avec quelques changements minimes uniquement. Pour plus d'informations sur les différences importantes entre Amazon Redshift SQL et PostgreSQL, consultez Amazon Redshift et PostgreSQL (p. 415).

Connexions

Amazon Redshift communique avec les applications clientes à l'aide de pilotes JDBC et ODBC standards de l'industrie pour PostgreSQL. Pour plus d'informations, consultez Amazon Redshift et PostgreSQL JDBC et ODBC (p. 416).

Clusters

Le composant principal de l'infrastructure d'un entrepôt de données Amazon Redshift est un cluster. Un cluster est composé d'un ou plusieurs nœuds de calcul. Si un cluster est provisionné avec deux nœuds de calcul ou plus, un nœud principal supplémentaire coordonne les nœuds de calcul et gère la communication externe. Votre application cliente n'interagit directement qu'avec le nœud principal. Les nœuds de calcul sont transparents pour les applications externes.

Nœud principal

Le nœud principal gère les communications avec les programmes clients et toute la communication avec les nœuds de calcul. Il analyse et développe des plans d'exécution pour effectuer des opérations de base de données : en particulier, la série d'étapes nécessaires pour obtenir des résultats pour les requêtes complexes. D'après le plan d'exécution, le nœud principal compile le code, distribue le code compilé aux nœuds de calcul et attribue une partie des données à chaque nœud de calcul. Le nœud principal distribue les instructions SQL aux nœuds de calcul uniquement quand une requête fait référence aux tables stockées sur les nœuds de calcul. Toutes les autres requêtes s'exécutent exclusivement sur le nœud principal. Amazon Redshift est conçu pour mettre en place certaines fonctions SQL uniquement sur le nœud principal. Une requête qui utilise une de ces fonctions retourne une erreur si elle fait référence aux tables qui résident sur les nœuds de calcul. Pour plus d'informations, consultez Fonctions SQL prises en charge sur le nœud principal (p. 414).

Nœuds de calcul

Le nœud principal compile le code des éléments du plan d'exécution et affecte le code aux nœuds de calcul. Les nœuds de calcul exécutent le code compilé et renvoient les résultats intermédiaires au nœud principal pour l'agrégation finale.

Chaque nœud de calcul a ses propres UC, mémoire et stockage sur disque attaché dédiés ; ils sont déterminés par le type de nœud. Lorsque votre charge de travail augmente, vous pouvez augmenter la capacité de calcul et la capacité de stockage d'un cluster en augmentant le nombre de nœuds, en mettant à niveau le type de nœud, ou les deux.

Amazon Redshift fournit plusieurs types de nœuds pour vos besoins en calcul et en stockage. Pour plus de détails sur chaque type de nœud, veuillez consulter les clusters Amazon Redshift dans le Amazon Redshift Cluster Management Guide.

Tranches de nœud

Un nœud de calcul est divisé en tranches. Chaque tranche se voit attribuer une partie de la mémoire et de l'espace disque du nœud, où elle traite une partie de la charge de travail affectée au nœud. Le nœud principal gère la distribution des données aux tranches et attribue la charge de travail des requêtes ou autres opérations de base de données aux tranches. Les tranches travaillent alors en parallèle pour terminer l'opération.

Le nombre de tranches par nœud est déterminé par la taille de nœud du cluster. Pour de plus amples informations sur le nombre de tranches pour chaque taille de nœud, veuillez consulter À propos des clusters et des nœuds dans le Amazon Redshift Cluster Management Guide.

Lorsque vous créez une table, vous pouvez éventuellement spécifier une colonne comme clé de distribution. Lorsque la table est chargée avec les données, les lignes sont distribuées aux tranches des nœuds selon la clé de distribution définie pour une table. Le choix d'une bonne clé de distribution permet à Amazon Redshift d'utiliser le traitement parallèle pour charger les données et exécuter les requêtes efficacement. Pour plus d'informations sur le choix d'une clé de distribution, consultez Choose the best distribution style (p. 27).

Réseau interne

Amazon Redshift tire profit des connexions à bande passante élevée, de la proximité et des protocoles de communication personnalisés pour fournir une communication réseau privée à très haut débit entre le nœud principal et les nœuds de calcul. Les nœuds de calcul s'exécutent sur un réseau isolé distinct auquel les applications clientes n'accèdent jamais directement.

Bases de données :

Un cluster contient une ou plusieurs bases de données. Les données utilisateur sont stockées sur les nœuds de calcul. Votre client SQL communique avec le nœud principal, qui à son tour coordonne l'exécution des requêtes avec les nœuds de calcul.

Amazon Redshift est un système de gestion de base de données relationnelle (SGBDR), compatible de ce fait avec d'autres applications SGBDR. Même s'il fournit les mêmes fonctionnalités qu'un SGBDR classique, y compris les fonctions de traitement transactionnel en ligne (OLTP) comme l'insertion et la suppression de données, Amazon Redshift est optimisé pour l'analyse hautes performances et la création de rapports d'ensembles de données très volumineux. Amazon Redshift est basé sur PostgreSQL. Amazon Redshift et PostgreSQL présentent un certain nombre de différences très importantes que vous devez prendre en compte lorsque vous concevez et développez vos applications d'entre�ot de données. Pour plus d'informations sur les différences entre Amazon Redshift SQL et PostgreSQL, consultez Amazon Redshift et PostgreSQL (p. 415).

Annexe 2 : Dremel: Interactive Analysis of Web-Scale Datasets

Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer,
Shiva Shivakumar, Matt Tolton, Theo Vassilakis
Google, Inc.

{melnik, andrey, jlong, gromer, shiva, mtolton, theov}@google.com

ABSTRACT

Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for nested records and discuss experiments on few-thousand node instances of the system.

1. INTRODUCTION

Large-scale analytical data processing has become widespread in web companies and across industries, not least due to low-cost storage that enabled collecting vast amounts of business-critical data. Putting this data at the fingertips of analysts and engineers has grown increasingly important; interactive response times often make a qualitative difference in data exploration, monitoring, online customer support, rapid prototyping, debugging of data pipelines, and other tasks.

Performing interactive data analysis at scale demands a high degree of parallelism. For example, reading one terabyte of compressed data in one second using today's commodity disks would require tens of thousands of disks. Similarly, CPU-intensive queries may need to run on thousands of cores to complete within seconds. At Google, massively parallel computing is done using shared clusters of commodity machines [5]. A cluster typically hosts a multitude of distributed applications that share resources, have widely varying workloads, and run on machines with different hardware parameters. An individual worker in a distributed application may take much longer to execute a given task than others, or may never complete due to failures or preemption by the cluster management system. Hence, dealing with stragglers and failures is essential for achieving fast execution and fault tolerance [10].

The data used in web and scientific computing is often non-relational. Hence, a flexible data model is essential in these domains. Data structures used in programming languages, messages

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were presented at The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... \$ 10.00.

exchanged by distributed systems, structured documents, etc. lend themselves naturally to a *nested* representation. Normalizing and recombining such data at web scale is usually prohibitive. A nested data model underlies most of structured data processing at Google [21] and reportedly at other major web companies.

This paper describes a system called Dremel¹ that supports interactive analysis of very large datasets over shared clusters of commodity machines. Unlike traditional databases, it is capable of operating on *in situ* nested data. *In situ* refers to the ability to access data 'in place', e.g., in a distributed file system (like GFS [14]) or another storage layer (e.g., Bigtable [8]). Dremel can execute many queries over such data that would ordinarily require a sequence of MapReduce (MR [12]) jobs, but at a fraction of the execution time. Dremel is not intended as a replacement for MR and is often used in conjunction with it to analyze outputs of MR pipelines or rapidly prototype larger computations.

Dremel has been in production since 2006 and has thousands of users within Google. Multiple instances of Dremel are deployed in the company, ranging from tens to thousands of nodes. Examples of using the system include:

- Analysis of crawled web documents.
- Tracking install data for applications on Android Market.
- Crash reporting for Google products.
- OCR results from Google Books.
- Spam analysis.
- Debugging of map tiles on Google Maps.
- Tablet migrations in managed Bigtable instances.
- Results of tests run on Google's distributed build system.
- Disk I/O statistics for hundreds of thousands of disks.
- Resource monitoring for jobs run in Google's data centers.
- Symbols and dependencies in Google's codebase.

Dremel builds on ideas from web search and parallel DBMSs. First, its architecture borrows the concept of a serving tree used in distributed search engines [11]. Just like a web search request, a query gets pushed down the tree and is rewritten at each step. The result of the query is assembled by aggregating the replies received from lower levels of the tree. Second, Dremel provides a high-level, SQL-like language to express ad hoc queries. In contrast to layers such as Pig [18] and Hive [16], it executes queries natively without translating them into MR jobs.

Lastly, and importantly, Dremel uses a column-striped storage representation, which enables it to read less data from secondary

¹Dremel is a brand of power tools that primarily rely on their speed as opposed to torque. We use this name for an internal project only.

storage and reduce CPU cost due to cheaper compression. Column stores have been adopted for analyzing relational data [1] but to the best of our knowledge have not been extended to nested data models. The columnar storage format that we present is supported by many data processing tools at Google, including MR, Sawzall [20], and FlumeJava [7].

In this paper we make the following contributions:

- We describe a novel columnar storage format for nested data. We present algorithms for dissecting nested records into columns and reassembling them (Section 4).
- We outline Dremel’s query language and execution. Both are designed to operate efficiently on column-striped nested data and do not require restructuring of nested records (Section 5).
- We show how execution trees used in web search systems can be applied to database processing, and explain their benefits for answering aggregation queries efficiently (Section 6).
- We present experiments on trillion-record, multi-terabyte datasets, conducted on system instances running on 1000-4000 nodes (Section 7).

This paper is structured as follows. In Section 2, we explain how Dremel is used for data analysis in combination with other data management tools. Its data model is presented in Section 3. The main contributions listed above are covered in Sections 4-8. Related work is discussed in Section 9. Section 10 is the conclusion.

2. BACKGROUND

We start by walking through a scenario that illustrates how interactive query processing fits into a broader data management ecosystem. Suppose that Alice, an engineer at Google, comes up with a novel idea for extracting new kinds of signals from web pages. She runs an MR job that cranks through the input data and produces a dataset containing the new signals, stored in billions of records in the distributed file system. To analyze the results of her experiment, she launches Dremel and executes several interactive commands:

```
DEFINE TABLE t AS /path/to/data/*
SELECT TOP(signal1, 100), COUNT(*) FROM t
```

Her commands execute in seconds. She runs a few other queries to convince herself that her algorithm works. She finds an irregularity in signal1 and digs deeper by writing a FlumeJava [7] program that performs a more complex analytical computation over her output dataset. Once the issue is fixed, she sets up a pipeline which processes the incoming input data continuously. She formulates a few canned SQL queries that aggregate the results of her pipeline across various dimensions, and adds them to an interactive dashboard. Finally, she registers her new dataset in a catalog so other engineers can locate and query it quickly.

The above scenario requires interoperation between the query processor and other data management tools. The first ingredient for that is a *common storage layer*. The Google File System (GFS [14]) is one such distributed storage layer widely used in the company. GFS uses replication to preserve the data despite faulty hardware and achieve fast response times in presence of stragglers. A high-performance storage layer is critical for *in situ* data management. It allows accessing the data without a time-consuming loading phase, which is a major impedance to database usage in analytical data processing [13], where it is often possible to run dozens of MR analyses before a DBMS is able to load the data and execute a single query. As an added benefit, data in a file system can be conveniently manipulated using standard tools, e.g., to transfer to another cluster, change access privileges, or identify a subset of data for analysis based on file names.

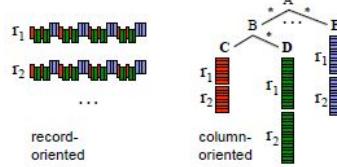


Figure 1: Record-wise vs. columnar representation of nested data

The second ingredient for building interoperable data management components is a *shared storage format*. Columnar storage proved successful for flat relational data but making it work for Google required adapting it to a nested data model. Figure 1 illustrates the main idea: all values of a nested field such as A.B.C are stored contiguously. Hence, A.B.C can be retrieved without reading A.E, A.B.D, etc. The challenge that we address is how to preserve all structural information and be able to reconstruct records from an arbitrary subset of fields. Next we discuss our data model, and then turn to algorithms and query processing.

3. DATA MODEL

In this section we present Dremel’s data model and introduce some terminology used later. The data model originated in the context of distributed systems (which explains its name, ‘Protocol Buffers’ [21]), is used widely at Google, and is available as an open source implementation. The data model is based on strongly-typed nested records. Its abstract syntax is given by:

$$\tau = \text{dom} \mid \langle A_1 : \tau^{[*|?]}, \dots, A_n : \tau^{[*|?]} \rangle$$

where τ is an atomic type or a record type. Atomic types in **dom** comprise integers, floating-point numbers, strings, etc. Records consist of one or multiple fields. Field i in a record has a name A_i and an optional multiplicity label. *Repeated* fields ($*$) may occur multiple times in a record. They are interpreted as lists of values, i.e., the order of field occurrences in a record is significant. *Optional* fields ($?$) may be missing from the record. Otherwise, a field is *required*, i.e., must appear exactly once.

To illustrate, consider Figure 2. It depicts a schema that defines a record type **Document**, representing a web document. The schema definition uses the concrete syntax from [21]. A Document has a required integer **DocId** and optional **Links**, containing a list of Forward and Backward entries holding **DocId**s of other web pages. A document can have multiple **Names**, which are different URLs by which the document can be referenced. A Name contains a sequence of **Code** and (optional) **Country** pairs. Figure 2 also shows two sample records, r_1 and r_2 , conforming to the schema. The record structure is outlined using indentation. We will use these sample records to explain the algorithms in the next sections. The fields defined in the schema form a tree hierarchy. The full *path* of a nested field is defined using the usual dotted notation, e.g., **Name.Language.Code**.

The nested data model backs a platform-neutral, extensible mechanism for serializing structured data at Google. Code generation tools produce bindings for programming languages such as C++ or Java. Cross-language interoperability is achieved using a standard binary on-the-wire representation of records, in which field values are laid out sequentially as they occur in the record. This way, a MR program written in Java can consume records from a data source exposed via a C++ library. Thus, if records are stored in a columnar representation, assembling them fast is important for interoperation with MR and other data processing tools.

DocId: 10	r ₁
Links	
Forward: 20	
Forward: 40	
Forward: 60	
Name	
Language	
Code: 'en-us'	
Country: 'us'	
Language	
Code: 'en'	
Url: 'http://A'	
Name	
Url: 'http://B'	
Name	
Language	
Code: 'en-gb'	
Country: 'gb'	

message Document {	
required int64 DocId;	
optional group Links {	
repeated int64 Backward;	
repeated int64 Forward; }	
repeated group Name {	
repeated group Language {	
required string Code;	
optional string Country; }	
optional string Url; }	

DocId: 20	r ₂
Links	
Backward: 10	
Backward: 30	
Forward: 80	
Name	
Url: 'http://C'	

Figure 2: Two sample nested records and their schema

4. NESTED COLUMNAR STORAGE

As illustrated in Figure 1, our goal is to store all values of a given field consecutively to improve retrieval efficiency. In this section, we address the following challenges: lossless representation of record structure in a columnar format (Section 4.1), fast encoding of record assembly (Section 4.2), and efficient record assembly (Section 4.3).

4.1 Repetition and Definition Levels

Values alone do not convey the structure of a record. Given two values of a repeated field, we do not know at what ‘level’ the value repeated (e.g., whether these values are from two different records, or two repeated values in the same record). Likewise, given a missing optional field, we do not know which enclosing records were defined explicitly. We therefore introduce the concepts of repetition and definition levels, which are defined below. For reference, see Figure 3 which summarizes the repetition and definition levels for all atomic fields in our sample records.

Repetition levels. Consider field `Code` in Figure 2. It occurs three times in r_1 . Occurrences ‘en-us’ and ‘en’ are inside the first `Name`, while ‘en-gb’ is in the third `Name`. To disambiguate these occurrences, we attach a repetition level to each value. It tells us *at what repeated field in the field’s path the value has repeated*. The field path `Name.Language.Code` contains two repeated fields, `Name` and `Language`. Hence, the repetition level of `Code` ranges between 0 and 2; level 0 denotes the start of a new record. Now suppose we are scanning record r_1 top down. When we encounter ‘en-us’, we have not seen any repeated fields, i.e., the repetition level is 0. When we see ‘en’, field `Language` has repeated, so the repetition level is 2. Finally, when we encounter ‘en-gb’, `Name` has repeated most recently (`Language` occurred only once after `Name`), so the repetition level is 1. Thus, the repetition levels of `Code` values in r_1 are 0, 2, 1.

Notice that the second `Name` in r_1 does not contain any `Code` values. To determine that ‘en-gb’ occurs in the third `Name` and not in the second, we add a NULL value between ‘en’ and ‘en-gb’ (see Figure 3). `Code` is a required field in `Language`, so the fact that it is missing implies that `Language` is not defined. In general though, determining the level up to which nested records exist requires extra information.

Definition levels. Each value of a field with path p , esp. every NULL, has a definition level specifying *how many fields in p that could be undefined (because they are optional or repeated)* are ac-

DocId	
10	0 0
20	0 0

Name.Url	
http://A	0 2
http://B	1 2
NULL	1 1
http://C	0 2

Links.Forward	
20	0 2
40	1 2
60	1 2
80	0 2

Links.Backward	
NULL	0 1
10	0 2
30	1 2

Name.Language.Code	
en-us	0 2
en	2 2
NULL	1 1
en-gb	1 2
NULL	0 1

Name.Language.Country	
us	0 3
NULL	2 2
NULL	1 1
gb	1 3
NULL	0 1

Figure 3: Column-striped representation of the sample data in Figure 2, showing repetition levels (r) and definition levels (d)

tually present in the record. To illustrate, observe that r_1 has no Backward links. However, field `Links` is defined (at level 1). To preserve this information, we add a NULL value with definition level 1 to the `Links.Backward` column. Similarly, the missing occurrence of `Name.Language.Country` in r_2 carries a definition level 1, while its missing occurrences in r_1 have definition levels 2 (inside `Name.Language`) and 1 (inside `Name`), respectively.

We use integer definition levels as opposed to is-null bits so that the data for a leaf field (e.g., `Name.Language.Country`) contains the information about the occurrences of its parent fields; an example of how this information is used is given in Section 4.3.

The encoding outlined above preserves the record structure losslessly. We omit the proof for space reasons.

Encoding. Each column is stored as a set of blocks. Each block contains the repetition and definition levels (henceforth, simply called levels) and compressed field values. NULLs are not stored explicitly as they are determined by the definition levels: any definition level smaller than the number of repeated and optional fields in a field’s path denotes a NULL. Definition levels are not stored for values that are always defined. Similarly, repetition levels are stored only if required; for example, definition level 0 implies repetition level 0, so the latter can be omitted. In fact, in Figure 3, no levels are stored for `DocId`. Levels are packed as bit sequences. We only use as many bits as necessary; for example, if the maximum definition level is 3, we use 2 bits per definition level.

4.2 Splitting Records into Columns

Above we presented an encoding of the record structure in a columnar format. The next challenge we address is how to produce column stripes with repetition and definition levels efficiently.

The base algorithm for computing repetition and definition levels is given in Appendix A. The algorithm recurses into the record structure and computes the levels for each field value. As illustrated earlier, repetition and definition levels may need to be computed even if field values are missing. Many datasets used at Google are sparse; it is not uncommon to have a schema with thousands of fields, only a hundred of which are used in a given record. Hence, we try to process missing fields as cheaply as possible. To produce column stripes, we create a tree of *field writers*, whose structure matches the field hierarchy in the schema. The basic idea is to update field writers only when they have their own data, and not try to propagate parent state down the tree unless absolutely neces-

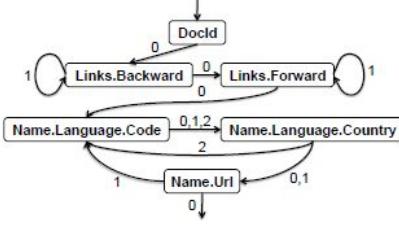


Figure 4: Complete record assembly automaton. Edges are labeled with repetition levels.

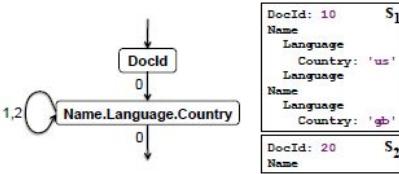


Figure 5: Automaton for assembling records from two fields, and the records it produces

sary. To do that, child writers inherit the levels from their parents. A child writer synchronizes to its parent's levels whenever a new value is added.

4.3 Record Assembly

Assembling records from columnar data efficiently is critical for record-oriented data processing tools (e.g., MR). Given a subset of fields, our goal is to reconstruct the original records as if they contained just the selected fields, with all other fields stripped away. The key idea is this: we create a finite state machine (FSM) that reads the field values and levels for each field, and appends the values sequentially to the output records. An FSM state corresponds to a field reader for each selected field. State transitions are labeled with repetition levels. Once a reader fetches a value, we look at the next repetition level to decide what next reader to use. The FSM is traversed from the start to end state once for each record.

Figure 4 shows an FSM that reconstructs the complete records in our running example. The start state is DocId. Once a DocId value is read, the FSM transitions to Links.Backward. After all repeated Backward values have been drained, the FSM jumps to Links.Forward, etc. The details of the record assembly algorithm are in Appendix B.

To sketch how FSM transitions are constructed, let l be the next repetition level returned by the current field reader for field f . Starting at f in the schema tree, we find its ancestor that repeats at level l and select the first leaf field n inside that ancestor. This gives us an FSM transition $(f, l) \rightarrow n$. For example, let $l = 1$ be the next repetition level read by $f = \text{Name.Language.Country}$. Its ancestor with repetition level 1 is Name, whose first leaf field is $n = \text{Name.Url}$. The details of the FSM construction algorithm are in Appendix C.

If only a subset of fields need to be retrieved, we construct a simpler FSM that is cheaper to execute. Figure 5 depicts an FSM for reading the fields DocId and Name.Language.Country. The figure shows the output records s_1 and s_2 produced by the automaton. Notice that our encoding and the assembly algorithm

```

SELECT DocId AS Id,
       COUNT(Name.Language.Code) WITHIN Name AS Cnt,
       Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;

```

Id: 10	t ₁
Name	
Cnt: 2	
Language	
Str: 'http://A,en-us'	
Str: 'http://A,en'	
Name	
Cnt: 0	

Figure 6: Sample query, its result, and output schema

preserve the enclosing structure of the field Country. This is important for applications that need to access, e.g., the Country appearing in the first Language of the second Name. In XPath, this would correspond to the ability to evaluate expressions like $/Name[2]/Language[1]/Country$.

5. QUERY LANGUAGE

Dremel's query language is based on SQL and is designed to be efficiently implementable on columnar nested storage. Defining the language formally is out of scope of this paper; instead, we illustrate its flavor. Each SQL statement (and algebraic operators it translates to) takes as input one or multiple nested tables and their schemas and produces a nested table and its output schema. Figure 6 depicts a sample query that performs projection, selection, and within-record aggregation. The query is evaluated over the table $t = \{r_1, r_2\}$ from Figure 2. The fields are referenced using path expressions. The query produces a nested result although no record constructors are present in the query.

To explain what the query does, consider the selection operation (the WHERE clause). Think of a nested record as a labeled tree, where each label corresponds to a field name. The selection operator prunes away the branches of the tree that do not satisfy the specified conditions. Thus, only those nested records are retained where Name.Url is defined and starts with http. Next, consider projection. Each scalar expression in the SELECT clause emits a value at the same level of nesting as the most-repeated input field used in that expression. So, the string concatenation expression emits Str values at the level of Name.Language.Code in the input schema. The COUNT expression illustrates within-record aggregation. The aggregation is done WITHIN each Name subrecord, and emits the number of occurrences of Name.Language.Code for each Name as a non-negative 64-bit integer (uint64).

The language supports nested subqueries, inter and intra-record aggregation, top-k, joins, user-defined functions, etc; some of these features are exemplified in the experimental section.

6. QUERY EXECUTION

We discuss the core ideas in the context of a read-only system, for simplicity. Many Dremel queries are one-pass aggregations; therefore, we focus on explaining those and use them for experiments in the next section. We defer the discussion of joins, indexing, updates, etc, to future work.

Tree architecture. Dremel uses a multi-level serving tree to execute queries (see Figure 7). A root server receives incoming queries, reads metadata from the tables, and routes the queries to the next level in the serving tree. The leaf servers communicate

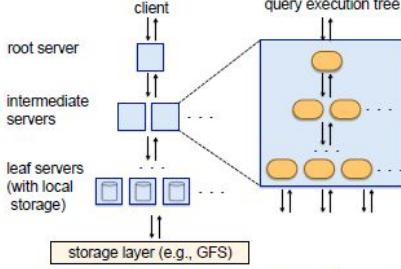


Figure 7: System architecture and execution inside a server node

with the storage layer or access the data on local disk. Consider a simple aggregation query below:

`SELECT A, COUNT(B) FROM T GROUP BY A`

When the root server receives the above query, it determines all *tablets*, i.e., horizontal partitions of the table, that comprise T and rewrites the query as follows:

`SELECT A, SUM(c) FROM ($R_1^1 \cup \dots \cup R_n^1$) GROUP BY A`

Tables R_1^1, \dots, R_n^1 are the results of queries sent to the nodes $1, \dots, n$ at level 1 of the serving tree:

$R_i^1 = \text{SELECT } A, \text{COUNT}(B) \text{ AS } c \text{ FROM } T_i^1 \text{ GROUP BY } A$

T_i^1 is a disjoint partition of tablets in T processed by server i at level 1. Each serving level performs a similar rewriting. Ultimately, the queries reach the leaves, which scan the tablets in T in parallel. On the way up, intermediate servers perform a parallel aggregation of partial results. The execution model presented above is well-suited for aggregation queries returning small and medium-sized results, which are a very common class of interactive queries. Large aggregations and other classes of queries may need to rely on execution mechanisms known from parallel DBMSs and MR.

Query dispatcher. Dremel is a multi-user system, i.e., usually several queries are executed simultaneously. A query dispatcher schedules queries based on their priorities and balances the load. Its other important role is to provide *fault tolerance* when one server becomes much slower than others or a tablet replica becomes unreachable.

The amount of data processed in each query is often larger than the number of processing units available for execution, which we call *slots*. A slot corresponds to an execution thread on a leaf server. For example, a system of 3,000 leaf servers each using 8 threads has 24,000 slots. So, a table spanning 100,000 tablets can be processed by assigning about 5 tablets to each slot. During query execution, the query dispatcher computes a histogram of tablet processing times. If a tablet takes a disproportionately long time to process, it reschedules it on another server. Some tablets may need to be redispersed multiple times.

The leaf servers read stripes of nested data in columnar representation. The blocks in each stripe are prefetched asynchronously; the read-ahead cache typically achieves hit rates of 95%. Tablets are usually three-way replicated. When a leaf server cannot access one tablet replica, it falls over to another replica.

The query dispatcher honors a parameter that specifies the minimum percentage of tablets that must be scanned before returning a result. As we demonstrate shortly, setting such parameter to a lower value (e.g., 98% instead of 100%) can often speed up execu-

Table name	Number of records	Size (unreplicated, compressed)	Number of fields	Data center	Rep. factor
T1	85 billion	87 TB	270	A	3x
T2	24 billion	13 TB	530	A	3x
T3	4 billion	70 TB	1200	A	3x
T4	1+ trillion	105 TB	50	B	3x
T5	1+ trillion	20 TB	30	B	2x

Figure 8: Datasets used in the experimental study

tion significantly, especially when using smaller replication factors.

Each server has an internal execution tree, as depicted on the right-hand side of Figure 7. The internal tree corresponds to a physical query execution plan, including evaluation of scalar expressions. Optimized, type-specific code is generated for most scalar functions. An execution plan for project-select-aggregate queries consists of a set of iterators that scan input columns in lockstep and emit results of aggregates and scalar functions annotated with the correct repetition and definition levels, bypassing record assembly entirely during query execution. For details, see Appendix D.

Some Dremel queries, such as top-k and count-distinct, return approximate results using known one-pass algorithms (e.g., [4]).

7. EXPERIMENTS

In this section we evaluate Dremel’s performance on several datasets used at Google, and examine the effectiveness of columnar storage for nested data. The properties of the datasets used in our study are summarized in Figure 8. In uncompressed, non-replicated form the datasets occupy about a petabyte of space. All tables are three-way replicated, except one two-way replicated table, and contain from 100K to 800K tablets of varying sizes. We start by examining the basic data access characteristics on a single machine, then show how columnar storage benefits MR execution, and finally focus on Dremel’s performance. The experiments were conducted on system instances running in two data centers next to many other applications, during regular business operation. Unless specified otherwise, execution times were averaged across five runs. Table and field names used below are anonymized.

Local disk. In the first experiment, we examine performance tradeoffs of columnar vs. record-oriented storage, scanning a 1GB fragment of table T_1 containing about 300K rows (see Figure 9). The data is stored on a local disk and takes about 375MB in compressed columnar representation. The record-oriented format uses heavier compression yet yields about the same size on disk. The experiment was done on a dual-core Intel machine with a disk providing 70MB/s read bandwidth. All reported times are cold; OS cache was flushed prior to each scan.

The figure shows five graphs, illustrating the time it takes to read and uncompress the data, and assemble and parse the records, for a subset of the fields. Graphs (a)-(c) outline the results for columnar storage. Each data point in these graphs was obtained by averaging the measurements over 30 runs, in each of which a set of columns of a given cardinality was chosen at random. Graph (a) shows reading and decompression time. Graph (b) adds the time needed to assemble nested records from columns. Graph (c) shows how long it takes to parse the records into strongly typed C++ data structures.

Graphs (d)-(e) depict the time for accessing the data on record-oriented storage. Graph (d) shows reading and decompression time. A bulk of the time is spent in decompression; in fact, the compressed data can be read from the disk in about half the time. As

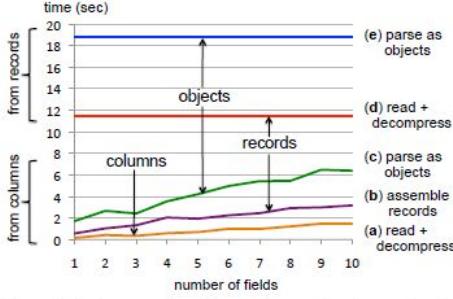


Figure 9: Performance breakdown when reading from a local disk (300K-record fragment of Table T_1)

Graph (e) indicates, parsing adds another 50% on top of reading and decompression time. These costs are paid for all fields, including the ones that are not needed.

The main takeaways of this experiment are the following: when few columns are read, the gains of columnar representation are of about an order of magnitude. Retrieval time for columnar nested data grows linearly with the number of fields. Record assembly and parsing are expensive, each potentially doubling the execution time. We observed similar trends on other datasets. A natural question to ask is where the top and bottom graphs cross, i.e., record-wise storage starts outperforming columnar storage. In our experience, the crossover point often lies at dozens of fields but it varies across datasets and depends on whether or not record assembly is required.

MR and Dremel. Next we illustrate a MR and Dremel execution on columnar vs. record-oriented data. We consider a case where a single field is accessed, i.e., the performance gains are most pronounced. Execution times for multiple columns can be extrapolated using the results of Figure 9. In this experiment, we count the average number of terms in a field `txtField` of table T_1 . MR execution is done using the following Sawzall [20] program:

```

numRecs: table sum of int;
numWords: table sum of int;
emit numRecs <- 1;
emit numWords <- CountWords(input.txtField);

```

The number of records is stored in the variable `numRecs`. For each record, `numWords` is incremented by the number of terms in `input.txtField` returned by the `CountWords` function. After the program runs, the average term frequency can be computed as `numWords/numRecs`. In SQL, this computation is expressed as:

`Q1: SELECT SUM(CountWords(txtField)) / COUNT(*) FROM T1`

Figure 10 shows the execution times of two MR jobs and Dremel on a logarithmic scale. Both MR jobs are run on 3000 workers. Similarly, a 3000-node Dremel instance is used to execute Query Q_1 . Dremel and MR-on-columns read about 0.5TB of compressed columnar data vs. 87TB read by MR-on-records. As the figure illustrates, MR gains an order of magnitude in efficiency by switching from record-oriented to columnar storage (from hours to minutes). Another order of magnitude is achieved by using Dremel (going from minutes to seconds).

Serving tree topology. In the next experiment, we show the impact of the serving tree depth on query execution times. We consider two GROUP BY queries on Table T_2 , each executed using

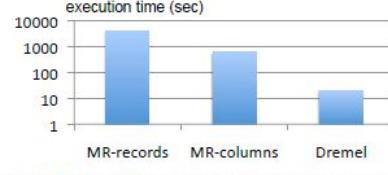


Figure 10: MR and Dremel execution on columnar vs. record-oriented storage (3000 nodes, 85 billion records)

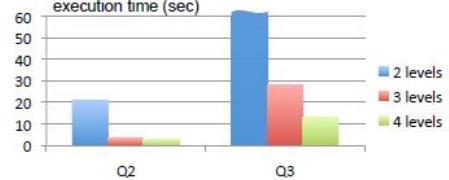


Figure 11: Execution time as a function of serving tree levels for two aggregation queries on T_2

a single scan over the data. Table T_2 contains 24 billion nested records. Each record has a repeated field `item` containing a numeric amount. The field `item.amount` repeats about 40 billion times in the dataset. The first query sums up the item amount by country:

`Q2: SELECT country, SUM(item.amount) FROM T2 GROUP BY country`

It returns a few hundred records and reads roughly 60GB of compressed data from disk. The second query performs a GROUP BY on a text field domain with a selection condition. It reads about 180GB and produces around 1.1 million distinct domains:

`Q3: SELECT domain, SUM(item.amount) FROM T2 WHERE domain CONTAINS '.net' GROUP BY domain`

Figure 11 shows the execution times for each query as a function of the server topology. In each topology, the number of leaf servers is kept constant at 2900 so that we can assume the same cumulative scan speed. In the 2-level topology (1:2900), a single root server communicates directly with the leaf servers. For 3 levels, we use a 1:100:2900 setup, i.e., an extra level of 100 intermediate servers. The 4-level topology is 1:10:100:2900.

Query Q_2 runs in 3 seconds when 3 levels are used in the serving tree and does not benefit much from an extra level. In contrast, the execution time of Q_3 is halved due to increased parallelism. At 2 levels, Q_3 is off the chart, as the root server needs to aggregate near-sequentially the results received from thousands of nodes. This experiment illustrates how aggregations returning many groups benefit from multi-level serving trees.

Per-tablet histograms. To drill deeper into what happens during query execution consider Figure 12. The figure shows how fast tablets get processed by the leaf servers for a specific run of Q_2 and Q_3 . The time is measured starting at the point when a tablet got scheduled for execution in an available slot, i.e., excludes the time spent waiting in the job queue. This measurement methodology factors out the effects of other queries that are executing simultaneously. The area under each histogram corresponds to 100%. As the figure indicates, 99% of Q_2 (Q_3) tablets are processed under one second (or two seconds).

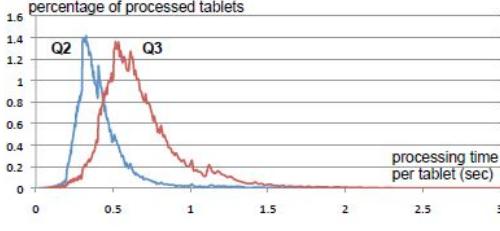


Figure 12: Histograms of processing times

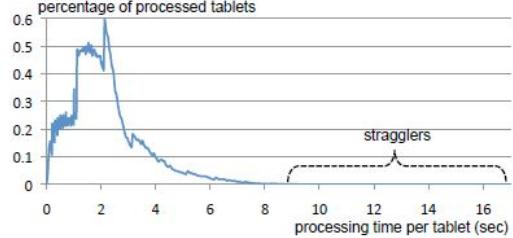


Figure 14: Query Q_5 on T_5 illustrating stragglers at 2 \times replication

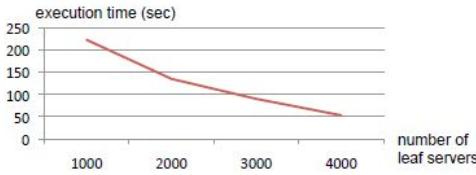


Figure 13: Scaling the system from 1000 to 4000 nodes using a top-k query Q_5 on a trillion-row table T_4

Within-record aggregation. As another experiment, we examine the performance of Query Q_4 run on Table T_3 . The query illustrates within-record aggregation: it counts all records where the sum of a.b.c.d values occurring in the record are larger than the sum of a.b.p.q.r values. The fields repeat at different levels of nesting. Due to column striping only 13GB (out of 70TB) are read from disk and the query completes in 15 seconds. Without support for nesting, running this query on T_3 would be grossly expensive.

```
Q4 : SELECT COUNT(c1 > c2) FROM
      (SELECT SUM(a.b.c.d) WITHIN RECORD AS c1,
           SUM(a.b.p.q.r) WITHIN RECORD AS c2
      FROM T3)
```

Scalability. The following experiment illustrates the scalability of the system on a trillion-record table. Query Q_5 shown below selects top-20 aid's and their number of occurrences in Table T_4 . The query scans 4.2TB of compressed data.

```
Q5: SELECT TOP(aid, 20), COUNT(*) FROM T4
     WHERE bid = {value1} AND cid = {value2}
```

The query was executed using four configurations of the system, ranging from 1000 to 4000 nodes. The execution times are in Figure 13. In each run, the total expended CPU time is nearly identical, at about 300K seconds, whereas the user-perceived time decreases near-linearly with the growing size of the system. This result suggests that a larger system can be just as effective in terms of resource usage as a smaller one, yet allows faster execution.

Stragglers. Our last experiment shows the impact of stragglers. Query Q_6 below is run on a trillion-row table T_5 . In contrast to the other datasets, T_5 is two-way replicated. Hence, the likelihood of stragglers slowing the execution is higher since there are fewer opportunities to reschedule the work.

```
Q6: SELECT COUNT(DISTINCT a) FROM T5
Query  $Q_6$  reads over 1TB of compressed data. The compres-
```

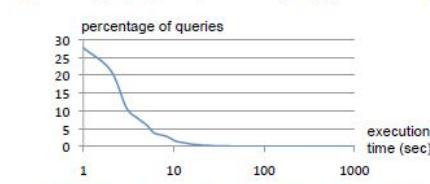


Figure 15: Query response time distribution in a monthly workload

sion ratio for the retrieved field is about 10. As indicated in Figure 14, the processing time for 99% of the tablets is below 5 seconds per tablet per slot. However, a small fraction of the tablets take a lot longer, slowing down the query response time from less than a minute to several minutes, when executed on a 2500 node system. The next section summarizes our experimental findings and the lessons we learned.

8. OBSERVATIONS

Dremel scans quadrillions of records per month. Figure 15 shows the query response time distribution in a typical monthly workload of one Dremel system, on a logarithmic scale. As the figure indicates, most queries are processed under 10 seconds, well within the interactive range. Some queries achieve a scan throughput close to 100 billion records per second on a shared cluster, and even higher on dedicated machines. The experimental data presented above suggests the following observations:

- Scan-based queries can be executed at interactive speeds on disk-resident datasets of up to a trillion records.
- Near-linear scalability in the number of columns and servers is achievable for systems containing thousands of nodes.
- MR can benefit from columnar storage just like a DBMS.
- Record assembly and parsing are expensive. Software layers (beyond the query processing layer) need to be optimized to directly consume column-oriented data.
- MR and query processing can be used in a complementary fashion; one layer's output can feed another's input.
- In a multi-user environment, a larger system can benefit from economies of scale while offering a qualitatively better user experience.
- If trading speed against accuracy is acceptable, a query can be terminated much earlier and yet see most of the data.
- The bulk of a web-scale dataset can be scanned fast. Getting to the last few percent within tight time bounds is hard.

Dremel's query language builds on the ideas from [9], which introduced a language that avoids restructuring when accessing nested data. In contrast, restructuring is usually required in XQuery and object-oriented query languages, e.g., using nested for-loops and constructors. We are not aware of practical implementations of [9]. A recent SQL-like language operating on nested data is Pig [18]. Other systems for parallel data processing include Scope [6] and DryadLINQ [23], and are discussed in more detail in [7].

10. CONCLUSIONS

We presented Dremel, a distributed system for interactive analysis of large datasets. Dremel is a custom, scalable data management solution built from simpler components. It complements the MR paradigm. We discussed its performance on trillion-record, multi-terabyte datasets of real data. We outlined the key aspects of Dremel, including its storage format, query language, and execution. In the future, we plan to cover in more depth such areas as formal algebraic specification, joins, extensibility mechanisms, etc.

11. ACKNOWLEDGEMENTS

Dremel has benefited greatly from the input of many engineers and interns at Google, in particular Craig Chambers, Ori Gershoni, Rajeev Byrissetti, Leon Wong, Erik Hendriks, Erika Rice Scherpelz, Charlie Garrett, Idan Avraham, Rajesh Rao, Andy Kreling, Li Yin, Madhusudan Hosagrahara, Dan Belov, Brian Bershad, Lawrence You, Rongrong Zhong, Meelap Shah, and Nathan Bales.

selective retrieval of columns.

The data model used in Dremel is a variation of the complex value models and nested relational models discussed in [2]. Dremel's query language builds on the ideas from [9], which introduced a language that avoids restructuring when accessing nested data. In contrast, restructuring is usually required in XQuery and object-oriented query languages, e.g., using nested for-loops and constructors. We are not aware of practical implementations of [9]. A recent SQL-like language operating on nested data is Pig [18]. Other systems for parallel data processing include Scope [6] and DryadLINQ [23], and are discussed in more detail in [7].

10. CONCLUSIONS

We presented Dremel, a distributed system for interactive analysis of large datasets. Dremel is a custom, scalable data management solution built from simpler components. It complements the MR paradigm. We discussed its performance on trillion-record, multi-terabyte datasets of real data. We outlined the key aspects of Dremel, including its storage format, query language, and execution. In the future, we plan to cover in more depth such areas as formal algebraic specification, joins, extensibility mechanisms, etc.

11. ACKNOWLEDGEMENTS

Dremel has benefited greatly from the input of many engineers and interns at Google, in particular Craig Chambers, Ori Gershoni, Rajeev Byrissetti, Leon Wong, Erik Hendriks, Erika Rice Scherpelz, Charlie Garrett, Idan Avraham, Rajesh Rao, Andy Kreling, Li Yin, Madhusudan Hosagrahara, Dan Belov, Brian Bershad, Lawrence You, Rongrong Zhong, Meelap Shah, and Nathan Bales.

- [11] J. Dean. Challenges in Building Large-Scale Information Retrieval Systems: Invited Talk. In *WSDM*, 2009.
- [12] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.
- [13] J. Dean and S. Ghemawat. MapReduce: a Flexible Data Processing Tool. *Commun. ACM*, 53(1), 2010.
- [14] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *SOSP*, 2003.
- [15] Hadoop Apache Project. <http://hadoop.apache.org>.
- [16] Hive. <http://wiki.apache.org/hadoop/Hive>, 2009.
- [17] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. In *SIGMOD*, 2000.
- [18] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: a Not-so-Foreign Language for Data Processing. In *SIGMOD*, 2008.
- [19] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *SIGMOD*, 2004.
- [20] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming*, 13(4), 2005.
- [21] Protocol Buffers: Developer Guide. Available at <http://code.google.com/apis/protocolbuffers/docs/overview.html>.
- [22] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and Parallel DBMSs: Friends or Foes? *Commun. ACM*, 53(1), 2010.
- [23] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *OSDI*, 2008.

Google Blog, Nov. 2008. At <http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html>.

- [11] J. Dean. Challenges in Building Large-Scale Information Retrieval Systems: Invited Talk. In *WSDM*, 2009.
- [12] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.
- [13] J. Dean and S. Ghemawat. MapReduce: a Flexible Data Processing Tool. *Commun. ACM*, 53(1), 2010.
- [14] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *SOSP*, 2003.
- [15] Hadoop Apache Project. <http://hadoop.apache.org>.
- [16] Hive. <http://wiki.apache.org/hadoop/Hive>, 2009.
- [17] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. In *SIGMOD*, 2000.
- [18] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: a Not-so-Foreign Language for Data Processing. In *SIGMOD*, 2008.
- [19] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *SIGMOD*, 2004.
- [20] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming*, 13(4), 2005.
- [21] Protocol Buffers: Developer Guide. Available at <http://code.google.com/apis/protocolbuffers/docs/overview.html>.
- [22] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and Parallel DBMSs: Friends or Foes? *Commun. ACM*, 53(1), 2010.
- [23] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *OSDI*, 2008.

```

1 procedure DissectRecord(RecordDecoder decoder,
2     FieldWriter writer, int repetitionLevel):
3     Add current repetitionLevel and definition level to writer
4     seenFields = {} // empty set of integers
5     while decoder has more field values
6         FieldWriter chWriter =
7             child of writer for field read by decoder
8         int chRepetitionLevel = repetitionLevel
9         if set seenFields contains field ID of chWriter
10            chRepetitionLevel = tree depth of chWriter
11        else
12            Add field ID of chWriter to seenFields
13        end if
14        if chWriter corresponds to an atomic field
15            Write value of current field read by decoder
16            using chWriter at chRepetitionLevel
17        else
18            DissectRecord(new RecordDecoder for nested record
19                read by decoder, chWriter, chRepetitionLevel)
20        end if
21    end while
22 end procedure

```

Figure 16: Algorithm for dissecting a record into columns

APPENDIX

A. COLUMN-STRIPPING ALGORITHM

The algorithm for decomposing a record into columns is shown in Figure 16. Procedure `DissectRecord` is passed an instance of a `RecordDecoder`, which is used to traverse binary-encoded records. `FieldWriters` form a tree hierarchy isomorphic to that of the input schema. The root `FieldWriter` is passed to the algorithm for each new record, with `repetitionLevel` set to 0. The primary job of the `DissectRecord` procedure is to maintain the current `repetitionLevel`. The current `definitionLevel` is uniquely determined by the tree position of the current writer, as the sum of the number of optional and repeated fields in the field's path.

The while-loop of the algorithm (Line 5) iterates over all atomic and record-valued fields contained in a given record. The set `seenFields` tracks whether or not a field has been seen in the record. It is used to determine what field has repeated most recently. The child repetition level `chRepetitionLevel` is set to that of the most recently repeated field or else defaults to its parent's level (Lines 9-13). The procedure is invoked recursively on nested records (Line 18).

In Section 4.2 we sketched how `FieldWriters` accumulate levels and propagate them lazily to lower-level writers. This is done as follows: each non-leaf writer keeps a sequence of (repetition, definition) levels. Each writer also has a ‘version’ number associated with it. Simply stated, a writer version is incremented by one whenever a level is added. It is sufficient for children to remember the last parent’s version they synced. If a child writer ever gets its own (non-null) value, it synchronizes its state with the parent by fetching new levels, and only then adds the new data.

Because input data can have thousands of fields and millions of records, it is not feasible to store all levels in memory. Some levels may be temporarily stored in a file on disk. For a lossless encoding of empty (sub)records, non-atomic fields (such as `Name.Language` in Figure 2) may need to have column stripes of their own, containing only levels but no non-NULL values.

B. RECORD ASSEMBLY ALGORITHM

In their on-the-wire representation, records are laid out as pairs of

```

1 Record AssembleRecord(FieldReaders[] readers):
2     record = create a new record
3     lastReader = select the root field reader in readers
4     reader = readers[0]
5     while reader has data
6         Fetch next value from reader
7         if current value is not NULL
8             MoveToLevel(tree level of reader, reader)
9             Append reader's value to record
10        else
11            MoveToLevel(full definition level of reader, reader)
12        end if
13        reader = reader that FSM transitions to
14            when reading next repetition level from reader
15        ReturnToLevel(tree level of reader)
16    end while
17    ReturnToLevel()
18    End all nested records
19    return record
20 end procedure
21
22 MoveToLevel(int newLevel, FieldReader nextReader):
23     End nested records up to the level of the lowest common ancestor
24     of lastReader and nextReader.
25     Start nested records from the level of the lowest common ancestor
26     up to newLevel.
27     Set lastReader to the one at newLevel.
28 end procedure
29
30 ReturnToLevel(int newLevel) {
31     End nested records up to newLevel.
32     Set lastReader to the one at newLevel.
33 end procedure

```

Figure 17: Algorithm for assembling a record from columns

a field identifier followed by a field value. Nested records can be thought of as having an ‘opening tag’ and a ‘closing tag’, similar to XML (actual binary encoding may differ, see [21] for details). In the following, writing opening tags is referred to as ‘starting’ the record, and writing closing tags is called ‘ending’ it.

`AssembleRecord` procedure takes as input a set of `FieldReaders` and (implicitly) the FSM with state transitions between the readers. Variable `reader` holds the current `FieldReader` in the main routine (Line 4). Variable `lastReader` holds the last reader whose value we appended to the record and is available to all three procedures shown in Figure 17. The main while-loop is at Line 5. We fetch the next value from the current reader. If the value is not NULL, which is determined by looking at its definition level, we synchronize the record being assembled to the record structure of the current reader in the method `MoveToLevel`, and append the field value to the record. Otherwise, we merely adjust the record structure without appending any value—which needs to be done if empty records are present. On Line 12, we use a ‘full definition level’. Recall that the definition level factors out required fields (only repeated and optional fields are counted). Full definition level takes all fields into account.

Procedure `MoveToLevel` transitions the record from the state of the `lastReader` to that of the `nextReader` (see Line 22). For example, suppose the `lastReader` corresponds to `Links.Backward` in Figure 2 and `nextReader` is `Name.Language.Code`. The method ends the nested record `Links` and starts new records `Name` and `Language`, in that order. Procedure `ReturnsToLevel` (Line 30) is a counterpart of `MoveToLevel` that only ends current records without starting any new ones.

```

1 procedure ConstructFSM(Field[] fields):
2   for each field in fields:
3     maxLevel = maximal repetition level of field
4     barrier = next field after field or final FSM state otherwise
5     barrierLevel = common repetition level of field and barrier
6     for each preField before field whose
7       repetition level is larger than barrierLevel:
8       backLevel = common repetition level of preField and field
9       Set transition (field, backLevel) -> preField
10    end for
11   for each level in [barrierLevel+1..maxLevel]
12     that lacks transition from field:
13     Copy transition's destination from that of level-1
14   end for
15   for each level in [0..barrierLevel]:
16     Set transition (field, level) -> barrier
17   end for
18 end for
19 end procedure

```

Figure 18: Algorithm to construct a record assembly automaton

C. FSM CONSTRUCTION ALGORITHM

Figure 18 shows an algorithm for constructing a finite-state machine that performs record assembly. The algorithm takes as input the fields that should be populated in the records, in the order in which they appear in the schema. The algorithm uses a concept of a ‘common repetition level’ of two fields, which is the repetition level of their lowest common ancestor. For example, the common repetition level of Links.Backward and Links.Forward equals 1. The second concept is that of a ‘barrier’, which is the next field in the sequence after the current one. The intuition is that we try to process each field one by one until the barrier is hit and requires a jump to a previously seen field.

The algorithm consists of three steps. In Step 1 (Lines 6-10), we go through the common repetition levels backwards. These are guaranteed to be non-increasing. For each repetition level we encounter, we pick the left-most field in the sequence—that is the one we need to transition to when that repetition level is returned by a FieldReader. In Step 2, we fill the gaps (Lines 11-14). The gaps arise because not all repetition levels are present in the common repetition levels computed at Line 8. In Step 3 (Lines 15-17), we set transitions for all levels that are equal to or below the barrier level to jump to the barrier field. If a FieldReader produces such a level, we need to continue constructing the nested record and do not need to bounce off the barrier.

D. SELECT-PROJECT-AGGREGATE EVALUATION ALGORITHM

Figure 19 shows the algorithm used for evaluating select-project-aggregate queries in Dremel. The algorithm addresses a general case when a query may reference repeated fields; a simpler optimized version is used for flat-relational queries, i.e., those referencing only required and optional fields. The algorithm has two implicit inputs: a set of FieldReaders, one for each field appearing in the query, and a set of scalar expressions, including aggregate expressions, present in the query. The repetition level of a scalar expression (used in Line 8) is determined as the maximum repetition level of the fields used in that expression.

In essence, the algorithm advances the readers in lockstep to the next set of values, and, if the selection conditions are met, emits the projected values. Selection and projection are controlled by two variables, `fetchLevel` and `selectLevel`. During execution, only

```

1 procedure Scan():
2   fetchLevel = 0
3   selectLevel = 0
4   while stopping conditions are not met:
5     Fetch()
6     if WHERE clause evaluates to true:
7       for each expression in SELECT clause:
8         if (repetition level of expression) >= selectLevel:
9           Emit value of expression
10      end if
11    end for
12    selectLevel = fetchLevel
13  else
14    selectLevel = min(selectLevel, fetchLevel)
15  end if
16 end while
17 end procedure
18
19 procedure Fetch():
20   nextLevel = 0
21   for each reader in field reader set:
22     if (next repetition level of reader) >= fetchLevel:
23       Advance reader to the next value
24     endif
25     nextLevel = max(nextLevel, next repetition level of reader)
26   end for
27   fetchLevel = nextLevel
28 end procedure

```

Figure 19: Algorithm for evaluating select-project-aggregate queries over columnar input, bypassing record assembly

readers whose next repetition level is no less than `fetchLevel` are advanced (see `Fetch` method at Line 19). In a similar vein, only expressions whose current repetition level is no less than `selectLevel` are emitted (Lines 7-10). The algorithm ensures that expressions at a higher-level of nesting, i.e., those having a smaller repetition level, get evaluated and emitted only once for each deeper nested expression.

Annexe 3 : Data Warehouse Tools: Faster Time-to-Value for Your Healthcare Data Warehouse

Insights



Data Warehouse Tools: Faster Time-to-Value for Your Healthcare Data Warehouse

By Douglas Adamson

One of the biggest challenges organizations face when implementing an enterprise data warehouse (EDW) is the time it takes from the start of the project to the point where it's delivering value. Typically, this has been anywhere from a 12 to 24 month process. This is where data warehouse tools can help.

One of the biggest challenges organizations face when implementing an enterprise data warehouse (EDW) is the time it takes from the start of the project to the point where it's delivering value. Typically, this has been anywhere from a 12 to 24 month process. This is where data warehouse tools can help.

Under any circumstances that's a long time to wait. In fact, it can feel like an eternity, especially if you have population health management initiatives that can't be launched until the EDW is fully functional. But now, with all the pressure coming down on hospitals and health systems regarding accountable care and moving to a pay-for-performance model, taking that long to complete an EDW implementation is simply unacceptable.

It's a problem we at Health Catalyst® began addressing a couple of years ago. Our original goal was to cut that time-to-value down to six months – a goal we have achieved, by the way. But now that we're there, we're seeing we can still improve on that. So while the rest of the industry is still measuring time-to-value in years, our objective is to bring it down to three to four weeks. As the old saying goes, make no small plans.

Problems Around Healthcare Metadata

One common reason that EDW implementations take so long is how much manual work goes into making it possible to extract data from source systems, such as EHRs or the general ledger, and bring it into the data marts for the EDW. The first step is determining where the needed data resides in each source system. That's what metadata — data about the data, such as the names of tables and columns — tells us.

The source mapping process, facilitated by SMD, is an essential step in creating the proper metadata to drive our automated extract, transform, and load (ETL) process. Our unique, metadata-driven ETL process enables a high degree of automation for nightly ETL practices.

Mapping that metadata manually, as it is normally done, is a tedious, labor-intensive chore. Take a typical EHR system for example. An EHR can have more than 1,000 tables, each of which can contain 100 or more columns. To visualize how much data that is, think of 1,000 Excel spreadsheets with 100 columns each, all open at the same time. That is a lot of metadata to examine and assign a value to.

What makes it worse, however, is that there hasn't been any consistency in working with the metadata. The long, arduous effort normally expended on metadata mapping can't be re-used the next time you are working with that same source system. The result is every new implementation requires the same amount of work.

Data Warehouse Tools: The Source Mart Designer

It was this issue that drove Health Catalyst to develop our Source Mart Designer (SMD) tool. The SMD is a knowledge base that collects everything we learn about mapping metadata from all the different source systems and makes it available in a single, comprehensive resource. Rather than starting over each time, we can go to the SMD and pull out that knowledge, helping us greatly reduce the amount of time spent just determining where to begin.

But it isn't just about knowing where to start. The SMD also automates much of the mapping we used to perform manually. You can point the SMD tool at the source system and the tool will pull out the raw data, giving us an incredible head start on the mapping process. This type of automation not only greatly speeds up the process but also improves the quality and consistency of the work. Greater consistency, of course, helps drive re-use and speeds up the process even more for the next time.

The source mapping process, facilitated by SMD, is an essential step in creating the proper metadata to drive our automated extract, transform, and load (ETL) process. Our unique, metadata-driven ETL process enables a high degree of automation for nightly ETL practices. A few simple, Health Catalyst-supplied ETL scripts are used to load any number of disparate source systems into the EDW.

Once a new source system is mapped, an ETL script is selected to load the new source system. The ETL process is driven entirely by the metadata of that source. This high degree of automation is a

Improving time-to-value involves more than simply mapping metadata from the source system to the EDW, however. Once it's there, the next step is for Health Catalyst's experts to rename it into terms readable by humans, making it easier to search.

huge advantage over traditional data warehousing systems in that ETL developers are unnecessary because maintenance of hundreds of ETL scripts is nonexistent. Metadata is the heart of the Health Catalyst solution, and SMD makes the creation and maintenance of that metadata simple and intuitive.

To fully understand the impact the SMD is having, consider that there are at least 60 significant EHR, patient satisfaction, general ledger, and other source systems in healthcare. To date, we have mapped more than half of those, with more added to our knowledge base every day. The more systems we have mapped, the more we can reuse that knowledge, enabling us to deliver faster time-to-value for the EDW at a lower cost.

Making the Metadata More Useful

Improving time-to-value involves more than simply mapping metadata from the source system to the EDW, however. Once it's there, the next step is for Health Catalyst's experts to rename it into terms readable by humans, making it easier to search.

For example, the EHR source system may have a column labeled Pat_ID. While that may make sense to someone who has been in healthcare IT software a long time, it's one of many names that could be used. It's also not the way a person unfamiliar with that code would search for that information.

Knowing this, the Health Catalyst team will change that column heading to the more intuitive PatientID. When users search for "Patient Identifier" information, they will be far more likely to receive the results they want.

The uses of suffixes, like ID in the case of PatientID, convey additional meaning to a name. Another example is the addition of a suffix such as NBR (indicating that the column contains a numerical value) or CNT (a count such as how many times a patient has had a particular type of test). The team uses its extensive knowledge of the source system, supplemented by the knowledge base in the SMD, to change all the cryptic source system names into ones reflecting the way humans think. All of that data is then placed into the metadata repository so users can browse through it with another tool we've developed called Atlas.

The Atlas tool is designed to simplify and speed up search by making it more intuitive. It's very similar to the way Google works for general internet searching.

The Search for Metadata

The Atlas tool is designed to simplify and speed up search by making it more intuitive. It's very similar to the way Google works for general internet searching. If you're a clinician who wants to find a patient identifier in the EHR, you wouldn't enter Pat_ID as your search term unless you already knew that's the term you needed. You would enter PatientID because that's how you think of it.

Atlas can browse for that term, or something similar, in the metadata and not only find the identifier but tell you additional information about it, such as the source system it came from, the table it came from, and the type of data it is (a number, for example). By taking cryptic, proprietary terms and turning them into terms a human would use and search for, we are able to speed up the process of building subject area marts and metrics based on the data. It also means users who are building the subject area marts don't need to consult an expert in the source system to help them find what they need. They can type in the terms they would use intuitively and Atlas will find the metadata they need.

Improving Population Health Management

This capability for intuitive search, without a requirement to understand all the proprietary terms, is critical when pursuing a population health management or ACO strategy. Because those systems incorporate data from many different hospitals, clinics, primary care physicians, specialists, etc., there is a high likelihood that several different EHRs from several different vendors are being used.

Once all of those different EHRs are mapped into the EDW with our SMD, Atlas can be used to find all instances of Patient Identifiers in the different systems without having to know what proprietary term is used by each of the EHRs. Users can type in "Patient Identifier," and it will find that information in the data mart, regardless its native EHR.

The same is true for any source system – general ledger, patient satisfaction, etc. If you need a Patient Identifier from any of those systems, you can use the same intuitive language to search for it. The result is it makes it easier to function in a heterogeneous environment which is the direction the entire healthcare industry is headed. Until we have complete standardization and interoperability – which may be well off into the future – these tools provide the next best thing. And in a timely fashion.

In most systems, creating metadata is a manual process that can get out of sync quickly. With our approach, the metadata is updated and reports are rebuilt automatically, which continues to deliver improved time to value on a daily basis.

Real-world Example

One of our best examples of using SMD and Atlas to improve time-to-value comes from the work we did with Indiana University Health (IU Health). It was the first time we'd seen the Cerner EHR, yet using SMD, we were able to map it in just 90 days. That's a process that used to take anywhere from 180 days to a year. Even better, as a result of that project, we were able to incorporate what we learned about Cerner into our knowledge base, so the next time we face a Cerner EHR we will be able to do it even faster.

Accelerating Healthcare Data Warehouse Time-to-Value

It's all about getting that tedious front-end process completed quickly, so you can start improving patient outcomes/satisfaction and lowering costs sooner instead of trying to solve a mountain of technical problems. Once you're there, you can build applications based on metadata in the data marts rather than having to use the painfully slow process of going all the way back to the source systems, finding an expert who knows how the data is labeled in that system, and then bringing it back in for the application.

This intelligent application of metadata also saves time and improves accuracy for nightly extractions of data from the source systems. Rather than having to use humans to perform the extraction, the metadata will indicate all the table names, column names, and other information about the source system that is used to extract and update the data needed for reporting and other uses.

In most systems, creating metadata is a manual process that can get out of sync quickly. With our approach, the metadata is updated and reports are rebuilt automatically, which continues to deliver improved time to value on a daily basis.

The bottom line: in Health Catalyst's view, metadata is king. It drives the value. The SMD and Atlas are the data warehouse tools you use to manage that kingdom.

Over the past few years, healthcare organizations in general have gotten much better at capturing data. The adoption of EHRs, driven by Meaningful Use requirements, along with many other

Over the past few years, healthcare organizations in general have gotten much better at capturing data. The adoption of EHRs, driven by Meaningful Use requirements, along with many other technologies, has given us the potential to know more than we've ever known about healthcare, especially as it relates to population health management (PHM).

technologies, has given us the potential to know more than we've ever known about healthcare, especially as it relates to population health management (PHM).

The key word in that sentence, however, is "potential." All that data is lying there waiting to be used, and we're accumulating more of it every day. Yet much of it remains right where it is, locked away in source systems rather than helping us actually understand and manage the health of populations.

It's not for lack of will. It's more a matter of how difficult it has always been to extract the data you need (and only the data you need) to build a subject area mart that will allow you to make calculations and use business intelligence (BI) tools to create visualizations that indicate trends.

The [Health Catalyst Late-Binding™ Data Warehouse](#) consolidates the data from EHRs and other hospital systems into one unified repository simplifying the process of combining disparate data sources into knowledge.

Data Warehouse Tools: The Subject Area Mart Designer

It was this issue that led Health Catalyst to create a tool we call the Subject Area Mart Designer (SAMD). It uses the work performed with our SMD (described above) to simplify the process of accessing the data you extract each night to create visualizations on a particular area. A SAM essentially is a subset of data focused on a particular subject.

Say, for example, you want to look at how well your organization is managing its entire diabetic patient population. In the past, pulling the information you needed out of the various source systems so you could perform the calculations was a time-consuming, arduous, manual task that required a lot of specialized IT knowledge. In fact, it could take months to build an individual subject area mart in your EDW using that method.

By relying on the metadata in the EDW's source mart, which uses human-readable, easily searchable terms rather than the source data and its proprietary terminology, the SAM Designer simplifies much of the process. Since you don't have to go all the way back to

By relying on the metadata in the EDW's source mart, which uses human-readable, easily searchable terms rather than the source data and its proprietary terminology, the SAM Designer simplifies much of the process.

the source system and root around in there to determine where the information you need is located, or how it's labeled, you don't have to be an IT expert (or wait until one becomes available) to create a SAM.

Here's where it's made a huge difference in delivering value. In the past, a data architect would sit with a clinician and try to interpret what that clinician was saying. For all intents and purposes they were speaking different languages, which made the process of building the SAM very slow.

Now, clinicians and data architects can actively work together to build the SAM. In most cases a SAM can be built in just one day – which means the organization starts realizing value from that SAM immediately. Once the basic SAM is in place you can make continual improvements that increase its value even further.

Looking again at the example of the diabetic patients, if you want to see how the organization is doing managing that population you can build a SAM goes into the metadata and draws in the relevant fields and types of data for managing those patients, such as HbA1c levels, and filters out anything you don't need. Once that data is there you can run calculations such as:

- ➊ How many patients in that population have their HbA1c levels under control, i.e., below 7 or whatever number you choose to use?
- ➋ How many of the patients who are out of control have I seen recently?
- ➌ Are any patients trending outside their normal A1c values, where they could be putting themselves at risk?

In short, you can ask whatever questions you need to manage the health of that particular population and get the answers back quickly.

You can see what a huge improvement that is in time-to-value. That's why Health Catalyst is making such a huge investment in this particular tool. We've already released a version of the SAM Designer to rave reviews from our customers. The focus is on giving business analysts and the more tech-savvy clinicians the ability to build SAMs without any IT involvement at all.

The key in delivering time-to-value in healthcare is getting the technology out of the way of the clinicians to ensure they're spending their time practicing medicine instead of their software skills.

It's also a tool we've been using extensively internally to create SAMs for our clients, and to build our applications. For example, our analytics applications team is using the same tool our customers use to create SAMs we can offer as stand-alone applications to aid clients looking for more of a turnkey-type approach (as opposed to building the SAMs themselves).

Making Technology Work for Clinicians

Of course, since everyone in the U.S. practices medicine a little differently, there's also a need to manage those differences. Our plan is to ship SAMs as starter sets to provide a very high beginning point, and let the organizations customize the details (such as changing parameters or enhancing metrics) based on their needs.

While I am a technologist at heart and find all the details that goes into extracting and using data fascinating, I also realize that I am in the minority. The key in delivering time-to-value in healthcare is getting the technology out of the way of the clinicians to ensure they're spending their time practicing medicine instead of their software skills. Making it easy to get to the data needed in order to create calculations and visualizations is critical to managing the health of populations effectively. The SAM Designer is one more way we're accomplishing that.



About the Author

Douglas Adamson joined Health Catalyst in June 2012 as Vice President of Architecture. Prior to joining Catalyst, Doug worked for GE Healthcare in a number of roles including Chief Technologist, Chief Architect and General Manager of Engineering. Doug also spent 14 years working as a software engineer on the Human Genome Project. He holds a Bachelor of Science degree in Computer Science from Purdue University in West Lafayette, Indiana with additional graduate work in Computer Science and Math.

Resources

- ➊ Late-Binding™ Data Warehouse Platform <http://www.healthcatalyst.com/late-binding-data-warehouse-platform>

ABOUT HEALTH CATALYST

Based in Salt Lake City, Health Catalyst delivers a proven, Late-Binding™ Data Warehouse platform and analytic applications that actually work in today's transforming healthcare environment. Health Catalyst data warehouse platforms aggregate and harness more than 3 trillion data points utilized in population health and ACO projects in support of over 22 million unique patients. Health Catalyst platform clients operate 96 hospitals and 1,095 clinics that account for over \$77 billion in care delivered annually. Health Catalyst maintains a current KLAS customer satisfaction score of 90/100, received the highest vendor rating in Chilmark's 2013 Clinical Analytics Market Trends Report, and was selected as a 2013 Gartner Cool Vendor. Health Catalyst was also recognized in 2013 as one of the best places to work by both Modern Healthcare magazine and Utah Business magazine.

Health Catalyst's platform and applications are being utilized at leading health systems including Allina Health, Indiana University Health, Memorial Hospital at Gulfport, MultiCare Health System, North Memorial Health Care, Providence Health & Services, Stanford Hospital & Clinics, and Texas Children's Hospital. Health Catalyst investors include CHV Capital (an Indiana University Health Company), HB Ventures, Kaiser Permanente Ventures, Norwest Venture Partners, Partners HealthCare, Sequoia Capital, and Sorenson Capital.

Visit www.healthcatalyst.com, and follow us on [Twitter](#), [LinkedIn](#), [Google+](#) and [Facebook](#).