A Data Vault Implementation for Neuroimaging Data

Abdelhameed, Shahd
sxa1240@student.bham.ac.uk

Alnoaimi, Hasan
hma120@student.bham.ac.uk

Liu, Ethan
ecl734@student.bham.ac.uk

Wang, Haoran
hxw188@student.bham.ac.uk

Woo, Doris
dxw103@student.bham.ac.uk

Storing and Managing Data (Module 06-32245), Autumn 2021, Semester Project
10 December 2021

***Abstract*—Functional near infrared spectroscopy (fNIRS) and electroencephalography (EEG) are complex procedures for monitoring brain activity. These processes produce analogue and digital readings from inducers which are required to be stored and manipulated before invoking further insight. The project aims to build a scalable and flexible medical imaging data vault in order to provide a system that allows for data to be stored in an intelligent operation-agnostic manner, but also as close to their original form as possible. For staging, the data structure of DataFrames was exploited to increase data manipulation efficiency and flexibility for extracting, transforming and loading (ETL) via a streamlined, automated and modular data pipeline. All data sources were standardised and stored in a repository with minimal user input. Success was achieved with building the raw data vault to enable simple and complex queries via Django-powered Web interface. Data cleaning was also achieved prior to fNIRS visualisation to remove noise in the form of frequency caused by non-haemoglobin activity. Other visualisations included response images showing relationships between dependent variables, and plots showing correlations between optical channels. Overall, the data vault model provided a robust solution to produce insights from fNIRS data. Detailed visualisations from cleansed data were revealed complex relationships, whilst allowing querying of specific data for correlation analysis. Although there remain aspects of the project which can be improved, the they can be implemented without compromising the scalable and flexible structure of the data vault.**

Keywords—data management, data vaults, DataFrames, Django, medical imaging, PostgreSQL, relational database

The authors have unanimously agreed to acknowledge that each author contributed equally to the project.

I. Introduction

In the field of neuroscience, there are many methods for monitoring different aspects of the human brain, depending on the aim, context and numerous other factors involved that defines a particular study [7], [17], [20].

This project will incorporate characteristics from the optical method of functional near-infrared spectroscopy (fNIRS) [11], [12], as well as the electrical method of electroencephalography (EEG) to build a data vault that can store, manifest and elicit collected neural activity data [15], [19].

By applying the Data Vault 2.0 model [5], the project will prioritise the accessibility of data in their original form stored within a uniform data structure repository. The overall objective is to build a scalable, flexible, and efficient medical imaging data vault, as a potential solution for storing and aid in the complex procedure of collection and elucidation of neurological data under various conditions and stimuli.

The following sections consist of a critical evaluation of data vault features, such as staging and schema development, which were carried out during the project lifespan and will provide an accurate reflection of the dynamic data vault construction process. An in-depth discussion is also presented to review the data vault as a whole and to what extent it is successful as a medical imaging data solution.

II. State of the Art of Data Vaults as Applied to Medical Imaging

Our literature review focused on applications of the data vault model in various medical and scientific fields. The publications addressed a number of common themes with respect to limitations in data warehouses using a normalised, relational database management system (DBMS) model, the drawbacks such limitations can have on scientific research, and ways in which data vaults can mitigate such issues:

- Relational data models lack flexibility to adapt to ongoing changes to data requirements based on user/business needs, changes to available technology and data sources and regulatory compliance such as data privacy laws [2], [4], [15]. Implementing those changes require modifying the data model itself which may result in data being offline and/or unfit for purpose for a period of time; the structure of data vaults allows such changes without altering the model or the same degree of downtime.
- Data provenance (i.e. preserving the origins and history of data) [1] is not a native feature of normalised, relational data models, yet is essential and better supported in data vault models for data traceability for supporting longitudinal studies and other research requiring data reuse [1], [2], [15].
- Allowing raw data to be loaded with minimal to no modifications may in many cases render the DBMS to only perform rudimentary in-database file processing of scientific data [4]; adopting hybrid solutions which allow raw files to be stored in databases is a stopgap which still relies on middleware and other external applications to handle more complex computations and data analysis [4], [15]. However, more robust integration with metadata in a data vault implementation can minimise risks of out-of-database

data processing creating data inconsistencies with the data warehouse.

- Closely related to the aforementioned points, in order to support heterogeneous data sources for maximum flexibility and data reuse, relevant additions and changes to the database objects and relationships between them must be supported in a data model in a scalable way [1], which a data vault model is able to achieve more naturally by separating information about the structure of a database object from the instantiation of the object itself [2].

The use cases highlighted in these publications which address the above issues with data vault-based solutions include:

- Improved sharing of externally stored sensor data between members of seismography networks for visualisations, aggregation and outlier detection [4].
- Correlating observations from astronomy surveys over the same sky objects stored among heterogeneous data sources [4].
- A data management system of neuroimaging cohorts which improves provenance management, sharing and re-use of data for different types of specialist neurological and longitudinal studies [1].
- Implementing data marts which integrate isolated data sources for aggregated or longitudinal studies of health IT adoption surveys, with the ability for changes to the data structure and additional components for dealing with data to be added to be made without changes to the structure of the data vault model [15].

III. Methods

A. *Schema*

Introduction

Before constructing a conceptual design for our data vault, we established that the main objectives were for the data vault to be scalable and flexible. As such, our schema design (Fig.1) was constructed to reflect those objectives.

The design allows for a systematic and seamless flow of information starting from the patient hub and satellite, which store relevant patient information, which in turn are connected to the experiment hub and satellites that store the relevant metadata for each experiment. This approach allows for flexibility and scalability since any increase in the number of patients and the number of experiments can be incorporated into these hubs and satellites without any time-consuming changes--regardless of the degree to which the amount of available data increases or business/research requirements change in the medical imaging domain. Flexibility and scalability are further embedded in our design given how the sessions hub and data sources hubs are created: any new sessions in an experiment can be added directly to the session's hub and satellite without the need to create further structures or risking changes that can lead to data inconsistencies. Moreover, if more data sources are obtained in future experiments, e.g. magnetic resonance imaging (MRI) data, then this can be easily integrated into the data vault through the addition of an MRI hub and an MRI satellite. The MRI hub can be directly linked to the data sources hub through a link table to represent the relationship between them. This was the approach taken to represent the fNIRS data and the EEG data sources and is a well-informed design choice. Furthermore, organising the data vault as defined by our schema allows for each hub (e.g. the fNIRS hub) to have multiple satellites that can store various fNIRS data in a consistent way such that future data can once again be easily integrated into the vault.

Although our conceptual design considered all the hubs, satellites and links that would be relevant for the development of a complete data vault for medical imaging data. It was decided to not include the EEG hub and satellite in the physical implementation. Instead, priority was given to fully implement and test the other remaining aspects of our schema at the staging level without processing the .mat files included in the EEG data. With fewer time constraints, we would have provided a full implementation of the EEG hub and satellite.

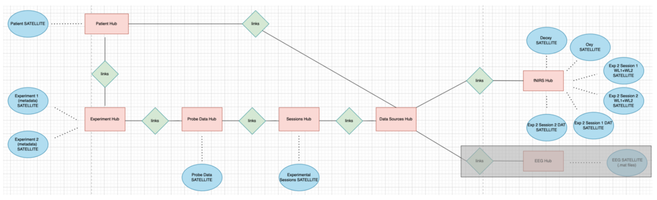

Patient SATELLITE
Patient Hub
links
links
Experiment 1 (metadata) SATELLITE
Experiment 2 (metadata) SATELLITE
Experiment Hub
links
Probe Data Hub
links
Sessions Hub
links
Data Sources Hub
Probe Data SATELLITE
Experimental Sessions SATELLITE
links
fNIRS Hub
Deoxy SATELLITE
Oxy SATELLITE
Exp 2 Session 1 WL1+WL2 SATELLITE
Exp 2 Session 2 WL1+WL2 SATELLITE
Exp 2 Session 2 DAT SATELLITE
Exp 2 Session 1 DAT SATELLITE
links
EEG Hub
EEG SATELLITE (.mat files)

Fig. 1. The schema and conceptual design of our data vault. The grey-shaded area is only part of the conceptual design and has not been included in the physical implementation of the data vault.

Physical Implementation

Hubs

From the conceptual to physical design, the hubs created contain four main attributes (columns) as shown in Table I.

TABLE I. SQL Data Types for Hubs

Hub Attributes	**Data type**
Hub Key (Primary Key)	Serial
Business Key	Varchar
Load Date Time	Timestamp
Record Source	Varchar

Traditionally, the hub key is a hash key created to provide a surrogate key for business keys, composite business keys, and business key combinations. However, due to time constraints, our hub keys were not generated using the hash function in Python. A stopgap was implemented instead in SQL using a serial data type in place of a hashed hub key. The hub key is used as a unique identifier and is effectively the primary key for the hub.

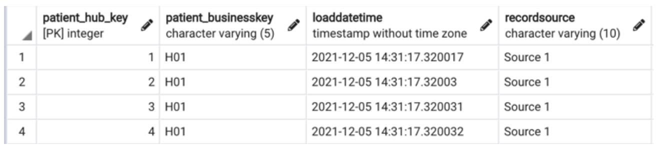
The business key in our hubs represents who entered the information and has been given the varchar data type in SQL. The business keys used throughout the data vault were “H01” and “E02”, representing two of the group members. The business key can be generated with a similar format (single letter and number) for additional admins who want to populate the data vault. This structure allows for the hubs and overall data vault to remain flexible and scalable, so that no matter how business requirements change, the business key can be linked to an individual without risk of errors in the data (e.g. if there are two people assigned the prefix “H” to the start of their business key, they can still be distinguished by unique numeric suffixes).

The load date time attribute describes when the data ingestion happened and is key for traceability, version tracking, and an essential component of data provenance principle highlighted in the literature review section.

The record source records the source from which the data was originally loaded [2]. As all the data was provided together in the project specification, a single data source has been assumed. Therefore, all record source attributes in the hubs are identified as "Source 1". If more data are obtained or business changes occur which affect the labelling of existing data sources, new record sources can be created (e.g. Sources 2 and 3), without needing to make time-consuming changes to the overall data vault structure.

An example of how the hub attributes appear in the physical implementation is shown in Table II below.

TABLE II. PATIENT HUB TABLE IN PGADMIN



	patient_hub_key [PK] integer	patient_businesskey character varying (5)	loaddatetime timestamp without time zone	recordsource character varying (10)
1	1	H01	2021-12-05 14:31:17.320017	Source 1
2	2	H01	2021-12-05 14:31:17.32003	Source 1
3	3	H01	2021-12-05 14:31:17.320031	Source 1
4	4	H01	2021-12-05 14:31:17.320032	Source 1

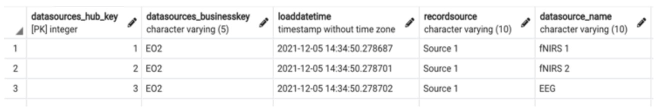
In total, we created six hubs for the datasets supplied for the project:

- The patient hub stores data from all 10 patients from experiment 1 and all 43 patients from experiment 2.
- The experiment hub represents the two experiments being conducted.
- The probe data hub stores the data from the probe.
- The sessions hub has six sessions, 4 of which are relevant to experiment 1 (moto, rest, viso (visual) and vimo (visuomotor)), and 2 of which are relevant to experiment 2 (stressed conversation and normal conversation).
- The data source hub stores the different sources. For the provided datasets, there are three such sources: fNIRS 1 from experiment 1, fNIRS 2 from experiment 2 and EEG data from experiment 2.
- The fNIRS hub stores all relevant fNIRS data from the experiments.

The actual data and descriptive attributes related to these hubs are stored in the satellites and will be discussed in the next section.

Some hubs contain additional attributes which extends the basic hub structure shown in Table I, and is a design choice made to improve query processing times. These attributes include experiment type, with possible values being "experiment 1" and "experiment 2" for the provided datasets, and data source name in the data source hub to identify fNIRS data from experiment 1, fNIRS data from experiment 2, or EEG data from experiment 2, as shown in Table III. The addition of extra attributes such as experiment type and data sources is consistent with the overarching purpose of the data vault to be flexible and scalable; new experiments and data sources can be added without reconstructing the schema.

TABLE III. DATA SOURCES HUB TABLE IN PGADMIN



	datasources_hub_key [PK] integer	datasources_businesskey character varying (5)	loaddatetime timestamp without time zone	recordsource character varying (10)	datasource_name character varying (10)
1	1	E02	2021-12-05 14:34:50.278687	Source 1	fNIRS 1
2	2	E02	2021-12-05 14:34:50.278701	Source 1	fNIRS 2
3	3	E02	2021-12-05 14:34:50.278702	Source 1	EEG

To improve on our physical design of the hubs, we would carry out the full implementation of the EEG hub to complete our data vault model and ensure maximum functionality. Another improvement would be to implement the hash function in Python to generate conventional hub keys.

Links

The link mainly identifies the relationship between hubs but is also occasionally used to represent the relationships between satellites. The flexibility in our data vault is owed partly to the link tables. Every relationship, regardless of its cardinality, is always represented as a table signifying an aggregation of two related objects (in our case, the two hub keys from the two hubs that have a relationship) [2]. Therefore, if changes in business processes require the addition of new hubs, their relationships to any existing entities can be represented by additional link tables in the data vault, creating minimal impact to the data vault structure.

Every link table in the physical implementation contain the following key attributes:

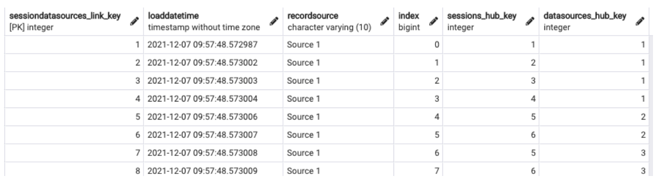
TABLE IV. SQL DATA TYPES FOR LINK ATTRIBUTES

Link Attributes	Data type
Link Key (Primary Key)	Serial
Load Date Time (Primary Key)	Timestamp
Record Source	Varchar
Hub 1 Key (Foreign Key)	Int
Hub 2 Key (Foreign Key)	Int

The link key behaves in the same way as a hub key in a hub, i.e. as the unique identifier and primary key. As explained previously with the non-hashed hub key stopgap, the link key uses the serial data type instead on the same basis. The load date time represents the date and time when the values were stored/loaded into the link. Like its function in a hub, the record source records the source from which the data was originally loaded and has been populated with Source 1 in our link tables. The two hub keys are included as foreign keys in the link to represent the relationship between them.

In our design, the links created in the schema have only been used to represent the relationships between hubs. A link table has been created to represent the relationship between the patient hub and the experiment hub. Another link table has been created to represent the relationship between the experiment hub and probe data hub, and the relationship between the probe data hub and sessions hub. A link has also been created between the session's hub and the data sources hub and is illustrated below in Table V. A similar link table was also created to represent the relationship between the data sources hub and the fNIRS hub. To improve connectivity within the data vault and query processing times, an additional link was also created to represent the relationship between the patient hub and the data sources hub. To improve on the existing data vault model, this approach can be repeated wherever there is scope to define more relationships and shorten data pathways for queries.

TABLE V. LINK TABLE FOR EXPERIMENTAL SESSIONS AND DATA SOURCES HUBS IN PGADMIN



sessiondatasources_link_key [PK] integer	loaddatetime timestamp without time zone	recordsource character varying (10)	index bigint	sessions_hub_key integer	datasources_hub_key integer
1	2021-12-07 09:57:48.572987	Source 1	0	1	1
2	2021-12-07 09:57:48.573002	Source 1	1	2	1
3	2021-12-07 09:57:48.573003	Source 1	2	3	1
4	2021-12-07 09:57:48.573004	Source 1	3	4	1
5	2021-12-07 09:57:48.573006	Source 1	4	5	2
6	2021-12-07 09:57:48.573007	Source 1	5	6	2
7	2021-12-07 09:57:48.573008	Source 1	6	5	3
8	2021-12-07 09:57:48.573009	Source 1	7	6	3

For example, based on our schema, the satellites "fNIRS_Deoxy_Satellite" and "fNIRS_Oxy_Satellite" could have a link to represent the relationship between them. Likewise, links can be added between the two fNIRS .dat satellites, and again between the two fNIRS .wl1 .wl2 satellites.

Satellites

Satellites store the context of the hubs and contains all attributes and descriptors relevant to a given hub [2]. All data capture and history is stored in the satellites. The key attributes contained in a satellite have been summarised in Table VI.

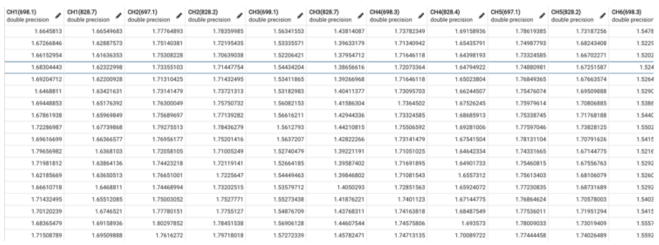
TABLE VI. SQL DATA TYPES FOR SATELLITE ATTRIBUTES

Satellite Attributes	Data type
Hub Key[a] (Primary and Foreign Key)	Serial
Load Date Time (Primary Key)	Timestamp
Other non-key attributes	various

a. Refers to the Hub Key from the hub that the satellite is associated with.

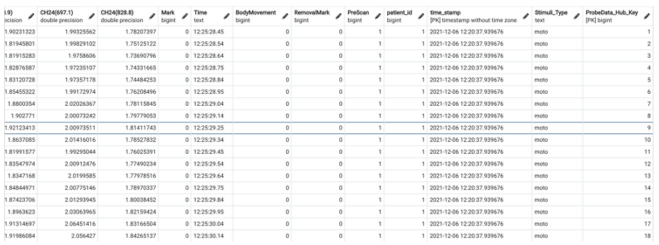
The hub key references the hub with which the satellite is associated and is a composite primary key with the load date time attribute. In the same manner as implemented in the hub, load date time represents the date and time when the values were loaded and stored into the satellite. A physical representation of these attributes in our data vault can be seen in Tables VII and VIII of our probe data satellite.

TABLE VII. EXCERPT OF THE PROBEDATA_SATELLITE TABLE



CH1(698.1) double precision	CH1(828.7) double precision	CH2(697.1) double precision	CH2(828.2) double precision	CH3(698.1) double precision	CH3(828.7) double precision	CH4(698.3) double precision	CH4(828.4) double precision	CH5(697.1) double precision	CH5(828.2) double precision	CH6(698.3) double precision
1.6645813	1.66549683	1.77764893	1.78359985	1.56341553	1.43814087	1.73782349	1.69158936	1.78619385	1.73187256	1.5478
1.67266846	1.62887573	1.75140381	1.72195435	1.53335571	1.39633179	1.71340942	1.65435791	1.74987793	1.68243408	1.5229
1.66152954	1.61636353	1.75308228	1.70639038	1.52206421	1.37954712	1.71646118	1.64398193	1.73324585	1.66702271	1.5202
1.68304443	1.62322998	1.73355103	1.71447754	1.54434204	1.38656616	1.72073364	1.64794922	1.74880981	1.67251587	1.524
1.69204712	1.62200928	1.71310425	1.71432495	1.53411865	1.39266968	1.71646118	1.65023804	1.76849365	1.67663574	1.5264
1.6468811	1.63421631	1.73141479	1.73721313	1.53182983	1.40411377	1.73095703	1.66244507	1.75476074	1.69509888	1.529
1.69448853	1.65176392	1.76300049	1.75750732	1.56082153	1.41586304	1.7364502	1.67526245	1.75979614	1.70806885	1.538
1.67861938	1.65969849	1.75689697	1.77139282	1.56616211	1.42944336	1.73324585	1.68685913	1.75338745	1.71768188	1.544
1.72286987	1.67739868	1.79275513	1.78436279	1.5612793	1.44210815	1.75506592	1.69281006	1.77597046	1.73828125	1.5502
1.69616699	1.66366577	1.76956177	1.75201416	1.5637207	1.42822266	1.73141479	1.67541504	1.78131104	1.70791626	1.5415
1.79656982	1.6368103	1.72058105	1.71005249	1.52740479	1.39221191	1.71051025	1.64642334	1.74331665	1.67144775	1.5216
1.71981812	1.63864136	1.74423218	1.72119141	1.52664185	1.39587402	1.71691895	1.64901733	1.75460815	1.67556763	1.5292
1.62185669	1.63650513	1.76651001	1.7225647	1.54449463	1.39846802	1.71081543	1.6557312	1.75613403	1.68106079	1.526
1.66610718	1.6468811	1.74468994	1.73202515	1.53579712	1.4050293	1.72851563	1.65924072	1.77230835	1.68731689	1.5292
1.71432495	1.65512085	1.75003052	1.7527771	1.55273438	1.41876221	1.7401123	1.67144775	1.76864624	1.70578003	1.5403
1.70120239	1.6746521	1.77780151	1.7755127	1.54876709	1.43768311	1.74163818	1.68487549	1.77536011	1.71951294	1.5415
1.68365479	1.69158936	1.80297852	1.78451538	1.56906128	1.44607544	1.74575806	1.693573	1.78009033	1.73019409	1.5557
1.71508789	1.69509888	1.7616272	1.79718018	1.57272339	1.45782471	1.74713135	1.70089722	1.77444458	1.74026489	1.5592

TABLE VIII. PROBEDATA_SATELLITE TABLE (CONTINUED)



.9) ecision	CH24(697.1) double precision	CH24(828.8) double precision	Mark bigint	Time text	BodyMovement bigint	RemovalMark bigint	PreScan bigint	patient_id bigint	time_stamp [PK] timestamp without time zone	Stimuli_Type text	ProbeData_Hub_Key [PK] bigint
1.90231323	1.99325562	1.78207397	0	12:25:28.45	0	0	1	1	2021-12-06 12:20:37.939676	moto	1
1.81945801	1.99829102	1.75125122	0	12:25:28.54	0	0	1	1	2021-12-06 12:20:37.939676	moto	2
1.81915283	1.9758606	1.73690796	0	12:25:28.64	0	0	1	1	2021-12-06 12:20:37.939676	moto	3
1.82876587	1.97235107	1.74331665	0	12:25:28.75	0	0	1	1	2021-12-06 12:20:37.939676	moto	4
1.83120728	1.97357178	1.74484253	0	12:25:28.84	0	0	1	1	2021-12-06 12:20:37.939676	moto	5
1.85455322	1.99172974	1.76208496	0	12:25:28.95	0	0	1	1	2021-12-06 12:20:37.939676	moto	6
1.8800354	2.02026367	1.78115845	0	12:25:29.04	0	0	1	1	2021-12-06 12:20:37.939676	moto	7
1.902771	2.00073242	1.79779053	0	12:25:29.14	0	0	1	1	2021-12-06 12:20:37.939676	moto	8
1.92123413	2.00973511	1.81411743	0	12:25:29.25	0	0	1	1	2021-12-06 12:20:37.939676	moto	9
1.8637085	2.01416016	1.78527832	0	12:25:29.34	0	0	1	1	2021-12-06 12:20:37.939676	moto	10
1.81991577	1.99295044	1.76025391	0	12:25:29.45	0	0	1	1	2021-12-06 12:20:37.939676	moto	11
1.83547974	2.00912476	1.77490234	0	12:25:29.54	0	0	1	1	2021-12-06 12:20:37.939676	moto	12
1.8347168	2.0199585	1.77978516	0	12:25:29.64	0	0	1	1	2021-12-06 12:20:37.939676	moto	13
1.84844971	2.00775146	1.78970337	0	12:25:29.75	0	0	1	1	2021-12-06 12:20:37.939676	moto	14
1.87423706	2.01293945	1.80038452	0	12:25:29.84	0	0	1	1	2021-12-06 12:20:37.939676	moto	15
1.8963623	2.03063965	1.82159424	0	12:25:29.95	0	0	1	1	2021-12-06 12:20:37.939676	moto	16
1.91314697	2.06451416	1.83166504	0	12:25:30.04	0	0	1	1	2021-12-06 12:20:37.939676	moto	17
1.91986084	2.056427	1.84265137	0	12:25:30.14	0	0	1	1	2021-12-06 12:20:37.939676	moto	18

Additional descriptive attributes have been added to each satellite depending on its context and the data relevant to it. The patient satellite contains the name, age, and other descriptive attributes of the patients. The experiment 1 metadata satellite has additional attributes to store all the metadata relevant to experiment 1, and likewise with the experiment 2 metadata satellite. The probe data satellite has been illustrated in Tables VII and VIII and has additional attributes such as the optical channels. The sessions satellite stores all the different sessions in the experiment. The various fNIRS satellites have been created to store the relevant information from each experiment, e.g. a fNIRS deoxy satellite for storing the deoxy files from experiment 1.

Although there can be many satellites for one hub [2], we adopted a more conservative approach of associating one satellite to only one hub. However, multiple satellites for one hub are permitted in our model; the fNIRS hub has six satellites (as shown in the schema in Fig. 1), each storing relevant data from each of the two experiments to improve the contextualisation of the fNIRS hub.

Our satellite structure maintains the same degree of flexibility and scalability that exists for hubs and satellites. For example, if we receive new fNIRS data from a new source, we could easily implement that in our data vault by adding a new fNIRS satellite instead of modifying/changing an existing satellite. Additionally, as all the data in each fNIRS satellite relate to the fNIRS hub, the approach of creating six different fNIRS satellites is further supported by the literature given that "the properties of a Hub can be split into satellites on the basis of the type of information" [2]. Once again, our physical implementation allows us to be resilient towards changes of business needs, data formats, requirements, and volume.

To improve the implementation of our satellites, we could have included a load end date attribute. The load end date would refer to the date and time when values of an attribute are overwritten with new values [2]. Not only would including load end dates be beneficial for providing users with a history of changes to the attribute values, but there would also be improvements to the functionality and query processing of our satellites. Nevertheless, we have achieved our objective of creating a scalable data vault.

B. Staging

Introduction

For the staging process, the primary goal is to reduce the workload of systems by loading all data into a temporary and separate database [5].

Collation of multiple sources into a uniform repository is broken down into three distinct steps of extraction, transformation and load (ETL). The data is first extracted from their original source, transformed in some appropriate way and finally loaded into a repository's data structure.

Through manipulation of the ETL process, this separate relational database approach reduces system workload by allowing technical control of the data in its raw form. The stored data are susceptible to relational database operations as well as delivering data in their actual data type, thus streamlining other data vault model construction processes down the line.

Software and libraries used specifically for staging are displayed in Table IX below:

TABLE IX. SOFTWARE AND PYTHON LIBRARIES USED FOR STAGING

Software / Library	Version
Python	3.9
pandas	1.3.2
numpy	1.21.2
psycopg2	2.92
SQLAlchemy	1.4.27
pgAdmin 4	5.2
PostgreSQL 14	14

Step 1: Deriving Target Data File Formats

After an initial inspection of the given datasets, a table was constructed to isolate the appropriate data files for staging (shown in Tables X and XI below).

In a true data vault model, every file should undergo staging. However, due to the scope and time constraints of this project [2], a small portion of source files were excluded in order to prioritise the construction of a working data vault.

TABLE X. FILE FORMATS INCLUDED IN THE STAGING PROCESS

Target Data File Formats		
File format	**Content**	**Data type to be stored as pandas DataFrames**
.csv	Data and metadata	Numerical and strings
.wl1	Data	Array
.wl2	Data	Array
.dat	Data	Array
.txt	Metadata	Numerical and strings

TABLE XI. FILE FORMATS OMITTED FROM STAGING

Omitted Files	
File format	**Reason**
.cfg	Deemed unessential due to time constraint and prioritisation of a working data vault.
.evt	Deemed unessential due to time constraint and prioritisation of a working data vault.
.hdr	Deemed unessential due to time constraint and prioritisation of a working data vault.
.set	Deemed unessential due to time constraint and prioritisation of a working data vault.
.tpl	Deemed unessential due to time constraint and prioritisation of a working data vault.
groupresults.mat	Derived data; also deemed redundant.
.avg	Derived data; also deemed redundant.

All Python objects are transformed into DataFrame objects due to their fast computational leverage as well as uniform data structure, which also provides extra file integrity for data sources that are both numerical and categorical in nature [10].

Staging consisted of processing both metadata and actual data. Both require slightly different operations due to the nature of the file contents. Specifically, the contents of the .dat, .wl1 and .wl2 files were stored as arrays inside DataFrame objects [7]. This is due to the file content losing their meaning if each value were to be parsed, unlike the .csv files which bypasses parsing before being converted into DataFrame objects.

In general, the metadata from the supplied datasets involved heavy parsing due to the non-uniform structure of the contents. Many attributes involved have differences in length and/or observations; therefore, extra columns were added to accommodate these attributes. The advantage of this was to be able to query specific parts of metadata and compare them, albeit at a cost of increased redundancy due to many null values which also required more memory and disk usage.

Ultimately, it was decided that prioritising accessibility over memory usage optimisation was a justified choice, as data vaults are designed to favour accessibility over redundancy, whilst keeping raw sources unmodified [6].

Step 2: Bulk ETL Pipeline Methodology

Data pipelines were constructed for ETL to automate the process as shown in the Fig. 2 below.

The main features of the automation process increased the flexibility and scalability of the project. Each stage of the ETL process was broken down into several modular functions, which allow for responsive modifications, as well as minimal user input. Regardless of the size of the data involved, should the scope of the data sources and project be expanded in the future, we would benefit from this time-saving feature.

Firstly, a pipeline was built to automatically search the target file formats and create a list of file directories. These included the visuomotor stimuli (VMS) .csv files, which contained both metadata and data, fNIRS .wl1 and .wl2 files. After that the files were split between metadata and data directories.

For the .csv files, as they contained both metadata and data. The metadata rows were parsed first into separate .csv files before being converted into DataFrames. This is due to the aforementioned non-uniform structure of the metadata. Parsing of "\n" delimiters was applied to the .txt files that contained the metadata for .dat files.

The rest of the data contained in .dat, .w12, .w12 and .csv files were simply converted into DataFrame objects. This approach allowed the usage of pandas libraries such as SQLAlchemy and psycopg2 to automatically create the tables into PostgreSQL according to file format and data type, automating and streamlining the entire ETL process.

The output of the staging data pipeline would simply be the required collection of the satellites, all populated and created with the appropriate data types within PostgreSQL.

Additional advantages of using DataFrames to automatically populate SQL tables include:

- Improved original data integrity as data types and attribute names are replicated as much as possible from original file to SQL tables.
- ETL was executed entirely with pandas scripts in Python, making debugging and data manipulation more convenient.
- Conversions from DataFrame to SQL tables were quick due to both data structures being very similar
- Data structures were automatically optimised via auto-indexing after DataFrames were created.

One disadvantage is that constraints such as primary keys must be added after SQL tables are created via the Query Editor in pgAdmin 4. Although it is an additional step, this has been successfully completed in our data vault. Another disadvantage is that the most appropriate data type may not necessarily be the most optimal data type due to automation; changes will require manual coding and input from users.

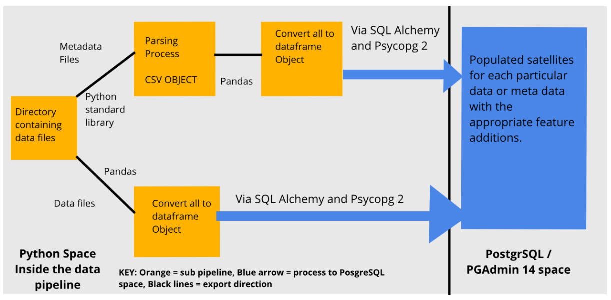
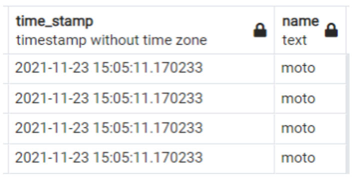

Metadata Files
Parsing Process
CSV OBJECT
Pandas
Convert all to dataframe Object
Via SQL Alchemy and Psycopg 2
Populated satellites for each particular data or meta data with the appropriate feature additions.
Directory containing data files
Python standard library
Pandas
Data files
Convert all to dataframe Object
Via SQL Alchemy and Psycopg 2
Python Space Inside the data pipeline
KEY: Orange = sub pipeline, Blue arrow = process to PosgreSQL space, Black lines = export direction
PostgrSQL / PGAdmin 14 space

Fig. 2. The split of staging data pipeline in order to deal with metadata and data separately, then inserting them into the PostgreSQL database.

One of the key features of a data vault is chronology recording and presenting. Therefore, the ELT pipeline also added timestamps for each processed file. This in return introduced the timestamp data type that would be useful for time-based querying.

Other additional columns were added to satellites to allow and optimise querying from one satellite to another, such as the identification of experiment type for each observation as shown in Table XII below:

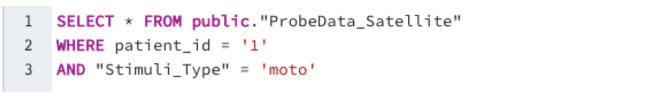
TABLE XII. Excerpt of Added Columns / Features in PostgreSQL Tables



time_stamp timestamp without time zone	name text
2021-11-23 15:05:11.170233	moto
2021-11-23 15:05:11.170233	moto
2021-11-23 15:05:11.170233	moto
2021-11-23 15:05:11.170233	moto

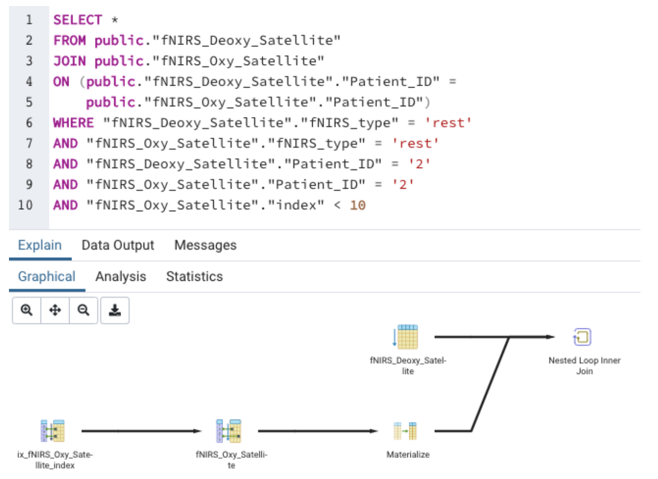
C. Queries

From the loaded-in experiment data, basic queries were performed in pgAdmin as shown in Figs. 3 and 4 to confirm that the data can be retrieved from single and multiple tables and exported as .csv files (which are included in the project data deliverables).



```
SELECT * FROM public."ProbeData_Satellite"
WHERE patient_id = '1'
AND "Stimuli_Type" = 'moto'
```

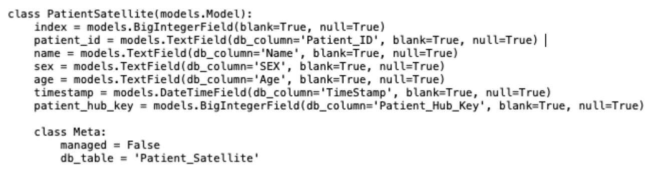
Fig. 3. Querying the *moto* stimulus subset from a single patient.



```
SELECT *
FROM public."fNIRS_Deoxy_Satellite"
JOIN public."fNIRS_Oxy_Satellite"
ON (public."fNIRS_Deoxy_Satellite"."Patient_ID" =
    public."fNIRS_Oxy_Satellite"."Patient_ID")
WHERE "fNIRS_Deoxy_Satellite"."fNIRS_type" = 'rest'
AND "fNIRS_Oxy_Satellite"."fNIRS_type" = 'rest'
AND "fNIRS_Deoxy_Satellite"."Patient_ID" = '2'
AND "fNIRS_Oxy_Satellite"."Patient_ID" = '2'
AND "fNIRS_Oxy_Satellite"."index" < 10
```

Fig. 4. A SQL JOIN query and pgAdmin's graphical representation for one patient's non-stimulus deoxy and oxy data from their respective satellites.

Web-based data queries were implemented with Django, a Python library for providing a Web framework for presenting data. Each table from our PostgreSQL database was converted into Python classes called models [10] using the Django utility inspectdb, as shown in Fig. 5 below.



```
class PatientSatellite(models.Model):
    index = models.BigIntegerField(blank=True, null=True)
    patient_id = models.TextField(db_column='Patient_ID', blank=True, null=True) |
    name = models.TextField(db_column='Name', blank=True, null=True)
    sex = models.TextField(db_column='SEX', blank=True, null=True)
    age = models.TextField(db_column='Age', blank=True, null=True)
    timestamp = models.DateTimeField(db_column='TimeStamp', blank=True, null=True)
    patient_hub_key = models.BigIntegerField(db_column='Patient_Hub_Key', blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'Patient_Satellite'
```

Fig. 5. The Patient_Satellite table represented as the Django model PatientSatellite, which was generated by Django's inspectdb utility.

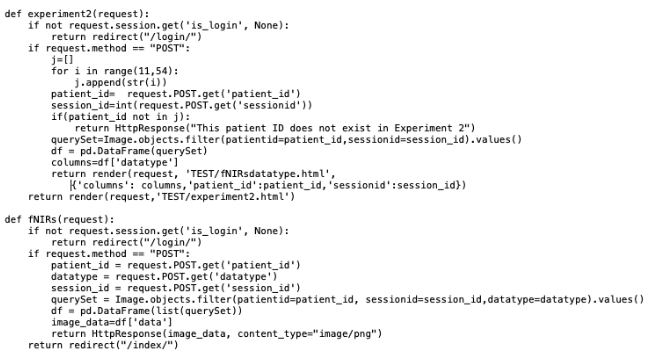
Database queries can also be performed in Django, whose results are returned as QuerySets [9]. From the web-based frontend, the user inputs criteria which are processed as HTTP GET requests with which the QuerySet results are filtered. Further processing of the query results as data visualisations are discussed in the "Results" section below.

D. Data Mart

Django was chosen to implement a Web-based frontend for our data mart/presentation layer. The files required for performing a discrete task within a Django-built website are collectively grouped as a Python module known as an app. In a basic Django implementation, each newly created app will generate a common set of files in its file structure for handling core functions such as models, views, templates and routing content to web addresses (i.e. Uniform Resource Locators, or URLs). A collection of apps used collectively as one web application are stored as a Django project. Our Django project is named "Visualisation"; the app which handles the queries and visualisations is named "TEST".

The 3 tables chosen from models.py for developing our data mart use cases were ProbeData_Satellite, fNIRS_Deoxy_Satellite and fNIRS_Oxy_Satellite.

The Django files which primarily enable the Web-based data presentations are views.py and urls.py: views.py contains the functions which processes the aforementioned QuerySet results into Web responses [17]. For the use cases, the Web responses consist of plots of experiment 1 data, with the plots created on-the-fly with Python's plotly package; experiment 2 visualisations are fNIRS images which were generated from raw data after cleansing with Python's mne package during the staging process and stored in the Images table in PostgreSQL. Fig. 6 below shows the views.py functions responsible for returning fNIRS images from Experiment 2.

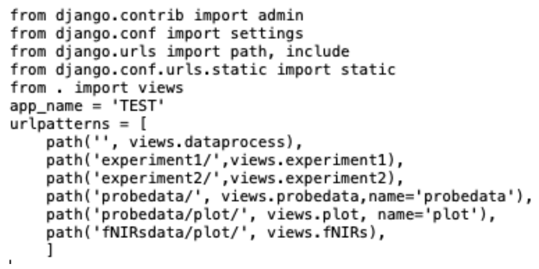


```
def experiment2(request):
    if not request.session.get('is_login', None):
        return redirect("/login/")
    if request.method == "POST":
        j=[]
        for i in range(11,54):
            j.append(str(i))
        patient_id=  request.POST.get('patient_id')
        session_id=int(request.POST.get('sessionid'))
        if(patient_id not in j):
            return HttpResponse("This patient ID does not exist in Experiment 2")
        querySet=Image.objects.filter(patientid=patient_id,sessionid=session_id).values()
        df = pd.DataFrame(querySet)
        columns=df['datatype']
        return render(request, 'TEST/fNIRsdatatype.html',
           |{'columns': columns,'patient_id':patient_id,'sessionid':session_id})
    return render(request,'TEST/experiment2.html')

def fNIRs(request):
    if not request.session.get('is_login', None):
        return redirect("/login/")
    if request.method == "POST":
        patient_id = request.POST.get('patient_id')
        datatype = request.POST.get('datatype')
        session_id = request.POST.get('session_id')
        querySet = Image.objects.filter(patientid=patient_id, sessionid=session_id,datatype=datatype).values()
        df = pd.DataFrame(list(querySet))
        image_data=df['data']
        return HttpResponse(image_data, content_type="image/png")
    return redirect("/index/")
```

Fig. 6. Functions from views.py responsible for retrieving fNIRS images generated from Experiment 2.

In turn, urls.py maps the URLs which will display the corresponding visualisation functions from views.py, as shown in Fig. 7.



```
from django.contrib import admin
from django.conf import settings
from django.urls import path, include
from django.conf.urls.static import static
from . import views
app_name = 'TEST'
urlpatterns = [
    path('', views.dataprocess),
    path('experiment1/',views.experiment1),
    path('experiment2/',views.experiment2),
    path('probedata/', views.probedata,name='probedata'),
    path('probedata/plot/', views.plot, name='plot'),
    path('fNIRsdata/plot/', views.fNIRs),
    ]
'
```

Fig. 7. The file urls.py, which maps URL paths from our web server to the corresponding data visualisation functions in views.py.

Table XIII summarises the data views which can be generated from our website:

TABLE XIII. AVAILABLE DJANGO DATA VIEWS

Available data views in Django		
Data source	**Input(s)**	**Output(s)**
Experiment 1	Patient ID, Stimuli data type, Channel data	Plots of MES probe readings for a specified motor stimulus
Experiment 2	Patient ID, Session ID, experiment data	Relative oxy (HbO) and deoxy (HbR) concentrations, haemodynamic frequency, time series plots of channel-specific data

E. *User Authentication*

The website created for the data mart use case also includes a login page. This was achieved by building another Django app called "login" and accessed via http://127.0.0.1:8000/login. User accounts are stored in the login_user table in PostgreSQL and mapped to a Django model called User; for our use case, one user account was entered manually into login_user. In a similar manner to how views.py retrieved the requested data for queries and visualisation in the "TEST" app, the "login" app compares login details entered by the user against the User model, and if the user-entered password entered the password for that user in the login_user PostgreSQL table, the user is redirected to index.html and prompted to another page where the user is able to configure parameters for the data visualisations.

IV. RESULTS

Overall, the staging process was successful in contributing to our project of building a flexible and scalable data vault, through the streamlined and automated processes for bulk ETL. In particular, the usage of DataFrames greatly increased the speed of ETL whilst keeping the integrity of the data, achieving the data vault goal of keeping data in their raw form.

The advantage of working purely in a Python environment for staging provided efficient workflows ease of adding features in bulk, such as timestamps. Furthermore, the ability to incorporate specific columns which were not contained in the raw data allowed us to flexibly meet the specific needs of different data, such as stimuli type in VMS data and patient identification in .dat files.

One major downside is that the usage of a standardised file structure introduced many null values and empty columns. This is because each data files were very different in their layout and therefore DataFrames had to accommodate all of the files, some of which were significantly different from each other.

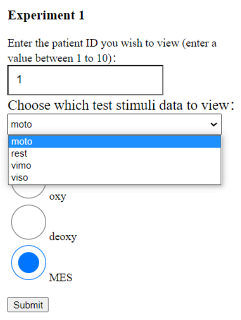
Another downside of the streamline process is the mass parsing of metadata. This meant that some satellites were bloated in size. However, it was ultimately decided that the querying of specific metadata values outweighed the memory consumption in order to meet the data vault goal of providing accessibility of data over speed.

The staging process pipeline could be more generalised as a further improvement to our implementation. That is, metadata pipelines could be further improved by making the metadata parsing parameters be less precise. As a result, each metadata satellite created will be less bloated as values are grouped together but can still be queried for comparison. We suggest that implementing this change would increase the speed of data queries, although at a cost of increased workloads prior to data presentation in the data mart layer.

The output of the SQL queries and web-based visualisations demonstrated data analytics and aggregation capabilities from our data vault; for example, we were able to create time series plots based on the user's selected channels for visualisation. Users are provided with the ability to compare specific optical channels so that analysing correlations between different channels is possible.

Python's mne package was also used to process and clean the fNIRS data, which was required before visualisations could be generated from that data. Further visualisation capabilities with fNIRS data could include the recreating the brain haemodynamic model.

Screenshots from the website are provided in the following figures to showcase a sample of the visualisation options available to the end user. Figs. 8-10 show the visualisation options and generated plot from experiment 1:



Experiment 1

Enter the patient ID you wish to view (enter a value between 1 to 10):

1

Choose which test stimuli data to view:

moto

moto
rest
vimo
viso

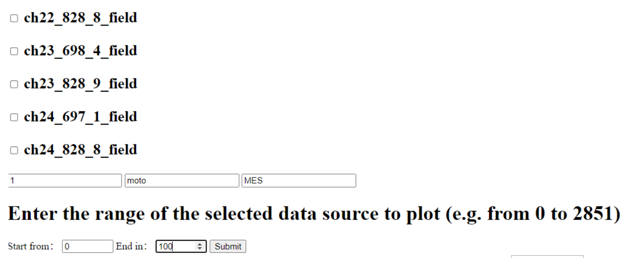
oxy

deoxy

MES

Submit

Fig. 8. Visualisation menu for Experiment 1 data, with options to choose which set of motor stimuli and oxy/deoxy data to query for plottting.



ch22_828_8_field

ch23_698_4_field

ch23_828_9_field

ch24_697_1_field

ch24_828_8_field

1 | moto | MES

Enter the range of the selected data source to plot (e.g. from 0 to 2851)

Start from: 0 End in: 100 Submit

Fig. 9. Visualisation options for Experiment 1 include selection of individual channels to plot.

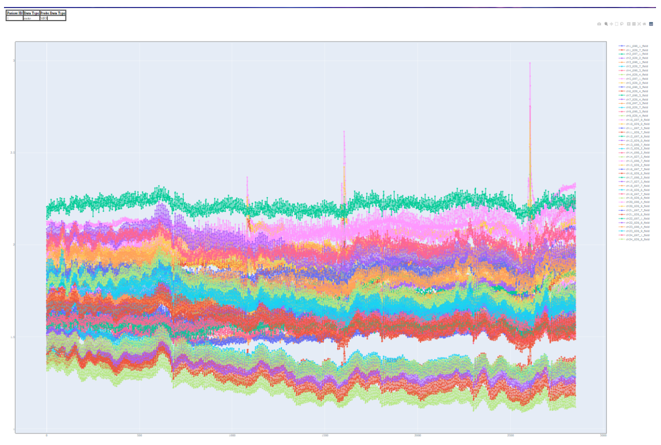
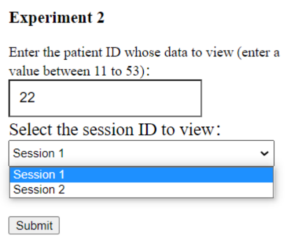


Fig. 10. A complete set of channel readings from patient 1's MES readings from their motor-only stimulus session in Experiment 1.

Figs. 11-12 show the visualisation options offered to the user for experiment 2 data, which are contingent on whether a particular set of channel data is available in the raw vault, e.g., raw data for generating response images were unavailable in some patients' data.



Experiment 2

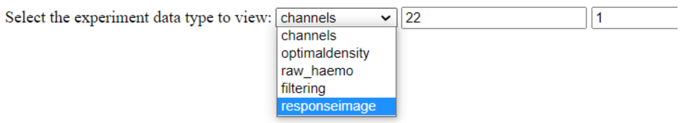
Enter the patient ID whose data to view (enter a value between 11 to 53):

22

Select the session ID to view:

Session 1

Fig. 11. Selection of patient ID and session ID to view from experiment 2.



Select the experiment data type to view: channels 22 1

Fig. 12. Available visualisation options for patient ID = 22, session 1. The available options in the drop-down menu is dependent.

Finally, Figs. 13 and 14 represent the most interesting images generated from Experiment 2's raw data:

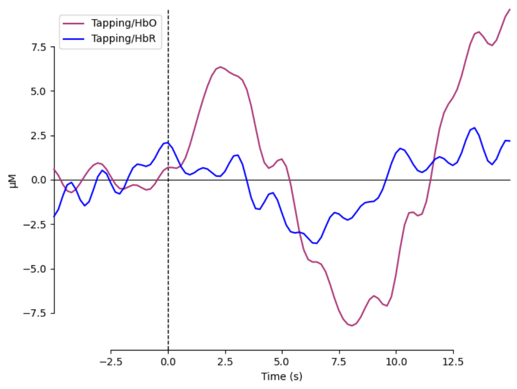


Fig. 13. Response image one patient's HbO and HbR readings to illustrate the relationship between the two signals.

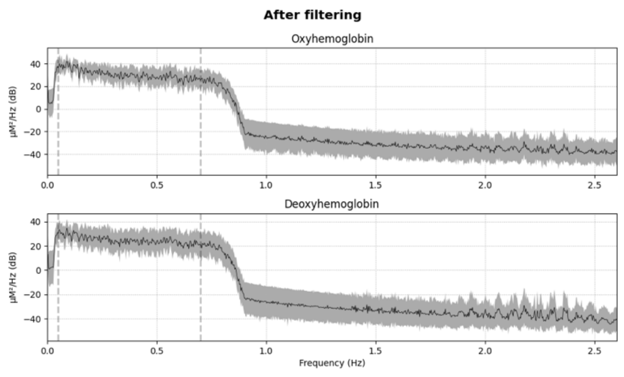


Fig. 14. Plot of haemodynamic frequency content after data cleansing using both high and low pass filters.

One particularly interesting finding from Fig. 14 reveals noise in the data in the form of the patient's heartbeat. This may be evidence that haemodynamic response activity occurs below 0.5 Hz, but activity above 1Hz can be seen in the signal. A low pass filter is used to remove this whilst a high pass filter is used to remove the slow drifts present in the data.

V. Discussion

Evaluating the project overall, one of the key areas that could be explored further is the staging area, which significantly influenced the transition of our group's conceptual schema to a physical implementation. Table XIV below shows an overall summary of the staging as it evolved through the project.

The desire to create a staging process that was flexible for modification, scalable for potentially wider project scope, and automated and streamlined processes to minimise user input worked in favour to achieve a data vault that prioritised accessibility and data integrity.

Unfortunately, the trade-off mentioned in the above staging section resulted in uniform DataFrames that were sparse in nature, thus greatly increasing storage usage and slowing down querying speeds.

A potential approach to fixing sparse DataFrames would be to make use of the modular design of the staging pipeline. In detail, additional sub-pipelines could be implemented so that the DataFrames could be converted to SciPy matrices which can be computationally identified as sparse. Using control flow programming, sparse SciPy matrices could undergo matrix operations to combine rows and columns, therefore significantly reducing the dimensionality when converting the matrices back to DataFrames. When the data is queried, inverse matrix operations could be carried out so that the original data is presented.

The matrix operation approach could have an impact on speed of queries due to the extra computational processes involved. However, a method to counteract this would be the splitting of DataFrames before matrix operations. Since metadata and data both contain features that vary in content, a DataFrame could be split into a sparse SciPy matrix and a dense SciPy matrix (done as best as possible). Therefore, any matrix operations would be carried out on the sparse matrix and greatly reduce the dimensions involved, because of the same operations are applied with significantly matrices.

The combination of DataFrame split and matrix operations could potentially improve the speed of querying via more optimised DataFrames. Considering one of the major drawbacks of the data vault is the overuse of JOINs which heavily impact querying speed, implementing this optimisation would be highly desirable for further improving our data vault.

TABLE XIV. Staging Self-Evaluation

Project Summary and Evaluation: Staging		
Major Design Choices	**Positive Effect(s)**	**Negative Effect(s)**
Usage of DataFrames as an intermediate data type for transforming and loading.	Greatly improved efficiency and data manipulation affinity.	Some files needed heavy parsing which required extra code.
Standardised pipeline to write DataFrames into PostgreSQL	Greatly improved efficiency in handling mass ETLs as well as able to handle a wide range of different data formats.	Everything is standardised, which may not be the most optimal data processing method, e.g. metadata.

Automated and streamlined process	Greatly improved efficiency in handling mass ETLs with minimum user input.	Errors in the data are carried over into the database tables and have to be fixed post-staging.
Modular design	Greatly improved efficiency in handling mass ETLs and ability to insert needed features that were not present in the raw data e.g. timestamp.	Scalable in terms of variety but not data amount, as a for loop was used. Therefore, the code implemented had to be quite complex to be able to be used by the rest of the pipeline.
Storing metadata also as DataFrames.	Greatly improve efficiency in handling mass ETLs as they are treated nearly the same as data. Specific values of the metadata could be queried therefore can handle changes to metadata.	Output DataFrames are sparse in nature and occupy more memory.

This section will discuss the problems and successes faced when creating the schema and the physical design of the data vault.

For our conceptual design in the schema, one of the problems we faced was deciding how to organise the different data sources arising from both experiments. Given that there are three data sources (fNIRS 1 from experiment 1 and fNIRS 2 and EEG from experiment 2), we tackled the organisational problem by creating a single data source hub and then a separate fNIRS hub which had fNIRS 1 and fNIRS 2 data in separate satellites. The same approach was included in our schema for EEG and satellite; however, it was not physically implemented. By organising the data sources in this way, we succeeded at creating a scalable model such that future data sources can be easily integrated as their own hub and satellite(s) into our data vault without major time-consuming changes that could affect the consistency of our data vault structure. The overall data vault structure will maintain its consistency and the existing data sources hubs (e.g., fNIRS and EEG hub and their satellites will not be affected or need change).

In our physical implementation, we had trouble deciding where to place different fNIRS files in our data vault. Initially, we thought to add all fNIRS data from experiment 2 in one single satellite. However, given the different file formats and attribute structures, this would have resulted in inconsistencies in the satellite and would be ineffective when running queries and extracting data. Therefore, we found a compromise and instead placed related fNIRS files into separate satellites, e.g. all the fNIRS .dat files from experiment 2's first session was added to one satellite, and all the .dat files from experiment 2's second session were added to another satellite. As a result, we achieved organising the files in a coherent and structured way, allowing for effective queries to be made using JOINs when needed. Such JOIN queries have been successful and are shown in Fig. 4 in the "“Queries” section of the report.

With our design and physical implementation approach, our data vault succeeded at running queries in hubs and between satellites, outputting relevant information as seen in the results section.

Moving forward, we would implement more link tables between related fNIRS satellites to allow for better connectivity in the data vault. Also, we would add the load end time attribute to all the satellites to help with identifying and trace changes in the vault. Finally, we would fully implement and test the EEG hub and satellite to see the full potential of the data vault and run queries using the information it stores and present data visualisations using our GUI.

Evaluating whether the data vault model is appropriate for medical images

To a large extent, our data vault model is appropriate for medical images given its ability to provide a streamlined and automated process to handle bulk ETL, handling heterogeneous data sources, and its flexible hub-link-satellite structure. This agility aligns with the main reason data vaults have been increasingly used for medical imaging; medical imaging data can be volatile and ingested by database systems in large amounts. Given that the data vault model is capable of handling business and research requirements, as well as large amounts of data at scale, it is arguable an appropriate model for medical images. This conclusion has been supported and is evidenced by the growing number of neuro-vaults created and the literature discussed in Section II.

However, although we have highlighted areas of scientific research which have benefitted from data vaults, the data vault model is not appropriate for all contexts. This is because in systems with few data sources, data of subjects not prone to change or only limited changes, the data vault structure contains more complexity that may offset any added value.

VI. Conclusions

In conclusion, we succeeded at fully implementing the data warehouse layer, including building and populating a medical imaging data vault. The data vault has been created in a scalable way to hold multimodal medical imaging data from multiple sources given the streamlined and automated process to handle bulk ETL. The schema structure with hubs, satellites and links enables data to be populated and for seamless retrieval as seen from our queries. Although the EEG hub and satellite were not physically implemented in the data vault due to time constraints and prioritisation, they were included in the overall conceptual schema. Because of this decision, some files such as the EEG .mat files were not included in the staging process and data vault, yet the loading and retrieval from the data vault from all the included data is time-efficient. Additionally, the information layer was set up through a Web-based GUI with a simple authentication use case, allowing data analytics and visualisation to be conducted with a few clicks to present plots and diagrams from experiment 1 and experiment 2 various patients and experimental sessions.

Overall, our data vault was able to meet the objectives we set out to accomplish, namely being flexible, scalable, and an efficient model that proved to be appropriate for medical imaging scientific data. To progress from what we achieved, incorporating the EEG data into our data vault would enable us to carry out the staging process for the .mat files, realise the full potential of the data vault, and provide opportunities to explore further visualisations and data analytics.

Acknowledgments

The authors would like to thank our module lead Dr. Felipe Orihuela-Espina for providing additional meetings and seminars to advise and clarify issues encountered during the development of our solutions for the semester project.

References

[1] M. Allanic, P-Y. Hervé, C-C. Pham, M. Lekkal, A. Durupt, T. Brial, A. Grioche, N. Matta, P. Boutinaud, B. Eynard, and M. Joliot, "BIOMIST: A Platform for Biomedical Data Lifecycle Management of Neuroimaging Cohorts," Front. ICT, vol. 3, art. 35, pp. 1-19, January 2017, doi: 10.3389/fict.2016.00035.

[2] I. Bojičić; Z. Marjanović; N. Turajlić; M. Petrović; M. Vučković, Vladan Jovanović, "A comparative analysis of data warehouse data models," 2016 6th International Conference on Computers Communications and Control, 2016, pp. 151-159, doi:10.1109/ICCCC.2016.7496754.

[3] W. H. H Inmon and D. Linstedt, Data Architecture: a Primer for the Data Scientist : Big Data, Data Warehouse and Data Vault, Elsevier Science & Technology, 2014. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/bham/detail.action?docID=1875436

[4] M. Ivanova, M. Kersten, S. Manegold, and Y. Kargin, "Data vaults: database technology for scientific file repositories," Computng in Science & Engineering,, vol. 15, iss. 3, pp. 32-42, May/June 2013, doi: 10.1109/MCSE.2013.17.

[5] D. Linstedt and M. Olschimke, Building a Scalable Data Warehouse with Data Vault 2.0. www.sciencedirect.com, 2016. (https://www.sciencedirect.com/science/article/pii/B9780128025109000118) (accessed Nov. 29, 2021).

[6] D. Linstedt and M. Olschimke, Building a scalable Data Warehouse with Data Vault 2.0 : implementation guide for Microsoft SQL Server 2014. Waltham, Ma Morgan Kaufmann Elsevier, 2016, pp. 1–15.

[7] N. Marklund et al, "Monitoring of β-Amyloid Dynamics after Human Traumatic Brain Injury," J. Neurotrauma, vol. 31, (1), pp. 42-55, 2014. Available: https://www.proquest.com/scholarly-journals/monitoring-beta-amyloid-dynamics-after-human/docview/1477523640/se-2?accountid=8630, doi: 10.1089/neu.2013.2964.

[8] C. Miller, Hands-On Data Analysis with NumPy and Pandas : Implement Python Packages from Data Manipulation to Processing. Birmingham: Packt Publishing, Limited; 2018.

[9] "Making queries," date unknown. Accessed on: Dec. 9, 2021. Available: "Writing Views," date unknown. Accessed on: Dec. 9, 2021. Available: https://docs.djangoproject.com/en/3.2/topics/db/queries/

[10] "Models," date unknown. Accessed on: Dec. 9, 2021. Available:: https://docs.djangoproject.com/en/3.2/topics/db/models/

[11] N. Naseer and K-S Hong, "fNIRS-based brain-computer interfaces: a review," Front. Hum. Neurosci. 9:3., Jan. 2015, doi: 10.3389/fnhum.2015.00003.

[12] F. Orihuela-Espina, D.R.Leff, D. R. C. James, A. W. Darzi, G. Z. Yang, "Quality control and assurance in functional near infrared spectroscopy (fNIRS) experimentation," Physics in Medicine and Biology, 55(13):3701, Jun. 2010, doi: 10.1088/0031-9155/55/13/009.

[13] D. Petersohn et al., "Towards scalable dataframe systems," Proceedings of the VLDB Endowment, vol. 13, no. 12, pp. 2033–2046, August 2020, doi: 10.14778/3407790.3407807.

[14] PR Newswire. Built by Data Scientists for Data Teams, Zuar's Mitto 2.8 Expands the Boundaries of ETL. PR Newswire US [Internet]. 2020 Dec 10 [cited 2021 Dec 2]; Available from: https://search-ebscohost-com.ezproxye.bham.ac.uk/login.aspx?direct=true&db=bwh&AN=202012100400PR.NEWS.USPR.UN18627&site=ehost-live

[15] M. Ullsperger, S Debener, "Simultaneous EEG and FMRI : Recording, Analysis, and Application [Internet]". Oxford: Oxford University Press; 2010 [cited 2021 Dec 6]. Available from: https://search-ebscohost-com.ezproxyd.bham.ac.uk/login.aspx?direct=true&db=nlebk&AN=311724&site=ehost-live

[16] J-P. Weiß, U Hübner, J. Rauch, J. Hüsers, F. Teuteberg, M. Esdar, and J-D. Leibe, "Implementing a data management platform for longitudinal health research," in German Medical Data Sciences: Visions and Bridges, pp. 85-89, September 2017, doi: 10.3233/978-1-61499-808-2-85.

[17] G. S. Wilson and A. C. Michael, Compendium of In Vivo Monitoring in Real-Time Molecular Neuroscience," World Scientific, 2019.

[18] "Writing Views," date unknown. Accessed on: Dec. 9, 2021. Available: https://docs.djangoproject.com/en/3.2/topics/http/views/

[19] K. Zeng, J. Yan, Y. Wang, A. Sik, G. Ouyang, X. Li, "Automatic detection of absence seizures with compressive sensing EEG," Neurocomputing, 171:497-502, Jan. 2016, doi: 10.1016/j.neucom.2015.06.076.

[20] I. Zubarev and L. Parkkonen, "Evidence for a general performance-monitoring system in the human brain," Human Brain Mapping, vol. 39, no. 11, pp. 4322–4333, Jul. 2018, doi: 10.1002/hbm.24273.