Reusing Dynamic Data Marts for Query Management in an On-Demand ETL Architecture

SUZANNE MCCARTHY

B.A., M.Sc., H.Dip., Grad.Dip.

A Dissertation submitted in fulfilment of the
requirements for the award of
Doctor of Philosophy (Ph.D.)

to

Dublin City University

Faculty of Engineering and Computing, School of Computing

Supervisors: Mark Roantree and Andrew McCarren

August 2020

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:

ID No.: 16213427

Date: 28th August 2020

List of Publications

1. Andrew McCarren, Suzanne McCarthy, Conor O. Sullivan, Mark Roantree: Anomaly detection in agri warehouse construction. ACSW 2017: 17:1-17:10

2. Suzanne McCarthy, Andrew McCarren, Mark Roantree: Combining Web and Enterprise Data for Lightweight Data Mart Construction. DEXA (2) 2018: 138-146

3. Michael Scriney, Suzanne McCarthy, Andrew McCarren, Paolo Cappellari, Mark Roantree: Automating Data Mart Construction from Semi-Structured Data Sources. The Computer Journal. June 2018.

4. Suzanne McCarthy, Andrew McCarren, Mark Roantree: A Method for Automated Transformation and Validation of Online Datasets. EDOC 2019

5. Suzanne McCarthy, Michael Scriney, Andrew McCarren, Mark Roantree: On-Demand ETL with Heterogeneous Multi-Source Data Cubes. In submission to PAKDD 2021

Acknowledgements

Firstly, I would like to thank the Insight Centre for Data Analytics, Science Foundation Ireland, Dublin City University School of Computing and Kepak Group.

I would also like to thank my supervisors Prof. Mark Roantree and Dr. Andrew McCarren for their hard work and support during this process. Both have been a constant source of encouragement and guidance.

I would not have succeeded in this Ph.D without the support of my colleagues and friends: Robin, Fouad, Congcong and Gillian. A special note of thanks to Dr. Michael Scriney for his collaboration and for always having a word of encouragement in the tougher times.

Finally, thank you to my parents and brother and to John, Dorrian, Kelsey and Emanuele for their love and support.

Abstract

SUZANNE MCCARTHY

Reusing Dynamic Data Marts for Query Management in an On-Demand ETL Architecture

Data analysts working often have a requirement to integrate an in-house data warehouse with external datasets, especially web-based datasets. Doing so can give them important insights into their performance when compared with competitors, their industry in general on a global scale, and make predictions as to sales, providing important decision support services. The quality of these insights depends on the quality of the data imported into the analysis dataset. There is a wealth of data freely available from government sources online but little unity between data sources, leading to a requirement for a data processing layer wherein various types of quality issues and heterogeneities can be resolved. Traditionally, this is achieved with an Extract-Transform-Load (ETL) series of processes which are performed on all of the available data, in advance, in a batch process typically run outside of business hours. While this is recognized as a powerful knowledge-based support, it is very expensive to build and maintain, and is very costly to update, in the event that new data sources become available. On-demand ETL offers a solution in that data is only acquired when needed and new sources can be added as they come online. However, this form of dynamic ETL is very difficult to deliver. In this research dissertation, we explore the possibilities of creating dynamic data marts which can be created using non-warehouse data to support the inclusion of new sources. We then examine how these dynamic structures can be used for query fulfillment and how they can support an overall on-demand query mechanism. At each step of the research and development, we employ a robust validation using a real-world data warehouse from the agricultural domain with selected Agri web sources to test the dynamic elements of the proposed architecture.

Table of Contents

List of Figures

Preface

In this dissertation, an architecture is introduced that can support On-Demand ETL and Query Reuse, using a set of dynamic data marts. In Chapter 1, we present an overview of data warehousing and schemas, data cubes and ETL. The limitations of the traditional approaches to these concepts when dealing with multiple, heterogeneous data sources and frequently changing data were highlighted and this motivated our research questions and hypothesis. In Chapter 2 we present the state of the art across the multiple areas of research in which this thesis is based. In particular, we present modern approaches to ETL and query reuse, before discussing the current literature on On-Demand ETL. In Chapter 3, we present our extended ETL architecture, our Data Model, a number of our data sources and our case studies. Our Common Data Model has been applied to a number of collaborative projects [64,91].

In Chapter 4, we show how the architecture introduced in Chapter 3 is used to present a set of dynamic data cubes, from importing data from an unseen source to loading. This methodology was published in [68], as well as an optimisation which was published in [69]. In Chapter 5, we present our approach to reusing and extending these dynamic data cubes in response to user queries by capturing two of our main constructs - CubeMaps and QueryMaps - which allows for an easy comparison of the two to identify full matches and partial matches. In Chapter 6, we show the methodology needed to fulfil a query using an existing cube, extended with data from outside the set of cubes. In Chapter 7, we show our various stages of evaluating this work.

Chapter 1

Introduction

This dissertation is motivated by the need for Business Intelligence that combines enterprise and web data in a way that is lightweight and flexible. Our goal is to provide the end user with queries fulfilled by newly imported data in an on-demand fashion. We will show how traditional methods of processing data are insufficient for the needs of data analysts, and provide a solution to the problem of expensive, time-consuming traditional data warehousing. This opening chapter presents the background to our research, defines the problem statement and articulates the research questions that will be addressed. In Sections 1.1 and 1.2, we provide a background to the areas of data warehousing and ETL, respectively. In Section 1.3, we present a discussion on web data which provides an important context to our research. We will show how traditional practices are insufficient for the current needs of today's data user and then present the hypothesis and aims of this research in Section 1.4. The major contributions of this research are presented in Section 1.5, before concluding with the structure of this dissertation in Section 1.6.

1.1 Data Warehousing

A *Database* is a data repository for the general purpose of data storage and management; a *Data Warehouse* is a particular type of database in which the entities

and relationships are designed for the purpose of the analysis of data. The exact definition of a data warehouse, depending on the author, may be focused on the innate attributes of the data warehouse, or its purpose.

The author of [42] specified the following attributes of a data warehouse:

- **Subject-oriented**, or domain-specific, meaning that a warehouse should provide topical information about a specific industry, as opposed to the day-to-day operations of a company.
- **Integrated** data from multiple sources.
- **Time-variant**, where the data is associated with a particular period of time.
- **Non-volatile**, or read-only, meaning data should not be removed from a warehouse.

Some of these attributes are flexible in terms of the reality of working with a data warehouse. For example, some users may choose to allow their data warehouse to be volatile, meaning data may be removed when it is old or found to be inaccurate.

The author of [54] instead defines a data warehouse as a copy of transaction data specifically structured for query and analysis. A data warehouse is designed with the industry and needs of the enterprise in mind. The goal for data warehouse designers and ETL architects is to provide users with the most accurate possible data in the most timely possible fashion. Early influential authors in data warehousing technology espoused different philosophies as to how the technology should be employed to address business requirements. Inmon [42] is known for a top-down approach wherein the developer would start with a fully specified overall data warehouse that spans the enterprise as a whole, from which data marts (i.e. single-source data warehouses or subsets of a data warehouse) for department-level analysis will emerge. A different approach to this is presented by Kimball [54], who instead recommended beginning with data marts designed to fulfil specific business needs at a lower level, followed by integrating these data marts using ETL processes, which results in an overall data warehouse.

1.1.1 Warehouse Schemas and Data Cubes

A *data cube* is a subsection of a data warehouse that (a) conforms to the data warehouse's *data model*, and (b) represents a specific *fact* of value to the business, such as their total sales analysed by day, location and product. Data Modelling is the process of designing a schema to represent the entities in the data and relationships between them. Depending on the needs of the enterprise, the schema of the data cube may be represented as a star, a snowflake or a constellation. The difference between these three schemas relates to the configuration of the entities - the facts and dimensions. Facts are the data metrics of interest to the analyst, e.g. sales, price, production levels. Dimensions are the axes along which this data might be viewed, providing context for the metric, e.g. date, location, products. In business informatics, researchers and practitioners often refer to cubes as *data marts*. In this thesis, we will interchange these terms depending on our context. Cubes are more often used in technical descriptions whereas the term data mart will be used in case studies or dealing with business requirements.

A data warehouse may employ a multi-dimensional model represented as a *star schema* [32], named as such because of its resemblance to a star when illustrated. A star schema model consists of a single fact table with a one-to-many relationship to any number of dimension tables.

In Figure 1.1, a data mart has objects labelled **Product**, **Salesperson**, **Date** and **Customer**, representing *dimensions* while **Order_facts** represents the *fact* with foreign key links to the four dimensions. In this case, the **Order_facts** fact has measures **quantity_ordered**, **order_dollars** and **cost_dollars**. This makes it very easy to find out, for instance, the total quantity of each product ordered on a certain day. This data mart is a *cube* of data that addresses a requirement to analyse sales by product, by day and/or by location.

Another schema that may be used to represent a data cube is a snowflake. A snowflake schema is used when the dimensions are hierarchical in nature. The dimension can then be normalised, i.e. split, into multiple tables to avoid repeating

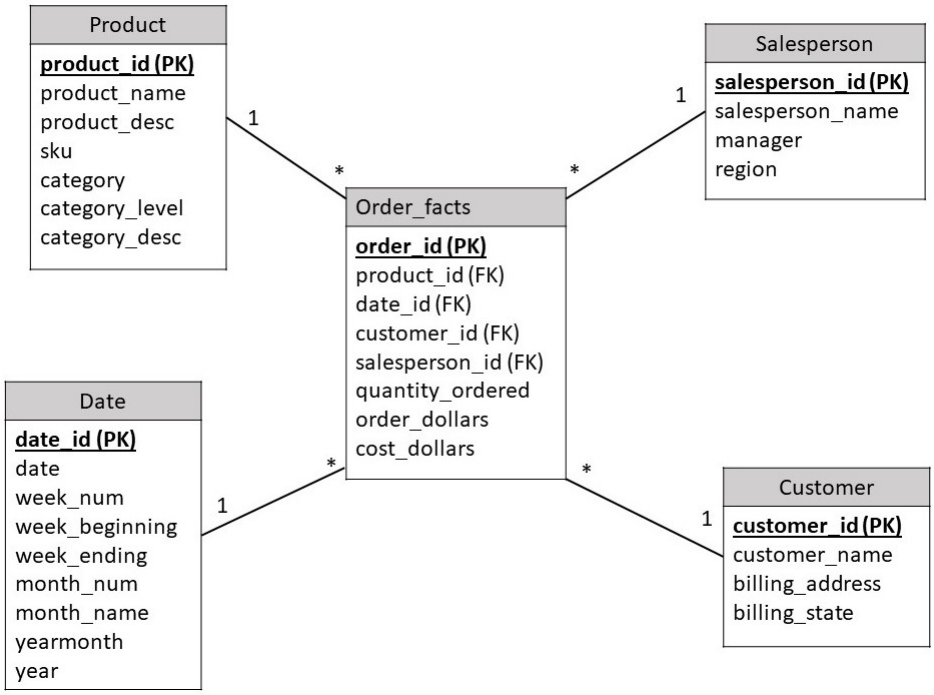

Figure 1.1: Sales Star Schema

values [21]. For example, the **Product** dimension in Figure 1.1 has several attributes, which could lead to the dimension becoming unwieldy if the company has a large number of products. In Figure 1.2, the product dimension is normalised and now has degrees of granularity by which it can be queried - **Category** and **Product** - as well as the **Date** dimension now being normalised.

Increasing the level of normalisation in this manner has the benefit of avoiding duplicated data and maintaining data integrity, but has the disadvantage of possibly making querying the data more complex as there are more joins required to use all the tables [72].

When there are multiple star schemas with different facts but shared dimensions, this is known as a constellation (or sometimes a galaxy) schema. In a constellation schema, the data warehouse will contain a set of conformed dimensions, which are shared by *multiple* facts. For example in Figure 1.3, the **Order_facts** and **Purchase_facts** fact tables share the **Date** and **Product** dimensions.

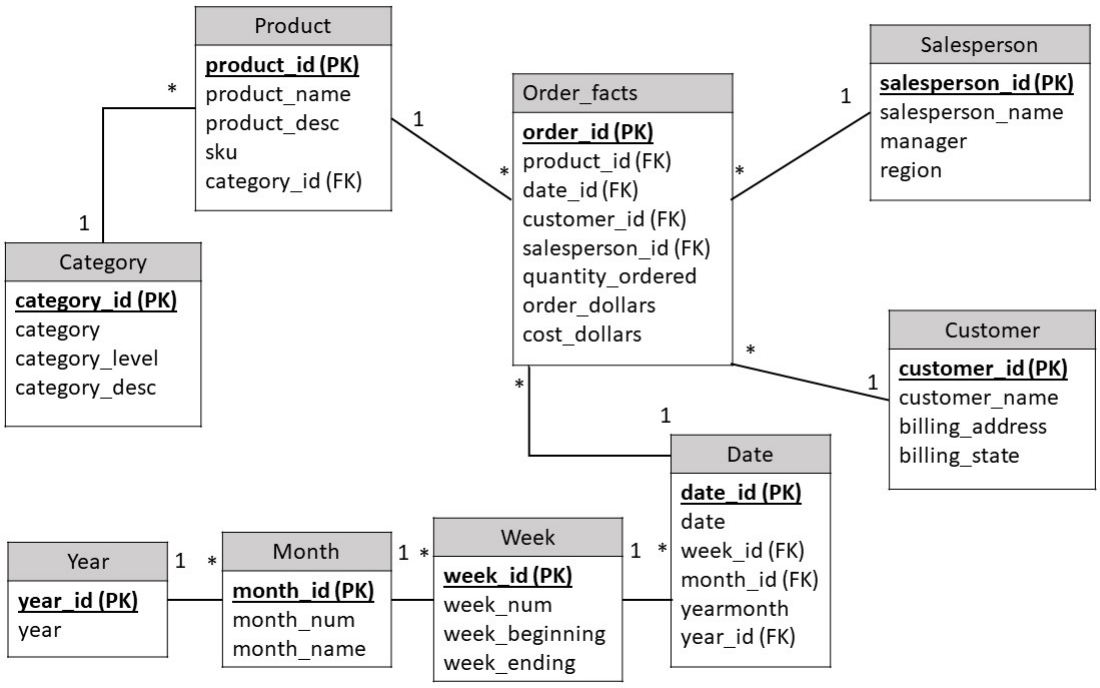

Figure 1.2: Example Snowflake Schema

It lacks the advantage over the snowflake schema in that dimension tables may still be large and not benefit from a higher degree of normalisation. However, it is a more sophisticated design and more likely to be used in bigger organisations that have several metrics that share dimensions to be used for analysis, for example, needing to produce a report that tracks the amount of a product that has been ordered as well as the amount that has been sold.

1.2 Extract, Transform and Load

Data in its raw state is rarely suitable for loading to a data warehouse or for use in a prediction algorithm. Additionally, datasets from multiple sources often have areas of structural and semantic heterogeneity which need to be resolved. Extract-Transform-Load (ETL) as a framework for the integration of datasets from disparate sources gained popularity in the 1970s and quickly became the standard. ETL is the process of migrating datasets from a source to a format by which it can be

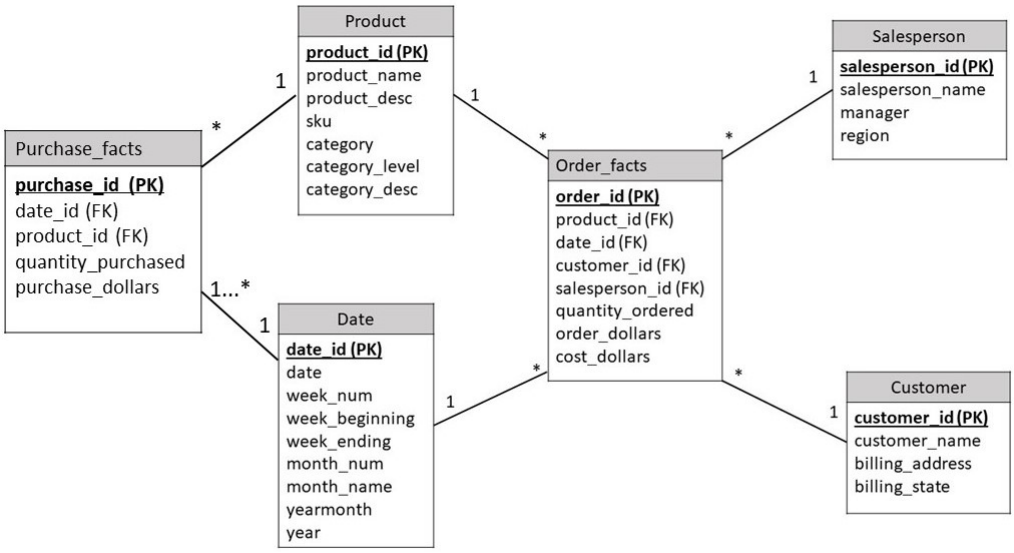

Figure 1.3: Example Constellation Schema

queried and analysed [54]. The aim is to provide the user with a *global* schema of multiple, heterogeneous data sources to reduce the need for manual work in Business Intelligence. The ETL process addresses this need to integrate data that comes from different sources but which needs to be viewed and analysed together.

A typical ETL workflow is seen in Figure 1.4 where data is extracted from both enterprise and possibly some external sources to a staging area, where a series of transformation functions is performed on the data. The data is then loaded to a warehouse, where it is ready for queries. In the architecture shown in this figure, Kimball's [51] bottom-up approach to data warehouses is shown, where the data is first loaded to data marts, which are then combined to form the data warehouse. The reverse approach is top-down, where the data is first loaded to the warehouse, which then produces data marts. The individual sub-processes involved in ETL are now outlined. Each may involve several stages and are usually customised to the datasets to which they will be applied. At each stage, the challenges involved will be explained as well as what must be validated at each step.

Extract. At the extraction stage, data is imported into the system from its sources.

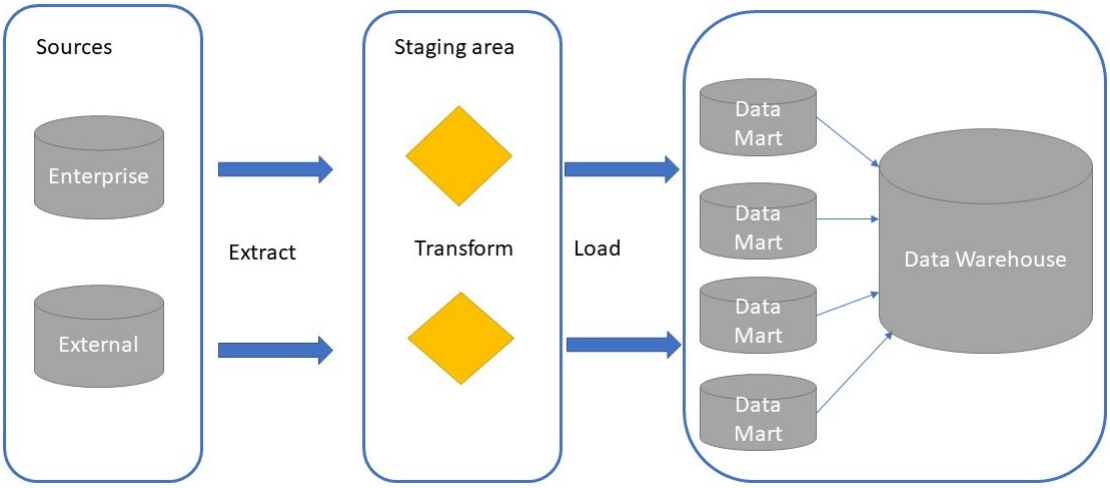

Figure 1.4: ETL Architecture

This may include a data import from a relational database, XML, JSON or flat files. Most data analytics undertakings require that data from more than one source be combined in the final data warehouse. Sources will have been curated by identifying the data providers that will support the decision-making process used by the end users. In [51], Kimball describes the planning stage of the extraction process, which involves designing a *logical data map*, i.e. documentation of the extraction process, the metadata of the data sources and their relationships to the final target data warehouse tables. The challenge at the extraction stage is to ensure that the data has not been inadvertently altered, and that the correct data has been acquired from the source. It must be validated to check the imported data matches the data in its source. The data is typically stored in a staging area before it undergoes transformation.

Transform. The purpose of transformation is to prepare the data for loading and subsequent use in some analysis and/or reports. By convention, the data that is loaded to a data warehouse is always transformed to conform to a schema, so as to preserve the integrity of the data warehouse, which will have been built to conform with the schema model.

Data transformation will involve a set of functions that are enacted on the data,

with the goal of creating a clean, uniform dataset from the multiple, disparate sets. These functions will be either selected or created by the ETL workflow designer. They may be a standard set of transformation functions or need to be tailored specifically to the dataset on which they will be performed, commonly a mixture of both.

Such functions may include (but are not limited to):

- The units in which the measures are expressed may be converted, e.g. Tonnes to KG. Therefore some calculation will need to be done on all measure values.
- Terms may need to be mapped from their native format to the target format. This is done because different sources may use different terms for the same concept. For example, if one source uses the term "U.K.", while another uses "Utd. Kingdom" and yet another uses "United Kingdom", these must be mapped to a single term.
- Constraints may be applied such as dropping Nulls.
- Roll-ups or drill-downs may be applied. For instance, if two sources are to be combined and one source uses weekly publishing and the other uses monthly, the weekly source may be rolled up into months.
- New measures may be derived from existing ones, such as calculating net profit from gross profit.
- Coding or de-coding values, e.g. if a source uses codes for a set of values: "I" for "Import".

Data cleansing is the process of removing unwanted data from a dataset, as opposed to changing the data that is kept. This may be done, for example, to remove duplicate data, or if there are Nulls in the data and the user constraints are that this data must be dropped. The Transformation process may involve tools such as lookup tables or transformation rule sets. The validation of the transformation

process is a complex one. The transformations will generally be documented and the source-target mappings can be examined post-hoc [51].

Load. Finally at the loading stage, the warehouse is populated with physical data. The fact tables will be populated with the measure values and foreign key links to the dimension tables. This should be a seamless process as the attributes of the data should now match the attributes of the facts and dimensions of the target tables as a result of the transformation.

1.2.1 Traditional ETL Approaches

The ETL processes are traditionally done upfront to all imported data, and results in the building, loading and reloading of a data warehouse. Because of the time and processing power involved in this, it is often completed outside of business hours as an overnight batch process. This is a problem for businesses that require their data to be as close as possible to real-time [100, 114].

The authors of [106] categorise the outstanding problems with traditional ETL, according to the specific sub-process in which each problem is found. For example, extracting from multiple sources leads to data heterogeneity, extensive data transformation is responsible for the bulk of the manual overheads, while loading has the most issues with performance.

There are commercial ETL tools with a GUI available, such as *Pendaho Data Integration*, *Talend Open Studio* and *Informatica Power Centre*. Their main tasks usually include (a) the identification of relevant information at the source side; (b) the extraction of this information; (c) the customisation and integration of the information coming from multiple sources into a common format; (d) the cleaning of the resulting data set, on the basis of database and business rules, and (e) the propagation of the data to the data warehouse and/or data marts [102]. These and other tools were evaluated [112] using several criteria, primarily to do with elements of the user's experience. The criteria did not include addressing existing challenges within the ETL process such as data quality or automation. These tools tend to

address only the representation of ETL processes; identifying the transformations, mappings and rules required still needs to be done manually [95].

1.3 Challenges with Web Data

Data analysts often have a need to integrate their in-house data warehouse with external datasets to view their reports. In particular, they may need to import web-based datasets into their systems in order to compare their data with the international standard. The quality and accuracy of their reports - whether descriptive or predictive - is reliant on the quality of the data being used as inputs [113]. There is no shortage of data published from web-based government sources, made freely available. But there is a lack of unity between these sources, as there is often a lack of a unified, agreed-upon model or set of data quality standards with which to publish the data [31], leading to issues with data quality and data heterogeneity. Although we expect to be able to handle a wide variety of data formats from our source providers, we make special mention of the challenges associated with web-based datasets because a large number of our sources are web-based. However, the focus of this research is not that of semantic web challenges as the aim is not to read or mine the data while it is stored on the web, but of implementing an ETL solution that is generic enough to address the disparity between sources.

The authors in [81] classified the possible heterogeneities in the following way:

- **Structural conflicts** refer to any kind of inconsistencies between the underlying schema of the datasets such as mismatch between data types, constraint mismatches, naming conflicts, aggregation conflicts or missing or duplicated items.
- **Domain conflicts** may include conflicts between the scale or units between datasets.
- **Data conflicts** refer to such issues as missing data, synonyms or spelling errors.

Web-based data may contain all of these types of heterogeneity, leading to a need to resolve them using the ETL processes. Additionally, data quality issues in web-based data can be caused by inadequate handling of the data during the publishing process. For example, there is often a requirement for speed in publishing web-based data as soon as it becomes available, leading to low quality data as the time taken to quality-check data would be considered an unacceptable delay [79]. In its raw format, the data is not considered "fit for use" [105], further leading to large manual efforts, increasing expense and possible unreliable reporting.

1.4 Problem Statement and Hypothesis

Thus far, we have laid out the technical background to our research and described the issues with the data on which our case studies will be based. We are now in a position to provide the motivation for our work, identify a key problem which we aim to solve, the challenges in solving it, and how to add to the body of research in our field.

1.4.1 Problem Statement

Due to the issues highlighted above such as data heterogeneity, the problem of efficiently integrating multiple data sources to fulfil a query remains an open problem in academia [9, 17, 19]. Traditional ETL is done as a batch process and is heavy on resources - money, time, personnel and often requires heavy input from a domain expert. This results in a bulky data warehouse and slow queries, as well as high amounts of redundant data and repeated work. Furthermore, the data needs of the users frequently change, as do the structure and/or vocabulary of the data sources. Traditional ETL is not equipped to be dynamic; with each of these changes comes a renewed need for heavy input of resources.

Data sources do not remain static, nor do user needs. A traditional data warehouse is not built for flexibility and is not sufficient for a data environment where new sources

will frequently need to be imported or when data sources will make significant changes to the way they publish data. Additionally, the way that queries are run on traditional data warehouses assumes a static set of data sources that are loaded in bulk to the warehouse. When the data sources are frequently changing, this type of query processing cannot require data outside of a warehouse environment. When a query requires the joining of datasets stored in an existing data mart with data from outside of that environment, this is done by performing ETL processes on all data outside the warehouse. However, this is time-consuming, costly and repetitive work done on a large amount of data when only a small amount may be needed. These traditional methods are not suitable given the problem of non-static data sources.

In order to satisfy the needs of modern data analysts, we conclude that we need to produce an on-demand ETL approach which can import previously unseen data sources, which can fulfil queries without having to launch the query on a traditional data warehouse, and which avoids having to run ETL processes on data until the data is needed to fulfil a query.

1.4.2 Research Questions

Research Question 1: Dynamic Data Marts.
In order to produce an on-demand ETL solution, our process must run on an ETL architecture that reflects the same basic architecture as traditional ETL but has additional features or components so it is not necessary to make changes every time a new data source must be added (or an existing one makes significant changes). The challenge is to produce an ETL architecture that can produce dynamic data marts, by which we mean a data mart system that can import previously unseen sources as seamlessly as those from sources used regularly. Hence our first research question: What is the architecture required to provide dynamic data marts?

Research Question 2: Query Reuse.
Assuming that we can achieve our first goal of a *dynamic* ETL system, the attention will turn to how this system will handle queries. Again, we wish to improve on the

traditional approach of querying a bulky data warehouse when we only need a small amount of data to address the query. Query reuse is a well-established area of research, but it traditionally is used in the context of a data warehouse from which data marts are launched [12], while we don't use a data warehouse. Our second research question is: Can we implement a query reuse system on a set of dynamic data marts?

Research Question 3: On-Demand ETL
Given a dynamic ETL architecture which incorporates query reuse, our third research question is: Can we deliver on-demand ETL to fulfil queries with data not currently residing in previously answered queries?

Hypothesis
Given the three research questions above, we now formulate our hypothesis as follows: We can deliver on-demand ETL using previously unseen data sources if we integrate a dynamic ETL architecture with a cube matching methodology and a caching method for external data.

1.4.3 Solution Methodology

Our approach to address each of the research questions and finally answer the hypothesis can be briefly summarised as follows:

1. To address the first research question, for a system to create data marts from unseen sources, we require a number of extensions to conventional ETL approaches. We will present an architecture with key components that allow the ETL processes to take place in a way that will remain robust in the event of data sources changing. We will demonstrate this methodology by creating a set of data cubes that can be used for evaluating our case studies. Our experiments for this will then evaluate the accuracy of our methodology with a query-based validation, as well as a specific focus on the automation of the process with a comparison against a ground-truth version.

2. Given the creation of dynamic data marts, the next research question requires that these data marts can be reused to fulfil incoming queries. It is a reasonable assumption that a case of an existing mart being a perfect match for a new query will not be a common occurrence. Therefore, we will use existing methods from query reuse approaches combined with our novel constructs to acquire the metadata of queries and identify a relationship between queries. In order to facilitate the next research question, we have a process to identify the elements of the query that are not fulfilled by the existing set of cubes. We will demonstrate this methodology by showing a demonstration of the outcome of key stages for a single query. The experiments will then test the extent to which a number of queries can be fulfilled using query reuse.

3. The final step is to address the final research question which is for a method to extract external data in response to a query which requires data outside the set of existing data cubes for fulfilment. In order to produce an on-demand ETL solution, we will show our method of selecting the optimal data sources from a large data environment. We will show this method is used using the gaps in the query from the query-matching process. It may be the case that there are multiple paths to solving a query, requiring strategies for prioritising matches and integrating partial matches

4. Our final evaluation will use a combined approach to query reuse with on-demand ETL with a set of case studies to investigate the runtime and effectiveness of this work.

A more detailed methodology overview will be provided in Chapter 3.

1.5 Contributions

Thus far, we have presented our technical background, problem statement and hypothesis. We now turn our attention to our solution. We are aiming for a solution that facilitates on-demand ETL - an architecture for importing previously unseen

data sources into dynamic data cubes, along with a methodology for reusing the cubes that fulfilled previous queries to answer new incoming queries, and a way to combine the cubes with new data.

Our main contributions are as follows:
We firstly present our dynamic ETL architecture, which uses metadata descriptions stored as **Import Templates** and **DataMaps**, to produce a set of dynamic data marts. Following this, we show our approach to query reuse - how we use a rich metadata capture of the data contained in a data cube, called a **CubeMap**; an identically structured metadata description of a data query, called a **QueryMap**; and a methodology for matching CubeMaps with QueryMaps, which relies on an additional structure called the **Cube Matrix**. Finally, we present a working on-demand ETL system implemented using a real-world Agri enterprise warehouse and a selected set of Agri-specific web sources, which are frequently updating.

1.6 Summary and Thesis Structure

It can be seen that traditional ETL processes are not sufficient for the needs of modern data analysts and companies whose data needs are for fast, flexible solutions. In this chapter, we have aimed to provide the background and motivation for our work as well as presented our working hypothesis. Following this in Chapter 2, we will present the current state of the art in our area. In Chapter 3, we will present our overall architecture and a case study demonstrating this along with details of our data sources. In Chapter 4, we present our dynamic approach to ETL which underpins our proposed solution. In Chapter 5, we present our methodology of capturing data cube metadata as a CubeMap and matching data cubes with queries. This is followed by our final On-Demand ETL methodology in Chapter 6. In Chapter 7 we present our evaluation of this work. Finally in Chapter 8, we draw our conclusions including the limitations of our work and opportunities for future work.

Chapter 2

Related Research

The goal of this research is to develop an innovative on-demand Extract Transform Load (ETL) architecture. This requires the delivery of a query processor which fulfils queries using both the data warehouse and sources external to the data warehouse. In addition, our view of an on-demand system is one which facilitates the integration of new or unseen data sources in a flexible manner. As such, this research crosses a number of innovative boundaries which will require a discussion on the state of the art in the areas of virtual warehouses or dynamic cubes, query reuse where only cubes have been materialised, and mechanisms for query fulfilment which extend the borders of traditional warehouse queries. This literature review is structured as follows: in Section 2.1, we focus on data warehouse research where web-based data sources are employed and examine the state of the art in dynamic cube construction; in Section 2.2, we present our critique on query reuse and how it differs in our approach; in Section 2.3, we examine modern and innovative ETL approaches, and finally in Section 2.4, we provide a critique on current on-demand query mechanisms for data warehouses. In a summary in Section 2.5, we highlight the research gaps where we can deliver innovation in our research.

2.1 Dynamic Data Cubes

In this section, we highlight the state of the art in building a data warehouse with web-based data as this is the primary source of dynamic cube construction. The incorporation of web-based data into data warehouses is becoming increasingly common as analysts have a need to compare their in-house data with international datasets. This has led to advances in programming packages such as Pandas [70], BeautifulSoup [11] and Scrapy [55], designed to extract data from websites that make it available freely, as well as a large body of research conducted on the integration of web-based data, which we will now examine.

Early research addressing the problem of maintaining a data warehouse with a set of changing data sources was presented in the EVE (Evolvable View Environment) framework [56, 87]. The EVE framework was put forward as a solution to the view adaptation problem from the point of view of frequently changing data sources, as opposed to queries. It does this through source metadata descriptions which are contained and used to define the views created over these data sources. In the first step to importing a data source into the EVE framework, all incoming sources must *self-register*, i.e. create a record of its data model, data content and capabilities in a Meta Knowledge Base (MKB). Then, when the source makes a change to its data content, the MKB will no longer be able to read the metadata record created. This triggers a process called MKB Evolution where the MKB can take a number of actions to bring the record of the data into alignment with the data's new version, such as *deleting an attribute*. Following this, a view maintenance tool carries these changes forward in the system to update the views on the datasets. The project also has an accompanying language - a SQL extension called Evolvable-SQL or E-SQL - to allow users to express their queries in such a way that they can define preference over the views, such as which elements of the query are mandatory; which elements can be dispensed with if necessary; and which can be replaced. In our case, although we cannot offer the flexibility to allow users to set preferences such as replacability of certain query fragments, we can facilitate users that have no knowledge of SQL by

providing a user interface to our Common Data Model, to allow them to express the queries they need with only minimal technical knowledge. The initial *registration* process involves creating a Model for Information Source Description (MISD) of the data source. This MISD is created using a wrapper function to extract data from its original source. Our approach to metadata at the point of importation is similar but more distributed. When the data is imported into our data lake, we describe the data and its source as an Import Template. However, the mappings between the source schema and Common Data Model are captured separately and, crucially for large numbers of data sources, this process is automated.

The issue of Slowly Changing Dimensions (SCDs) and various strategies for managing this type of change was addressed in [50] and further developed in [53], focusing on dimensions whose values do not remain stable, for example a dimension of personnel in an organisation who may be promoted, transferred or leave. Later in [54], the authors formalised a list of strategies for altering the dimensions, e.g. type 1: overwrite the old record with a new one; type 2: retain the old record but generate a new one with a new primary key, etc. The cleanest and easiest method of dealing with a change to a dimension is type 1 - simply over-writing the previous record with new values, retaining the unique identifier for the record. The downside of this is the lack of an historical record of the change. For our dimensions, we use different approaches depending on the nature of the dimension. For example, a dimension that is rapidly changing like currency conversions would be handled by frequently adding new rows (type 2), whereas for the case of a product being re-categorised according to its classification system, our strategy is more similar to type 3 - adding a new attribute to contain the new value while retaining the old attribute and value for the historical record. Thus, we refer to our changing dimensions as dynamic dimensions as opposed to SCDs because the changes that might happen to them are (a) not slow, and (b) not predictable.

In a rapidly changing data environment, the aim is to avoid having to recompute data cubes from scratch each time the data source changes. The general aim of reducing data latency is often discussed in approaches towards active warehousing

or real-time data warehousing [15, 90, 114]. Key to this approach over traditional warehousing is that the data in the warehouse is updated as it becomes available, as opposed to being buffered throughout the day then the warehouse updated in an overnight batch update [82]. The difficulty with many of these approaches is that they rely on frequent extractions from the initial data staging area (e.g a transactional database). In a business setting, the data warehouse should have an update policy that is cost-effective while reducing data latency insofar as possible. It is costly and inefficient to update data in a data warehouse in an upfront way each time there is a change at source. However, the data must be readily available when it is needed. In our case, the data lake is a low-overhead way of having up to date data within our system but without the costly processing. We are aiming for a way of selecting the data that is needed to address a query, to be loaded to a data cube, assuming that the source will change, so that the cube does not need to be recomputed.

The authors of [52] describe how web-based data and their term a *data webhouse* raises significant challenges in terms of the timeliness, volumes and response time required of the data. It also adds additional stages that must be taken to transform data, such as resolving encoding conflicts, where the web extraction process may have to convert special characters. Their data webhouse allows for the creation of pre-computed data cubes which are downloadable from the World Wide Web. However, web data publishers use various formats, leading to significant challenges with heterogeneity of data formats, schemas and semantic heterogeneity as well as data quality issues [10]. A data warehouse assumes a relational model such as those shown in Figures 1.1, 1.2 and 1.3. However, web data is often not tabular when published. When the data does not have an organised relational schema but has its structure contained within the data itself such as a set of semantic tags or column headers in a comma-separated value (CSV) file, it is considered semi-structured [16]. When the data is semi-structured, there is necessary processing involved to add structure before it can be integrated into a data warehouse. The most common method found in the current body of research in this area is the

use of XML [78, 89, 103]. The advantage of using XML to integrate datasets either without predefined schemas or with heterogeneous schemas lies in its facility to represent the data and the data structure in the same document by way of a set of semantic tags. The research points to the advantages of XML in general, i.e. it is free, extensible, modular, platform-independent and well-supported, as well as its particular suitability for manipulating web-based datasets because of its flexibility in terms of structure. It contains its structure in sets of tag pairs - start-tags and end-tags - with the content being the characters between tags. Examples of the extensibility of XML are the technologies available for combining a relational data warehouse with web data, such as XPath - a language to point to particular elements in a document; or XQuery - a method for querying XML. Although we are avoiding having to integrate to a large data warehouse, we too had to address the issue of semi-structured data and of heterogeneous schemas. We must deliver the same level of flexibility as XML when we are manipulating web datasets, including those used for the case studies presented in this dissertation. However, most of these datasets will be imported in a different type of semi-structured format - CSV files. Like XML, the CSV files contains a basic schema within the data itself, so presents the challenge of source-to-target schema matching where we use a Common Data Model (CDM) to supply the target schemas. The benefit of using CSV files is the ease of importing a CSV to a DataFrame, where there is a straightforward one-to-one mapping of the headers of a CSV file with the columns of a DataFrame, all of which is handled with Python libraries. CSV files have the benefit over XML of being more easily readable by humans, for sanity-checking query results during development.

An example of an approach to using XML technology to address the issue of heterogeneity between data sources is found in [101], where the metadata descriptions of the cubes available in a warehouse are stored in an XML database, one per cube. The authors firstly define the types of conflicts that may be encountered when trying to integrate existing data warehouses - cube-to-cube conflicts, where the schemas are different between cubes; or differences between the specific dimensions or measures within the cubes. To resolve these conflicts, they propose the following steps: (i)

capture the cube schema metadata and the fact data as XML documents and store each in a native XML database, (ii) using this metadata along with the site metadata (the semantic metadata for the dimensions and measures), the data is transformed to conform to a global cube structure, and (iii) integrate to a global cube using XQuery. This global cube is then presented to the users for analysis and querying. This approach of using a number of local databases with local schemas, but which are required to share data in order to meet certain business requirements while each retains its own constraints, is sometimes known as a *federated* database [39]. The drawback of this approach is the amount of upfront processing that is done on each local database, making it resource intensive and time consuming. Although our approach uses similar metadata constructs as [101], it avoids the upfront processing by storing the data in a data lake until it is needed for a query. This way, we never process data not required for user queries. Our metadata constructs describe the source metadata, the unique valuesets of the data cubes, and the source-to-target schema mappings and term mappings, which allow us to avoid upfront data processing while easily preparing sets of parameters that can be used when the data is needed to fulfil a query.

The authors in [49] present an ontology-based approach to constructing a data warehouse (OBDB: Ontology-Based Data Warehouse) using web-based sources. Similar to [85], the approach is driven by user requirements and firstly involves an analysis of the requirements. They model the requirements in a goal-oriented model where each business requirement is represented as a *goal* with properties: unique identifier, name, description, purpose and priority level. This allows for the relationships between goals to be distinguished: goals may conflict or complement each other, or have a parent-child relationship. The parameters of the goals are then mapped to the properties of the global ontology. The global ontology is then used to generate a set of data warehouse ontologies to represent the properties and concepts specifically required for each goal. In the event that the global ontology cannot provide all the concepts for a particular goal, the data warehouse ontology will be extended manually in order to meet it, but without the global ontology being extended.

We adopt a different approach as our data cubes will conform strictly to a subset of the CDM, while the CDM may be extended to capture elements of new source data where necessary. Although we do not construct local ontologies from the global ontology, we do use the global ontology to create a set of mappings for source-to-target schema and data transformation. Considering our large number of data sources compared to the case study presented in [49], generation of multiple ontologies could pose a challenge in terms of overheads. Our approach has no requirement for multiple ontologies and thus, will not face this issue of scalability.

Many existing approaches to ETL or data integration share the difficulty of users requiring answers to their data queries at low latency, while the data sources do not remain stable. When using web data especially, the schema information, method of publishing and vocabulary of the data are highly subject to change. As we have seen, traditional data warehousing techniques are not good at coping with these changes, generally needing a large manual input to adapt the ETL pipeline or even re-structure the data warehouse itself. Therefore, we run into the problem of how to produce a set of data cubes, expressed in terms of a Common Data Model, that can respond to queries when the underlying data may be changing its structure or semantics. The problem of maintaining an up to date database with constantly changing sources has an impact on business intelligence and managerial decisions. The source-to-target mapping relationships between the data sources and the global schema is categorised into: Global-As-View, Local-As-View and Global and Local As View, which is a generalisation of the previous two approaches [47].

In [57], the authors compare two of these approaches to data integration - Global-As-View and Local-As-View, in terms of how suitable each is for coping with typical data integration problems. The Global-As-View (GAV) approach, the older and more traditional approach, means the system has a global schema, which is expressed as a view of the source (local) schemas. Our approach is similar in that we adopt a single Common Model which describes all entities in our system. Queries can then be expressed in the terms of the global schema (common model). From the query-processing point of view, the GAV approach is more efficient as the system

benefits from the *single database property*, i.e. it behaves like a single database. However, a limitation of this approach is that it assumes that the data sources are stable, and may require heavy reworking if the source changes. Additionally, it lacks extensibility in that the global schema cannot contain information not contained in at least one local source. We want our users to have the benefit of having a global schema in which to express their queries, and therefore use the common model as a global schema in our system. However, we assume that the data sources will be unstable and the resulting data cubes will need to be changed, i.e. they will be dynamic data cubes. Therefore, it is necessary for us to adapt the GAV approach. In a data integration system, the user queries are expressed in a *virtual mediated schema*, i.e. a canonical schema to represent all data sources, but not necessarily the schema in which the data is stored. Therefore, the system will necessarily include some form of a mapping or query re-writing process to get the relevant data for the query.

The Local-As-View (LAV) approach, on the other hand, defines a global schema independently from any sources, and the local schemas are then expressed as views of the global schema. For this reason, it is more flexible in terms of adding new sources as the local schemas are more easily extensible, but has the drawback of making query processing difficult as each local schema provides only partial information about the data. We adopt the extensible properties of this approach as we must assume that the system will regularly be adding new sources or managing changes in existing ones. Our approach combines the benefits of GAV and LAV - the canonical vocabulary is more compatible with a LAV approach as it is highly extensible and we assume the sources may change frequently. However, we have the additional benefit of a global schema to which all sources will be mapped and therefore, have the efficiency of having the users express their queries in the language of the global schema.

In [3], the authors propose a method of dynamically updating data cubes when there is a need for *stationary data* - i.e. data owned by the decision maker which in our approach is generally enterprise data - to be merged with situational, i.e.

external and often web-based, data. This situational data - web-based in particular - tends to be more dynamic and unstable. They term their approach *self-service business intelligence*, emphasising the role of the user, but it is related both to *near real-time* and *on-demand* approaches. We use the latter terms more frequently as we do not make the assumption that the end user is either technically adept or a domain expert. However, their approach to dynamic data cubes has influenced our work. The main construct in their architecture is called a *fusion cube* and differs from a traditional data cube in that it can be extended both in its schema and its instances. Each fusion cube is associated with a set of annotations describing common metadata elements: source, freshness etc. In their framework, self-service Business Intelligence differs from a typical OLAP query session when the query requires an existing stationary data cube to access situational data - either dimensional values or additional measures - in order to be fulfilled. To achieve this, they describe a new OLAP operator called *drill-beyond*. The drill-beyond method describes how and where to extend a data cube, where there are two options: drill-beyond the schema, which incorporates new attributes into the cube; or drill-beyond the instances, which adds new instances for an existing attribute. In both cases, the attribute could be a measure or a dimension. The drawback to their approach is the need for the user to choose *which* attributes and/or instances to add to the cube. This is done by way of either question answering or requiring keywords. In our case, the adapting of data cubes to new queries requires minimal input from the user after the initial query launch, as the metadata annotations captured at the importation process enable a query to be rewritten in a data source's local language so that that data source can be queried. Another area of similarity between our work and theirs are in the source-to-target mapping of schemas. Like many of our related works, they have the challenge of heterogeneity in their data sources which requires a reconciliation process between the sources and the final data cube. Traditionally, this is done with a manual process but, like us, the authors of [3] found the need for automation when using fusion cubes.

The authors of [92] present a graph-based approach to integrating online data

sources, where there is a one-to-one relationship between a data source and their initial construct, termed a *StarGraph*, and one-to-many relationship between a StarGraph and a *ConstellationGraph*. Using datasets from Irish transport providers, the data is extracted and undergoes a three-stage process. In the first step, a StarGraph is created, i.e. a graph that acts a common model for the data, representing the set of measures and dimensions found in the source. A set of mapping rules is generated for source-to-target transformation, and stored in a mappings repository. Similarly, we generate a set of parameters for the transformation process to be performed later. However, because we do not adopt a graph-based approach and only process new data when needed for a query, we extend their list of mappings to include source and target terms at the data level as well as at the schema level. Their process continues with an optional transformation step when the measure is not easily detected in the data but needs to be generated from the existing columns. They then integrate the StarGraphs using the mappings collected at the previous stage, and finally populate the resulting ConstellationGraph with the data which is stored in a data lake. The graph is then ready for querying. There are some similarities between this approach and our approach to supplying parameters to both our transformation and query reuse processes. However, our system is not a graph-based system. A CSV file-based data lake and a relational model for data cubes offers us the flexibility required to handle web-based data as well as easily human-readable results. The supplying of parameters from the data lake to the transform and load processes is done in response to a query, not upfront. Therefore, the low-overhead data lake serves our purposes better than pre-loading to a graph.

2.2 Query Reuse

In this section, we present research in the area of query reuse, some key studies from the 1990s to give a background, as well as the more modern approaches that have influenced this dissertation. The storing and reusing of the results of queries launched on a data warehouse reflects Inmon's Top-Down approach where data

cubes are launched from a data warehouse [42], as opposed to Kimball's method of beginning with data cubes and merging them to form the warehouse [54]. As previously stated, a data mart is a data cube that addresses a specific business requirement. Key to query reuse is how to avoid recomputing the same query results over the same data warehouse multiple times, particularly when the data must be combined from multiple sources. As we have already seen, data sources often do not remain static, nor do user needs. This leads to the problem of how to reuse previous views (i.e. query results) when the incoming query and/or incoming data has changed, while still avoiding having to launch the query on the data warehouse as a whole. This is called the *view adaptation* problem [37, 73].

Similarly, in [36], Gupta refers to a problem of view maintenance, defining a materialised view as the storing of a query result, for the purpose of acting as a cache for the data rather than recomputing it every time it's needed. View maintenance refers to the issue of having to update a materialised view whenever the source data is changed. In our research, we address this from the perspective of a data query that requires the data cube(s) - i.e. materialised views - to be updated. The caching of views is part of the solution in the approach towards a real-time data warehouse. In [114], the authors use a multi-level data cache to achieve close to real-time query fulfilment. Their solution proposes a real-time data warehouse architecture where Changed Data Capture (CDC) with a multi-level cache is used alongside ETL. Using OLTP systems as data input streams, the data stored in a cache reduces the query load for the data warehouse. The CDC would drive the caching while ETL directly populates the warehouse. The data from the warehouse and caches then go through Real-Time Data Integration (RDI) to prepare the data for the report and analysis level. They acquire the data in XML format using update cycles from 5 to 60 minutes before the data is moved from the final cache into the warehouse. The benefit of this approach is in the reduction of the lag-time in the warehouse provided by the caching system. However, the view maintenance approach still assumes that the materialised views will be updated as the source data is updated, whether or not it is required for query fulfilment. In our approach, rather than perform this

effort upfront, we cache the results of previous queries and store unused data in a data lake rather than a warehouse, which reduces the need to perform expensive data processing whether or not it is actually required.

Query fragmentation is a classical approach to making query fulfilment more efficient since the 1980's [88]. For example, the authors of [25] differentiate between table level caching and query level caching. Table level caching is suitable for a static data warehouse which, as we have established, is not suitable for our purposes. They proposed a method of caching query results in chunks, and a method to search and reuse these chunks to fulfil queries. This work [25] forms the basis for later work on deconstructing queries as a way of optimising query fulfilment. For example, the authors of [12] use an approach to query reuse that focuses on the adaptation of queries in response to adapting user needs. They make use of the *fragment-based approach* to the view adaptation problem, where the query is deconstructed into the lowest granularity parts possible, and each part is analysed and materialised. The authors of [12] use this approach by materialising a view of the commonly used fragments and adapting these as needed when the data or the query changes. The views are modelled using a Multiview Materialisation Graph. The graph's nodes are one of two types: AND-nodes and OR-nodes. An AND-node represents an operation such as a `select` and `join` while an OR-node represents a logical expression that would yield the same result. This has the effect of allowing them to modify their query results in a low-cost fashion because the less commonly used fragments are not updated if they are not needed, i.e. data independence is preserved. Similar to [12], this reduces the cost of the view adaptation. Our approach to query reuse deconstructs a query into fragments, followed by an initial process to determine whether the set of fragments can be answered by an existing data cube, whether an existing data cube could be updated to answer the query, or whether it is necessary to extract data from the data lake. The updating of a data cube is in response to a query, as opposed to in [12] where it may also be updated because of a change in the source data, even if it is not required by a user query. The authors of [59] extend the work presented in [12] for use on semi-structured data. They use the

view materialisation approach for an XML database, as opposed to a relational model, by materialising commonly used XPath expressions. XML as a data format is often used as a solution to the problem of structural heterogeneity, particularly the need to integrate semi-structured data with a relational data warehouse [84]. Similarly, the authors of [61] apply fragmentation to an XML database, but using fragmentation techniques that usually are applied to relational tables - specifying vertical, horizontal and hybrid fragmentation. Fragmentation that results from the query expression is horizontal and is adopted in our approach. This provides the ability to split the query to be answered by a *combination* of relational tables and CSV sources in the data lake.

As previously discussed, the challenge in view adaptation is in avoiding recomputation of the resultset every time the query is launched, while also avoiding storing so many query results that storage becomes inefficient. The authors of [84], using a fragmentation approach, present a set of heuristics to determine the fragments to be materialised and stored. They do this by first computing a cost associated with the materialisation of each fragment - the shareability of the fragment, the maintenance/storage requirements, cost of reuse, cost of materialisation and the cardinality. Following this, a cost matrix is produced to compare the cost and benefit of each fragment. This allowed them to develop a set of heuristics based on these costs, where either the cost of materialisation must be below a certain threshold or the benefit of materialisation be above a threshold, in order for the fragment to be materialised and stored. The approach was validated using a case study with World-Bikes data. In our case, we have multiple sources and thus, need to consider the cost of joining cubes in addition to constructing them. We adopted the heuristic-based approach as there was early evidence to show that there were multiple ways of fulfilling a query with data captured in both cubes and the data lake.

Probabilistic solutions are also employed for query reuse. In a probabilistic setting, the certainty of the data values is $< 100\%$ but may have a number of possible values, called a *set of possible worlds* with associated probabilities of accuracy [34, 58]. This approach is applicable when users are satisfied to trade a certain degree of

accuracy in their query results in favour of speed. In [30], the authors use approximate query processing to reduce the *latency* of query processing. The results of the queries launched in their system will have an uncertainty, or error, calculated for each approximate result. The authors assume that in order to produce low latency interactive queries, either a substantial amount of upfront pre-processing or considerable a priori knowledge of the data sets is needed from the user. However, our approach uses only a lightweight upfront pre-processing of metadata, while assuming the user does not have extensive knowledge of the data sources. We do not use a probabilistic solution but instead avoid heavy upfront processing by creating a set of metadata mapping templates that provide a transformation solution without having to complete the transformation process until the data is required. Our approach is to retrieve missing data from *outside* the system.

In [44], the authors explore probabilistic query reuse in terms of the effect on the error in the query. They point out the possibility of a cumulative effect where the more a query is reused in a non-deterministic way, the more there can be an error within the error. This influenced our decision-making against using a probabilistic approach, although their approach offered useful design considerations for us where there are multiple possible options for query reuse.

In their work, the optimal query reuse situation is one where the incoming query is a *child* of a previous query, or where the incoming query is *contained* within a previous query. Similar to our approach, the authors use set theory to determine whether a view is a subset, superset or disjoint, first caching a set of potentially usable views before more thoroughly investigating each. If more than one match is found, their system investigates which is optimal to use, e.g. the one with lowest cardinality. In the case of partial matches, they add incrementally to the view by adding shards (i.e. horizontal partitions) of the data until the query is resolved. We do not use a strict sharding approach but use a similar methodology based on cube segments and employ a similar query containment-checking approach. Query containment is the condition where the result of a query is a subset of a previous query. The authors of [71] define relative containment, which is a specification of query containment for

use in a data integration framework, because traditional containment is insufficient in cases of multiple data sources being integrated. In a data integration system, the results produced by a query $Q1$ may be a subset of the results of another query $Q2$, but without $Q1$ being contained by $Q2$. In relative containment, a query is only considered contained by another query if, for every instance of the set of views over which the query is run, that instance of the query result within that view has containment. In our work, our cube metadata constructs give the system quick access to the ranges and valuesets of the attributes of the cube. In this way, without having to query the cube as a whole, we can quickly check if the query is contained in a previous query.

2.3 Modern ETL Architectures

With traditional ETL, data is transformed and loaded in bulk to a data warehouse on a regular schedule. This is an inflexible system with a considerable amount of work needed upfront to program the transformations needed to prepare the data for the data warehouse and resolve heterogeneities. If there is a business requirement for a new data source to be integrated with the system, further engineering and manual effort are required. Before specifying a new approach to the ETL process, it is necessary to understand approaches and challenges in the existing research. It is crucial that we examine on-demand ETL and understand the usage and importance of metadata in ETL solutions. The works described in this section aim to address these challenges and it will be seen how some of the previous research has influenced our work, as well as how we have extended the state of the art. The design of the ETL processes is usually based on both the characteristics of the data and the requirements of the user. Understanding the business requirements of the end user is the first stage of any data analytics undertaking that follows the CRISP-DM methodology [107] - a benchmark for designing the life cycle of a data analytics project that begins with business understanding and finishes with the deployment of a solution, which in turn can lead to new business understanding. Hence, the

design of the ETL processes is often customised to the business requirements.

In [85], the authors present a semi-automated, requirement-driven process to generate a multi-dimensional schema and the accompanying ETL workflow. The inputs to their system are a set of source data stores, each captured in its own OWL ontology, and a set of business requirements. The final output is a multidimensional schema and set of ETL operations. They do this by first identifying which data sources can meet a specific business requirement - requesting user feedback in the case of ambiguity - by looking for corresponding concepts between the business requirements and the ontologies of the data sources, then completing the business requirements with additional information where required. This additional information is then labelled as either a fact or a dimension with a multi-dimensional tagging system. The ETL operations are then defined using the information from the sources adapted to typical ETL architecture. The multi-dimensional schema is built from the concepts in the data that the system determines the most likely to be useful. This differs from our approach in that the multi-dimensional schema is predetermined after an extensive analysis of several Agri sources, and new data sources are adapted to the existing model, with extensions to the data model being rare. Additionally, the authors of [85] use ontologies for each individual source, while ours has only a single domain ontology. We then use source-to-target schema matching to adapt data from all our sources to a Common Data Model. This means that integrating the transformed datasets greatly limits problems with heterogeneity between sources.

A large portion of the literature focuses on increasing the level of automation that can be built into the ETL processes with the goal of reducing the need for user input for processes such as mapping a native schema to a canonical schema. Therefore, many researchers make use of an ontology as a tool to capture the knowledge of a particular industry as a set of concepts and relations. This may be a single, global ontology which captures the semantics of an entire domain, or multiple, local ontologies for individual data sources. Without this tool or set of tools, the manual input required to design the ETL workflow and the specific terms to transform, is considerable.

In [93], the authors present a way to use an ontology to construct the ETL workflow, the reasoning being that an ontology is a solution to the main challenge with integrating datasets: data heterogeneity, both structural and semantic. The ontology used to construct the ETL workflow is first manually constructed using a set of inputs from the designer - a canonical vocabulary capturing the semantics of the domain, plus a set of annotations of the data sources. Although the construction of our ontology is outside of the main contribution of our work, we use this ontology to capture the semantics of our domain in a similar fashion. This ontology then provides the mappings for terms native to the source data.

In [94], Skoutas et al exploit an XML graph-based representation together with an ontology to construct ETL workflow for both structured and semi-structured data. They assume the data inputs will be in a relational schema. They first construct the graph representation for relational and XML data, which are then stored in their first graph construct called the *datastore graph*. Following this, the ontology, with a set of classes, properties and operations, is constructed as a graph representation called the *ontology graph* where the nodes represent the classes and the edges represent properties. The final stage involves the mapping of the datastore graph to the ontology graph, with the ontology providing a set of terms for the former to be mapped to. Similarly, in our work, an ontology is used to provide the set of canonical terms for the native data to be mapped to. However, different from both this work and from [93], we use our ontology to generate a *DataMap* - a lightweight metadata structure that provides a blueprint to drive the transformation process. This also has the benefit that these transforming templates can also be stored and used in the query processing stage of our work. Additionally, the process in [94] of providing the vocabulary and annotations is still a manual one, while we have managed to largely automate this task, significantly reducing the time spent on importing a new data source.

In later research by the same authors [95], they extended their work with a set of graph transformation rules to determine the workflow of the ETL processes - i.e. the selection and ordering of operations. An ontology provides the semantics

for allowing the system to *choose*. Therefore, the authors again use an ontology in order to provide this domain-specific information to drive the transformation stage of the ETL process. The key information constructs stored in the graph are: the data store schemata, the domain ontology, semantic annotations and ETL operations. The use of a graph as opposed to a relational model enables them to integrate structured and semi-structured data sources. The authors illustrate their work with source and target schemas selected from the TPC-H schema. These schemas were chosen in order to resemble real-world schemas and typical scenarios that require ETL. In our work, we use a set of Python libraries to extract data directly from the web-based sources, flatten them in memory and store as a CSV file until needed for a query. However, we have still managed to produce a semi-automated approach to source-to-target transformation without the use of a graph. Additionally, the schemas used in their evaluation were artificial whereas we evaluate ours with multiple real-world datasets as well as an enterprise warehouse. This gives us a much more challenging level of heterogeneity between sources and requires more complexity in our importation process.

In [109], the authors present an ontology-based knowledge fusion method based on data fusion and semantic web technologies. It involves a six-step method called *Primary Knowledge Fusion*, beginning with the extraction of information from a source. The second step is an analysis stage to determine which elements in the data are elements of their ontology, which is a global ontology for the agricultural domain. In the third step, the data is annotated with semantic information from the ontology. The data instances are then clustered by similarity and fused according to a knowledge fusion ruleset. Their ontology defines an integrated hierarchy of agricultural knowledge: definitions and relationships. The purpose was to use an ontology to drive a knowledge fusion method to resolve disparity when integrating Agri data sources. We have a number of architectural elements that are similar to this approach, as well as using datasets in the same domain. Our approach begins with an analysis process that results in enhanced metadata constructs which are used both in the ETL processes and again in the query fulfilment strategies.

However, the authors in [109] raise an important issue that challenges the use of ontologies for transforming data, which is the *knowledge inconsistency problem*. The knowledge inconsistency problem arises when there is a conflict between two pieces of information in the ontology, such as an element being part of two disjoint classes. We will show in Chapter 3 how we make use of an extensible canonical vocabulary to help resolve this problem when it arises.

The use of an ontology to provide semantic annotations is seen again in [13], albeit this time implemented as a graph model. However, the challenges with graphs are much the same in that the authors begin with a set of heterogeneous data sources that must undergo an ETL process, where traditional ETL will be insufficient for this type of data model. Additionally, their approach aims for reusability of the processed data. Their extraction process differs from ours in that they assume their inputs are a set of spreadsheets. As our sources include web-based data, we extend our extraction process into two stages - importation from the original sources, the usage of Python wrappers to convert any web data to CSV files and then, the storage of these sources in a data lake for later extraction. However, in both approaches, their extraction and our importation processes are followed by an annotation process which is guided by an ontology. For their semantic annotation process, they focus on Temporal and Spatial classes. These data types are also important in our work. However, their case study involves only a single agricultural product class, i.e. cereals, while ours involves a complex set of products across multiple sub-domains, meat, dairy etc. Therefore, our ontology needs a heavy extension with product codes and descriptions extracted from the Harmonised System classification of trade products [97], specifically the `04` subset (dairy products), the `02` subset (meat products) and to a lesser extent, the `35` subset (casein).

The traditional approach to ETL is a bulk upload of data, carried out at a time when operational systems required for the running of the business are offline (i.e. overnight). However, as the international sharing of data becomes the norm and data sources are expected to be *always on*, even over a distributed network of branches across different time-zones, this delay while the data warehouse updates is becoming

unacceptable. Modern approaches aim to reduce this delay, or latency, in the data. These approaches to ETL are often seen where there is a changing data source or sources, and a user requirement to have the most up-to-date data available at any one time. This is referred to as *real-time* or sometimes *near real-time* data warehousing. For example, in [15], the researchers identify technologies that can support close to real-time data warehousing, aiming to reduce the latency in the data warehouse. They first identify some key requirements of a real-time data warehouse: (i) continuous data integration as opposed to batch updates, (ii) a rule-driven active decision engine, and (iii) highly available analytical environments, as opposed to the data being inaccessible until each update is complete. To achieve this, their ETL architecture is streamlined by using what they call *ETLets* - Java classes that are dynamically loaded, a process which is managed by an *ETL container* that executes and monitors ETL tasks. Unlike traditional ETL approaches, the data is not stored in intermediary data storage components during the process. The difference with on-demand ETL, however, is that the ETL processes are triggered by user queries, as opposed to by changes in the data. This is an important distinction as it prevents the unnecessary processing of potentially high volumes of data.

The use of near real-time ETL often relies on increasing the frequency of extractions from source data. Obviously, the extraction, transformation and loading of all the data available from a source at each update would be highly inefficient. Therefore, researchers aim to perform ETL only on the changes made to the source data since the last extraction, called incremental loading. The authors in [43] highlight a risk to this approach when the warehouse is constructed from multiple sources, which is often the case. The risk is that anomalies may arise when the changes made to one source - an update or deletion - happen at different time intervals from the changes made to another source with which it is to be joined. The authors propose two basic approaches to prevent these conflicts: either preventing the ETL processes from detecting the mismatch between the historical data or the change data, or allowing the ETL processes to work correctly in spite of the mismatch. They categorise the properties of the data sources to determine which approach is more suitable, such

as sources that are timestamped using the timestamp to capture the most recent update. In our case, our data warehouse consists only of data that is required for a query - a set of data cubes rather than a bulky data warehouse. This allows us to make multiple extractions from a source without the disadvantage of using a lot of processing time on the source data, as it is stored in a low-overhead repository until it is needed. At query runtime in our approach, the results of previous queries are first analysed for possible reuse.

The drawback of the real-time approach to ETL and warehousing systems is the upfront loading of data, done more frequently than in the conventional approach to ETL but still a resource-heavy process because of the frequent extractions. Closely related to the real-time approach but a more sophisticated version of it is *right-time* ETL. The authors of [100] propose a middle-ware to facilitate this called RiTE. In this architecture, rows of data are inserted and stored locally in a buffer until a certain policy is met, defined by user requirements in terms of e.g. data freshness, in which case they are flushed to a memory storage construct termed a catalyst. Following this, the data is loaded to the warehouse. The data may also be loaded to the warehouse in a conventional manner, by-passing the catalyst. The rationale for this approach is that there are some rows of data that are required by the user as soon as they become available, while others may allow a degree of latency. This mitigates the issue of slow insert speeds found in real-time warehouses. The user's queries may then access both the warehouse and the catalyst. In our case, we cache data as data cubes that may be reused, making the data very quickly available in the case of queries that are frequently used, without the need for a large data warehouse.

In a right-time data warehouse solution such as that proposed in [28], the data warehouse is populated when there have been real-world changes to the data and the user requires a strictly up-to-date view. In this work, the architecture is that of an on-demand solution but the data is loaded to the data warehouse before it is transformed - ELT (Extract, Load, Transform) rather than ETL (Extract, Transform, Load) - then transformed in response to queries. This is useful because

the transformation stage of ETL is the most resource-heavy stage as well as the part that is most impacted by changes to the source data. This work was continued in [104], where again the raw data is loaded directly to the data warehouse, where it is stored in a *landing pad* in its native format. From there, the data is cleaned and loaded to a warehouse table, then materialised as a view. This appears to be upfront initially, while refreshing of the MV's (Materialised Views) takes place on-demand, in response to queries. An additional component called the active component is added as a way to query the data already loaded to the warehouse as well as the incoming data, allowing it to perform live analysis on the combined datasets. The aims of this architecture were to increase data freshness as well as facilitate a combined analysis of historical and stream data. We also propose using on-demand processing and selectively processing data that addresses the current user query while using only low-effort processes for the rest. The benefit of ELT over ETL is speed as the data can be loaded to the target database and transformed in place [4]. However, this assumes that minimal transformations are required, while we require all datasets to conform to a Common Data Model. For this reason, instead of performing the transformations after loading, we use a template-based transformation process which makes the process lightweight and generic but ensures that the conformity of the data cubes to the data model is preserved.

2.4 On-Demand ETL

In recent years, On-Demand ETL has represented a big change in how data analysis and business intelligence is performed. With On-Demand ETL (OD-ETL), ETL processes are triggered as needed, as opposed to data being extracted, transformed and loaded in bulk. The key features of OD-ETL are that: (i) the ETL processes are not run until the data is needed for a query, and (ii) only the data that is needed to fulfil that query will undergo the ETL processes. This represents a more lightweight approach to both data processing and data warehousing as well as reducing data redundancy. In this way, it is driven by real business needs. An interesting approach

that aims to fully avoid the processing resources of a database, is [40]. In this work, the queries define how the data is stored and processed. The data is initially stored in CSV files and the user points their queries to the files they wish to query. The system then uses an *adaptive loading* process that assesses when and how to load the data and how much data to load. Like this work, we store our data in the data lake as CSV files, which has the advantage of avoiding the initial overhead of loading to a database. However in their work, the authors require their users to know in which CSV file the data they need is stored, while we create a metadata structure that stores this information so that the user can simply specify their query, and our query processing system uses the metadata gathered from the sources available to select the data sources that can fulfil the query.

A separate approach to OD-ETL is presented in [46], which they call *Lazy ETL*. The approach hinges on an initial load of the metadata in the traditional fashion, *Eager ETL*, while the actual data is not processed until it is needed to address a query, i.e. it is done in a lazy way. The approach is highly scalable because metadata provides insight into the actual data but is inexpensive to load. The metadata is referenced at query time to facilitate the actual extraction, transformation and loading of the data. Their lazy ETL approach was implemented using MonetDB and used a file repository of seismological sensor data as its input for evaluation. Their evaluation shows that their lazy ETL approach resulted in a significant time saving over traditional ETL. Their approach is demonstrated in more detail in their follow-up work [45], which explains how each element of ETL is done in a 'Lazy' fashion. Again, the key to this approach is the difference in how metadata and *actual* data are handled and the metadata is used as a reference for the ETL processes when they are queried. Our approach has similarities to what is presented in [45,46] in that we also load metadata in an eager or upfront fashion, while the loading of the actual data into data cubes is delayed until query time. However, the authors validate their approach with a demo using an existing data warehouse of seismological data. Our work has the additional challenge of needing to address the heterogeneity inherent in using multiple data sources in their native format. Therefore, our solution is more

robust as it uses an interchangeable data model to which multiple sources will be mapped.

Researchers in the Lenses project [111] propose an approach that again delays ETL until query time and is based on probabilistic query processing. In effect, it may not provide exact answers to the queries, but instead makes use of C-Tables (conditional tables) coupled with a model for expressing data selection tasks based on probabilistic components. Lenses are elements of the ETL process but generate PC-Tables as their output. PC-tables - Probabilistic Conditional tables - define a set of possible outputs along with the probability measure of the accuracy of each output. The assumption is that the user may accept an approximate answer if it is sufficiently less expensive. This is evaluated by a heuristic called the Cost of Perfect Information (CPI) [111] which represents the trade-off between the required data accuracy and the effort required to achieve it. The Lenses project is an extension of the work in [110] which presents an on-demand query result processing. In this work, a traditional query is expressed against a probabilistic database and the results are assumed to be incomplete. This differs from typical query processing where the data would have been cleaned and filled before a query can be launched. Hence, the query is what triggers the data cleaning process in [110]. This can be seen as a forerunner to on-demand ETL. This work is also where the Cost of Perfect Information was introduced, to be used as a metric for performance. This research was evaluated using a set of selected datasets such as credit data and real estate data. It is assumed that these datasets are imported once and remain static, whereas we are working with datasets that change frequently. We also assume that new datasets will be added and therefore necessitate a more generic ETL architecture.

The authors in [8] propose an on-demand query fulfilment framework, which operates in a similar fashion to ours. In this work, a *dice* is an abstraction of a set of facts (in this work, facts refers to both measure data and dimensional data), where those facts may be contained in existing data cubes or may be missing; and may be required to fulfil a query or may be irrelevant to the query. In the case where the facts are contained within the existing cubes, but are irrelevant to the query, the facts

will be *dropped* from the cube if space requires it. In the case where the facts are required for the query but missing from the cube, the facts will be extracted from their source. Therefore, the query is launched against a dice management system to determine whether to perform an ETL process, a dropping process or a filtering process in the case where the facts fetched from the source are a superset of those needed to fulfil the query. The main function of the dice management process is to determine the set of missing dice required to fulfil the query. Our query reuse and on-demand system is facilitated by a similar construct which provides a map of the elements of the data contained within a cube, and a methodology for finding what is missing to fulfil the query. A difference between our approach and theirs is how the missing facts are handled. In their case, if there is data required that is missing in the existing set of cubes, they must send a request to the source provider. In our case, the data lake provides an additional resource that is within our data environment. This is necessary because unlike in [8], we are using datasets from a large number of web-based providers, with high amounts of disparity between both the datasets themselves and how the data is published, as well as datasets that may change between instances of publication.

2.5 Summary

The body of work in the area of creating a data warehouse with web data, all comes to the same conclusions in terms of the challenges of working with web-based datasets: that they are unstable and prone to changes, and that there is a lack of conformity between datasets from the same domain but different providers. XML is a commonly used solution to creating a data warehouse from web-based data [59,101]. XML has the benefit of being free and extensible as well as containing the structure of the data within a set of tags. However, with the growth in the use of newer semi-structured data formats such as JSON and CSV, we have elected to make use of Python libraries to store data in a CSV data lake.

Other approaches use local ontologies to represent each web-based dataset, using

these local ontologies to either be integrated to a global ontology or be mapped to an existing global ontology, to construct a data warehouse [49, 85]. Such a solution would be unscalable for us and instead we use a canonical vocabulary to create a metadata layer to provide mappings for transformation before loading, as well as access to the data lake for querying directly, similar to the *virtual mediated schema* [57]. Graph databases are gaining a lot of attention and are used as a solution to integrating web-based datasets in [92]. However, this approach loads data to the graph in an upfront manner, while we have a requirement to only load the data that is needed for a query, when it is needed for a query.

In an environment where both the data sources and user needs are frequently changing, we have a requirement to create a set of dynamic data cubes on which to launch queries and to update in response to queries. The fusion cubes approach seen in [3] has some similar requirements as ours, in terms of integrating web-based data with enterprise data. We are influenced by their *drill-beyond* method which describes how and when to extend a data cube. However, a more automated approach is needed, as their method required a high level of knowledge of the data on the part of the users.

Having a set of dynamic data cubes on which to launch queries gives an additional challenge to the area of query reuse. The overarching goal of this area of research is to avoid the unnecessary re-computation of existing view for query fulfilment. Therefore, many approaches search for a way of reusing the results of previous queries to address new ones. The view adaptation problem is the challenge of doing this over dynamic, changing data sources. Some researchers will use an architecture called Changed Data Capture (CDC) to track changes in the source data and take action when it detects changes. We use some specific elements of this type of methodology, such as a timestamp of the most recent importation from source. But in an environment of potentially rapidly changing data, as opposed to the more commonly seen Slowly Changing Dimensions (SCDs), taking action triggered by every change in the source data would be a heavy use of resources for possibly redundant data. Instead, changes to our query results are triggered by new queries.

Our approach has more in common with fragmentation-based approaches such as [12], which deconstructs a query so that it can be decided which elements require a cube to be updated to fulfil the query. We use the fragmentation method but also consider how this method needs to be adapted for querying a data lake instead of data cube, hence it needs to be combined with a query rewriting method like in [57].

Modern approaches to ETL focus on the automation of individual tasks as well as the automation of the designing of the ETL pipeline. The use of an ontology is common in these types of research, particularly for the supplying of term mappings from the source native terms to a set of canonical terms. For example, in [85], each source is captured in a local ontology and the user's requirements are matched with an element of an ontology. However, this raises the issue of knowledge inconsistency or term ambiguity such as mentioned in [109]. We use our global ontology to produce metadata constructs to provide parameters to a transformation process for a more generalisable solution. Our approach of using a canonical vocabulary is highly extensible and requires minimal human intervention compared to other approaches.

Traditional approaches to ETL can cause delays, or latency, in the data. Modern approaches prioritise the reduction of this latency between the extraction of the data and having it loaded for querying. This is generally known as near-realtime. Many of these approaches rely on more frequent extractions from the source. A smarter approach is incremental loading, or right-time ETL, where only the data that has changed at the source since the last extraction is loaded. However, this can lead to anomalies when the warehouse is constructed from multiple sources [43], which ours is. For that reason, we investigated what is necessary to design and deliver a fully On-Demand ETL.

In an On-Demand ETL framework, data is not processed until it is needed to fulfil a query. For example in [45], metadata is processed in an upfront manner and provides a guide to the curation of data for transforming and loading. Other approaches incorporate a probabilistic database for data imputation on the fly [111]. Our On-Demand ETL framework differs from previous approaches in a number of ways, most importantly in our use of web-based datasets and in our large number of sources, and

in our ability to process previously unseen datasets with no additional engineering. We have extended our ETL architecture to enable us to process multiple disparate datasets with minimal intervention. The approach most similar to ours [8] uses a construct called a dice which acts as a map of the available data for querying. We can draw some comparisons between their results and ours in our evaluation as we use a similar means of manipulating the use of query reuse and on-demand ETL to address queries. However, it should be noted that our methodology has additional tasks to undertake when searching for query matches, when integrating partial matches and additional complexity when selecting one match over another.

Chapter 3

An Architecture to Support On-Demand ETL

The primary goal of this research is to provide a lightweight, on-demand ETL system that delivers query fulfilment using a mix of pre-existing queries and new data sources. This chapter provides an overview of the key components of the data environment and overall workflow.

In Section 3.1, we present a high-level overview of the three phases of our approach, which will be described in detail in the following chapters. An outline of our extended ETL methodology is presented in Section 3.2 while in 3.3, the main constructs developed for our query processing architecture are presented. We present our Common Data Model in Section 3.4 as a set of facts and dimensions and a canonical vocabulary. In Section 3.5, we present the case studies that are used for evaluation throughout the dissertation.

3.1 Methodology Overview

In order to deliver the challenging goal of on-demand ETL, there are a number of requirements that will be placed on our system architecture. Data must be

imported from multiple sources, including websites, and the system must be capable of handling previously unseen datasets with minimal adjustments when adding a new source. We must be able to process queries that can both reuse previous results and go outside the traditional ETL system to generate the results.

Figure 3.1 presents a high-level overview of the three phases of our methodology. In the first phase, we will demonstrate our approach to addressing RQ1 - we introduce our novel components that facilitate the creation of dynamic data cubes in the absence of a static data warehouse. This is discussed in detail in Chapter 4. We will demonstrate our ETL processes on a subset of our data sources before the launching of any queries. However, a user query is assumed to be the trigger for this phase.

In phase 2, the resulting data cubes are used to address our RQ2 in our query reuse process which compares an incoming query with the set of existing data cubes, without directly querying the cubes. We will show how we process data cubes and queries into our enhanced metadata constructs so that a cube can be identified as a potential match for the query. This will be described fully in Chapter 5

Finally, we will present in Chapter 6 how our on-demand ETL methodology addresses RQ3 by using the systems devised for RQ's 1 and 2 to fulfil the query by seamlessly combining previously fulfilled queries with external data, without needing to adapt the ETL processes for new data sources. The combined approach to query reuse with on-demand ETL will result in the resultset being returned to the user after post-processing.

At the end of Chapters 4-6, we provide a brief demonstration of the outcome of the methodology presented in the main body of the chapter.

3.2 Dynamic Data Cubes

In this section we present the extended ETL processes and components that address the need for a methodology to use previously unseen data sources to create dynamic data marts. Our ETL workflow follows traditional ETL but has additional features

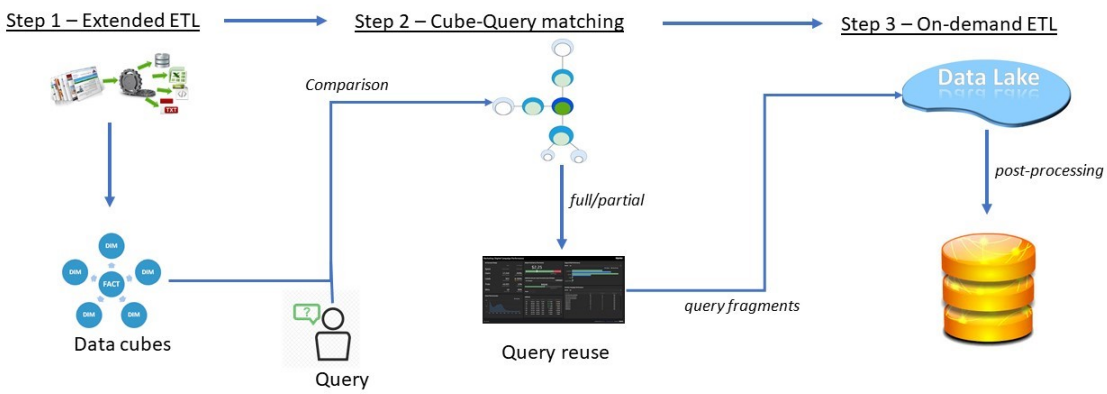

Figure 3.1: Methodology Outline

to meet the requirements and the first research question specified in Chapter 1, of creating an architecture to create dynamic data marts. This question requires the creation of a number of components and processes to create and maintain a set of dynamic data marts - which do not come from a data warehouse using traditional ETL methods but area created from potentially unstable data sources. With traditional ETL, the extraction process typically involves fetching the data from its source and placing it in a local database, known as a *staging area*. This is unsuitable for our needs because we must manage multiple heterogeneous data sources, some of which are new to the system.

Figure 3.2 outlines our extended ETL architecture, with the processes in blue and the storage components represented as graphics. There are five processes: Import, Analyse, Extract, Transform and Load; and the key components are the Common Data Model, which refers to a collection of standardised schemas for modelling business concepts, their entities, attributes and relationships; the Data Lake, used as a low-overhead repository; and the Metabase, which is a multi-model repository for all of our metadata descriptions. A worked example of a dataset at each stage of this architecture can be found at Appendix E.

Our ETL process starts with the importation of new data sources. When we discuss data **importation**, we are referring to the process of acquiring data from its publishing source, and storing it in our data lake. For each data source provider,

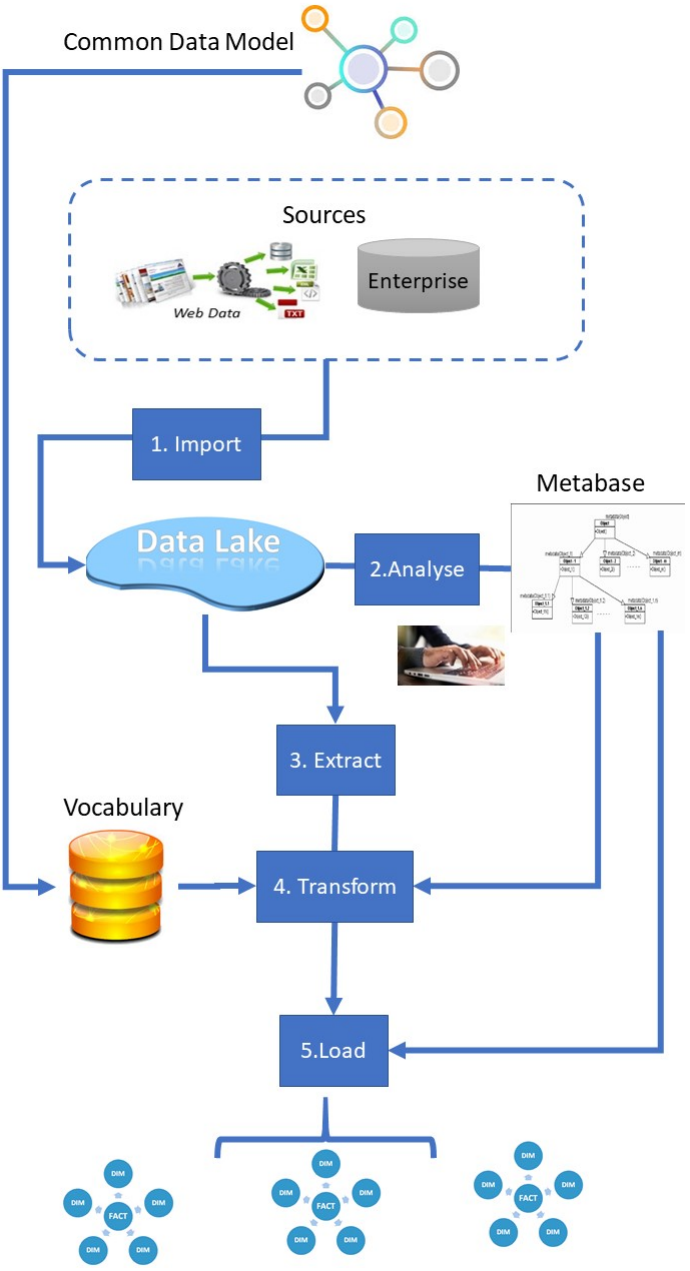

Figure 3.2: Dynamic ETL Architecture

the user must specify a source provider record which is used to provide a set of parameters that facilitate a generic data importation, without having to tailor the importation process to each individual data source.

Using a number of Python libraries specifically for these purposes [1, 2, 11, 70], we can accomplish data importation in the following ways:

- Importation into CSV from web-based formats such as JSON.
- Importation from relational tables e.g. MySQL.
- Parse documents constructed with tags, i.e. HTML and XML.
- Interact with elements on a web page, e.g forms, to produce a dataset to be downloaded.

The data is brought into memory using the Python libraries and stored in the data lake as a CSV file. The use of a data lake for the initial storage of the data, rather than bringing it into a database structure, is different from traditional ETL and satisfies our ETL system requirement for avoiding the upfront processing of data. The data lake is updated from its contributing sources as a set of batch process and thus is assumed to be up-to-date, i.e. the version of data available at source will be imported from the source and stored in the lake. If the data required for a query is not available in the lake, it is not available.

A further requirement of our ETL system is the capturing and management of metadata. The upfront/eager metadata processing and delayed/lazy processing of data facilitates our on-demand ETL solution is similar to the approach presented in [45, 46], described in Chapter 2. For that reason, our architecture includes a Metabase to contain the different metadata constructs with metadata descriptions of newly imported data captured in an eager fashion.

In the next step in our methodology, with the data stored in the data lake as a set of CSV files, the user puts each through a brief (approximately five minutes) initial **analysis**. The analysis of a newly imported dataset has two main functions:

a vetting process to determine whether the dataset can conform with our Common Data Model, and the creation of metadata descriptions if so. The vetting process is to determine that the dataset has a base level of attributes that can be associated with a dimension found in our Common Data Model.

This is followed by the constructing of a metadata layer for the data lake. The specific constructs will be described in detail in Chapter 4. The metadata layer acts as a portal through which the data in the lake is accessed only when required to populate a data cube to answer a query. Traditionally, the format for metadata comes from the Dublin Core schema and its extensions. The Dublin Core Metadata Element Set (DCMES) [41] is a vocabulary of 15 metadata fields, used to describe any kind of object. Our component-based templates will select a subset of fields from this set (Format, Coverage, Subject) and use them to create a set of structured fields with which to describe the data.

The final requirement of our ETL architecture is that it should be as generic as possible, avoiding having to be customised for each data source. Therefore, it will be seen that the metadata constructs generated upfront provide sets of parameters to the Extract, Transform and Load processes, which can then be performed on data newly imported from a new source provider with no adjustments needing to made to the workflow.

We define **extraction** as the process of selecting a dataset from the data lake and bringing it into a staging area to be processed, without bringing the data into a data cube. When being extracted from the data lake, the data is converted to a set of attribute-value pairs, as described in §3.4.5. The benefit of this is to be able to extract data from many different schemas and integrate them without having to tailor the loading process for each file.

Data **transformation** is the process of preparing a dataset to be loaded to a target data structure so that it can be viewed with the data of the structure as a global view. Examples of the kinds of transformations performed when integrating data from multiple sources include standardising terms when different sources use differ-

ent terms for the same concept, the mapping of source attribute names to target dimensional attributes, conversion of measures when data sources use different units of weight/price etc.

In this step of the ETL process, the data has been extracted from the data lake as a set of attribute-value pairs. Each of the attributes is mapped to either a dimensional attribute from one of the dimensions in §3.4.1, or the name of a measure found in one of the facts in §3.4.2. The measure values are converted using a set of conversion functions, while the dimensional values is mapped to a canonical term.

Following the transformation process, the data is **loaded** to a **data cube** - an aggregation operation on a fact table in a data warehouse [33], wherein the cells of the cube are occupied by the measure values, aggregated across each of the dimensions. Our cubes are represented as a star schema as seen in Chapter 1.

3.3 Cube-Query Matching

In this section, we outline our approach to fulfilling the requirement to reuse previous queries (RQ2) and to then query data from the lake in an on-demand way (RQ3). It is assumed at this point that the data has been imported to the data lake and undergone the ETL processes into a set of data cubes. From this point, the stage of matching cubes with incoming queries can be described as five processes:

1. Cube metadata description
2. Query fragmentation
3. Cube-Query Comparison
4. Lake querying
5. Data cube materialisation

Figure 3.3 shows our query fulfilment architecture. For each data cube, an enhanced **metadata description** is saved to the Metabase. Information about the extent

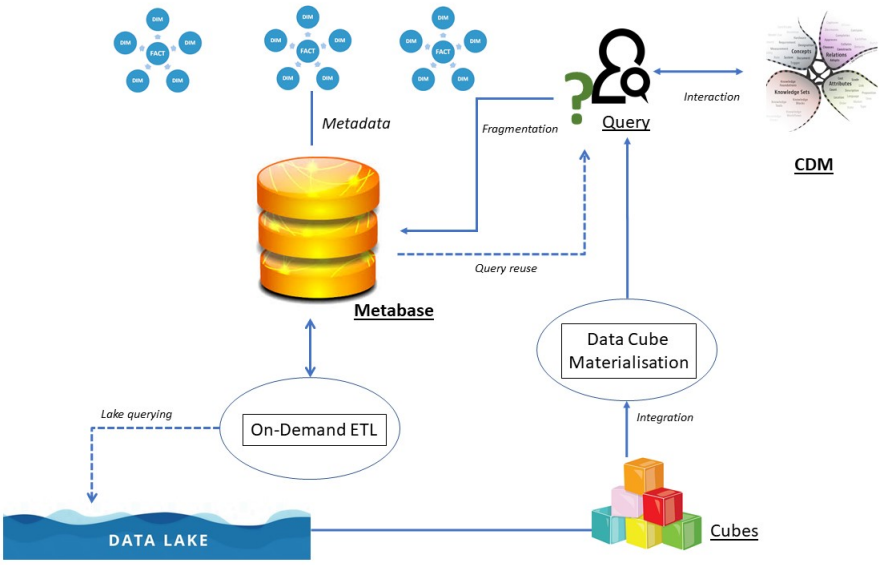

Figure 3.3: Query Fulfilment architecture

to which the cubes contain the elements of the Common Data Model is saved in the Metabase for accessing when a query is launched. In Chapter 5, we present our enhanced metadata constructs of cubes and how we represent a map of the cubes contained within the data cube store and the attributes and ranges of values available. These metadata descriptions of the existing data cubes enable us to address research question 2 by providing a guide to which cubes can be reused in response to a query.

We have shown in §2.4 that the on-demand ETL process is triggered by the launching of a query against the data stored in the data lake. The query is **compared** with existing cubes from a map of which elements of the Common Data Model are already contained in pre-existing cubes. A full match means that one or more cubes contains the required attributes and constraints and as a result, we do not need to generate a new cube in order to satisfy this query. A partial match means that there is overlap between the required attributes and values of the query, and those of one or more cubes, meaning a strategy is required to generate and execute a sub-query to find the data missing to fulfil the query. If an existing cube can answer it, the cube is simply returned to the user. Assuming that this easy case will be unusual, the next step is our final on-demand ETL solution.

If the query reuse process does not return a full match for the query, the final stage is **data lake querying**, in which the metadata of the data lake provides access to the data contained within the system that has not been processed by a previous query into a data cube. We will show in Chapter 6 how a query is re-written to be launched on the data lake, and how the files that can fulfil the missing elements of the query can be found and transformed.

The results of both the query reuse and lake querying undergo **post-processing** stages including integration and filtering.

3.4 Common Data Model

In order to produce an On-Demand ETL that can process a large number of heterogeneous data sources, we use a Common Data Model (CDM) to encapsulate a shared language to which all data sources will conform. In this section we present the three main components in the CDM - a set of *dimensions*, a set of *facts*, and a *vocabulary*. These dimensions, facts and the vocabulary are specific to an industry. By specifying a CDM, we have a structure and set of terms to which imported data will conform after processing - i.e. every dimension, dimensional value and fact in the data must be mapped to a canonical dimension, dimensional value or fact from the CDM. In traditional ETL systems, this model is designed by hand and holds a rigid structure which is not extensible without significant manual intervention. The automation of this mapping process forms an important contribution in this work as we will investigate the accuracy of this automation while producing considerable time savings over the manual method. The specific CDM used in this work was presented in [65]. Due to the size of the model, we will provide an overview description in this section and, in §3.5, provide a detailed discussion on those segments that play a prominent part in our case studies and query examples.

3.4.1 Dimensions

The first component in the CDM is a set of dimensions and dimensional attributes. This gives us a canonical set of terms to describe the attributes of an industry. A Dimension is defined in Definition 3.1.

Definition 3.1. A Dimension is a four-tuple $Dimension = \langle N, A, V, SD \rangle$ where N is the name, A is a set of attributes, V is the valueset, and SD indicates whether the dimension is static or dynamic.

In Definition 3.1, the name will be a unique name assigned to the dimension when the CDM is initially designed, attributes are a set of dimensional attributes, the valueset will be the set of canonical values, and the dimension may be static or dynamic. Examples of both will be shown in §3.5 when we present our implementation case study. Static dimensions contain a set of values that (it is assumed) should never be updated, such as a set number of terms to describe gender. They are populated once at system setup and are not altered afterwards. Conversely, a dynamic dimension is one that we anticipate will change, and possibly frequently, such as products available from a manufacturer. Although these changing dimensions are closely related to *Slowly Changing Dimensions* [53,54], here they are referred to as *dynamic* dimensions because our approach to updates does not align with any of the sub-types of SCD's found in [54].

3.4.2 Facts

The second component of the CDM is the set of facts. The facts of the CDM are one or more measures and foreign key links to sets of dimensions.

Definition 3.2. A Fact is a four-tuple $Fact = \langle N, M, F, D \rangle$ where N is the name, M is the set of measures, F is the frequency, and D is the set of dimensions.

In Definition 3.2, the name of the Fact will be unique. The relationship between a fact and a measure may be one-to-one, one-to-many or many-to-one. A fact may be

connected to one or more dimensions, and a dimension may be connected to one or more facts creating a constellation schema.

3.4.3 Canonical Vocabulary

The next element of the CDM is a vocabulary of canonical terms or dimensional values to which all source terms are mapped. This is a highly extensible vocabulary as terms in the source data may change or a new source may need to be imported which contains terms not yet mapped to any canonical term. It will be shown in Chapter 4 how this is managed and when user intervention is required.

The components of the vocabulary are:

- `Attribute_lookup` - a set of attribute name matches to convert source attribute names to target attribute names, where the target attribute names are a set of measure names and dimensional attributes;
- `Standard_term` - the complete set of canonical dimensional values;
- `Source_term` a large and extensible set of terms found in the data sources, annotated with a flag stating whether it is a dimension or a measure, and, if the former, which dimension it is related to;
- A set of `mappings` from `source_term` to `standard_term`, where there may be a many-to-one relationship between `source_term` and `standard_term`;
- A `ruleset` - a set of conversion rules to convert measure values when the unit of measure requires transformation;
- A set of currency conversions, stored in the dynamic dimension tables `dim_currency_daily, dim_currency_weekly, dim_currency_monthly` and `dim_currency_annual`.

The differences between the ruleset and the currency conversion tables are that the former is for cases where the unit is a unit of weight, volume or price, i.e. currency per weight. Therefore the conversions will remain constant, e.g. to convert *Euro per*

ton into *Euro per KG* as seen in Example 3.4.1, the conversion function will remain the same regardless of currency exchange rates. The `dim_currency_` dimensions are populated with currency exchange rates from the International Monetary Fund [29] and the dimension provides a foreign key link for the relevant unit of currency for the relevant date, so the exchange rate to the Special Drawing Right can be accessed by the user if required.

In Example 3.4.1, a number of conversion rules are shown for a fact table with price as the measure.

Example 3.4.1. Unit conversion rules

$$\texttt{Euro per ton} \mapsto \texttt{EUR/KG} : f : x' = \frac{x}{1000}$$

$$\texttt{US - \$/lb} \mapsto \texttt{USD/KG} : f : x' = \frac{x}{0.453592}$$

$$\texttt{\$/mt} \mapsto \texttt{USD/KG} : f : x' = \frac{x}{1000}$$

$$\texttt{EUR/KG} \mapsto \texttt{cent/kg} : f : x' = x \times 100$$

In Example 3.4.2, the conversion rule shown is for a measure found in our Common Data Model, animal slaughter numbers. Some source providers publish their data on animal slaughters in `1000 head` and others provide them in `head`, where a 'head' is a single animal, so this rule allows us to combine those figures in the fact table. However, in other cases the rule is simply a mapping to change the spelling of the term to that of the canonical vocabulary.

Example 3.4.2. Slaughter conversion rule

$$\texttt{1000 head} \mapsto \texttt{head} : f : x' = x \times 1000$$

$$\texttt{Kg per carcass} \mapsto \texttt{KG/carcass} : f : x' = x$$

The ruleset is extensible with the current version found in Appendix A, where each rule has a standard unit to which the source unit will be converted. Apart from the

Field	Value
source_term	INDICATORS
standard_term	unit
attr_type	D
dimension	dim_unit
dimension_attribute	unit

conversion rules, the vocabulary currently contains 81,545 `source_term` instances and 80,841 `standard_term` instances.

The attribute_lookup currently has 81 mappings from the attribute names found in source data to a canonical attribute name. In Example 3.4.3, the lookup mapping from the native source term "INDICATORS" is matched to the canonical term "unit". It can happen that there is a many-to-one relationship between a `source_term` and a `standard_term`.

This is an example of the knowledge inconsistency problem arising - that a source attribute name, depending on its source, may be mapped to more than one dimensional attribute. Therefore, we annotate each attribute mapping with the name of the dimension to which it applies, to resolve this ambiguity.

Example 3.4.3. Attribute lookup

3.4.4 Data Source Providers

Our data providers publish data in varying intervals: some monthly, some weekly etc. There is one-to-many relationship between a data source provider and a data source. Hence, the information about the sources is contained in two places.

The information about the source provider that does not change with each interval is stored in a *source provider record*, defined in Definition 3.3. These are somewhat similar to elements of a PolyWeb data-summary [48] but expanded for non-web data.

Definition 3.3. A **source provider record** is a 5-tuple $SPR = \langle N, W, I, F, U \rangle$ where N is the name, W is a URI, I is the industry, F is the update interval and

U is the update method.

In Definition 3.3, N is the name of the organisation or company which is publishing the data; W is the way of accessing the data such as the Uniform Resource Identifier - such as a web address, API, file path - at which the source can be accessed, which may have a one-to-many relationship with data sources; I is the type of industry or domain of the data such as dairy, meat etc; F is the frequency with which the data is published at the source - weekly, monthly etc; and U refers to how the system should treat new datasets incoming from this source, in relation to the previous extraction - either append or overwrite. Some examples of the bodies that provide this data are shown in Table 3.1, which shows the name of the company or organisation that provides the data, the frequency with which it updates, the method for updating, the industries published by the source and a link to the URL found in the bibliography. The full set of source provider records is stored in the `dim_source` dimension, which is a dynamic dimension. For each data source required for inclusion in the system, a source provider record is specified. This facilitates a more dynamic importation of new sources, where traditional warehousing would require re-engineering for a new source.

3.4.5 Data Types

Here we provide a brief description of some data types and ways of representing data that we will be using frequently throughout this thesis, and the purpose for which they are used.

- **Attributesets** refers to any set of attributes in a dataset - the column names in a relational table, the headers in a CSV file, tags in XML etc.
- **Valuesets** refers to the **unique** set of values for any attribute. An attribute and value will have a one-to-many relationship but an attribute and a valueset will have a one-to-one relationship.

Table 3.1: Data Source Provider Details

name	**frequency**	**update**	**industry**	**URI**
Eurostat	weekly, monthly, annual	overwrite	meat, dairy	[26]
USDA	weekly, monthly, annual	overwrite	meat, dairy	[77]
StatCan	monthly	overwrite	meat, dairy	[18]
Comtrade	monthly	overwrite	multiple	[22]
StatsNZ	monthly	overwrite	dairy	[76]
Bord Bia	weekly	append	pigs, dairy	[14]
Quandl	monthly	append	grain	[83]
OECD	monthly, annual	overwrite	financial	[27]
Tesco	daily	append	retail	[98]
ZuivelNL	weekly	append	dairy	[115]
Pig333	weekly	append	pigs	[80]
DairyAustralia	monthly	append	dairy	[23]
CLAL	monthly	append	dairy	[20]

- **Queries** are expressed in our research as a four-tuple: $Q = \langle ID, RA, RV, F \rangle$ where ID is a unique identifier, RA is an Attributeset, RV is a set of constraints and F is one or more aggregate functions. If an attribute is present in RA but not in RV, we consider this the equivalent of a `SELECT *` for this attribute.

- **Constraints** are the filters specified by a query applied on the dimensions and measures. Each constraint is a tuple $C = \langle A : v_1, ..., v_n \rangle$ where A is an attribute and v_x is a filter. The filter may be a valueset where a list of dimensional values is specified for that attribute. It may be a measure filter such as $x < 50$ or a $\neq null$ restriction.

- **attribute-value pairs** is a method of representing data as a set of tuples $\langle attribute, value \rangle$. This is a commonly used way of representing data and has the advantage in its simplicity, where the data can be passed to a function and that function need not have to deal with varying data structures.

- **dates** refer to any value found in one of the four dimensions in the CDM

to do with dates: `dim_date_daily, dim_date_weekly,dim_date_monthly` and `dim_date_annual`. These dimensions are static and are populated with dates at their respective intervals for a period of 200 years. The dimensions also specify a standard format in which dates are to be expressed. For daily and weekly datasets, the dates are to be expressed as YYYYMMDD. For monthly datasets, the dates must be expressed as YYYYMM, and for annual datasets, the format is YYYY.

3.5 Agricultural Data Case Study

In this section, we provide an overview of the case studies which will be used to evaluate our on-demand ETL solution. Firstly, we provide details on the Common Agri Trade Model (CATM), which is the specific implementation of the Common Model used for implementing our architecture.

Following this, we describe four typical business requirements to motivate some of the queries used in our evaluation. For each requirement, we show how the query will be expressed using our format shown in §3.4.5. Case study 1 is an example of a simple query where the user requires a small subset of the data available, aggregated to produce a key insight. Case studies 2, 3 and 4 are examples of data marts created with larger amounts of data, to be used in some analysis such as prediction.

Finally, we describe a number of our datasets which will undergo our extended ETL process to form a set of data cubes, which will later be used in our query reuse system. These datasets were taken from data source providers from the Agricultural sector, selected by an expert in this domain. Additionally, there is a sample of data from our industry partner used as one of the sources. These datasets vary in size, format and dimensionality.

3.5.1 Common Agri Trade Model

To provide an application of this research, we implemented an Agri-specific CDM called the Common Agri Trade Model (CATM) to provide both a standard representation for data acquired by the system [64]. The Agri industry was selected as a use-case for this project as it is an example of an industry that often has a need to integrate internal with external datasets, as well as there being a large number of diverse datasets available from government sources. However, there are other industries that would meet the same criteria which may also have supplied an application for this work, as the contributions of this work are not dependent on being implemented on an Agri system.

A small number of extensions have been made to the CATM since its original specification in [65]. Here, the facts and dimensions are described as they are today: in Tables 3.2 and 3.3, the full set of dimensions is listed, and are sub-categorised as 16 *static* and 14 *dynamic* dimensions. In these tables, `rows` is the number of rows in the table, `values` is the number of *unique* values in the dimension not including the primary key, and `facts` is the number of fact tables to which this dimension is linked. Each of these dimensions captures an element of the trade of agricultural products, such as the products, the entities buying and selling the products, the units in which the production, or price of the products as measures.

An example of a static dimension is `dim_trade_flow`, which is a dimension that describes the flow of traded products. This was created as a one-off action using the finite number of terms that can describe trade flow. This dimension, its attributes and its set of values can be seen in Table 3.4 where `flow_sk` is the primary key, `flow` is a single character code and `flow_desc` is the full word for each type of flow.

An example of a dynamic dimension is `dim_trade_product`, which uses the Harmonised System [97] to categorise trade products. However, over the course of the project, multiple products were re-categorised as well as new ones added and old ones discontinued; in each case the dimension required an update. This dimension currently has 1,227 records, each containing the product code and description of an

Table 3.2: Static Dimensions

name	rows	values	facts
dim_age_group	25	45	1
dim_status	6	13	5
dim_conformation	5	5	1
dim_date_annual	201	201	8
dim_date_daily	73414	123732	4
dim_date_monthly	2412	2625	10
dim_date_weekly	10660	10915	6
dim_gender	3	6	1
dim_fat_score	15	15	1
dim_grade	10	20	2
dim_trade_flow	6	12	1
dim_outlier	6	11	28
dim_price_type	31	51	3
dim_stat_regime	8	8	1
dim_channel	5	5	1
dim_breed	3	4	1

Table 3.3: Dynamic Dimensions

name	rows	values	facts
dim_trade_product	1227	2436	1
dim_supply_demand_type	39	75	2
dim_currency_annual	13267	1481	1
dim_currency_daily	4845325	303958	2
dim_currency_monthly	159193	15494	2
dim_currency_weekly	703561	64879	1
dim_geo	666	1728	28
dim_index_type	1	2	1
dim_measurement_feature	5	6	1
dim_population_economics_type	248	248	3
dim_org	12	24	1
dim_unit	647	657	28
dim_product	71871	72241	22
dim_source	108	143	28

Table 3.4: Dim_Trade_Flow details

flow_sk	flow	flow_desc
1	N	not specified
2	I	import
3	E	export
4	R	re-import
5	X	re-export
6	T	total export
7	F	total import

Table 3.5: Dim_Trade_Product details

trade_product_sk	product_code	product_desc
1	0401200000	Milk not concentrated nor sweetened 1-6% fat
2	0401300000	Milk and cream not concentrated nor sweetened <6% fat
3	0402100000	Milk powder <1.5% fat
4	0402210000	Milk and cream powder unsweetened <1.5% fat
5	0402290000	Milk and cream powder sweetened <1.5% fat
6	0402910000	Milk and cream unsweetened- concentrated

agri-food product, either from the meat or dairy industry. The first six rows are shown in Table 3.5.

The fact tables are subdivided by the frequency with which their associated sources update. Tables 3.6, 3.7 and 3.8 show the fact table names as categorised by their level of granularity: Yearly, Monthly and more fine-grained respectively. In addition, these tables contain the number of dimensions linked to the fact, together with the measure(s). Some facts have several measures, usually the economic statistics of a country. In this case, there will be a link to a dimension that contains a set of measurement features, `dim_measurement_feature`; a set of population/economic features, `dim_population_economics_type`; or a set of features of supply and demand, `dim_supply_demand_type`.

3.5.2 Queries

We begin with a set of queries that will be used in our query reuse and On-Demand ETL evaluation. We will use the following suite of queries to evaluate our query

Table 3.6: Facts: Yearly

Name	dims	measure
fact_consumption_annual	6	consumption
fact_demographic_population	9	population stats
fact_economic_forecast	8	multiple economic measures
fact_economic_stats	7	multiple economic measures
fact_population_annual	6	population (human)
fact_production_annual	6	production
fact_slaughter_annual	6	slaughters
fact_supply_demand_annual	7	supply and demand
fact_vital_statistics	7	multiple measures

Table 3.7: Facts: Monthly

Name	dims	measure
fact_cold_storage_monthly	6	storage
fact_index_monthly	7	protein index
fact_normalised_gdp	5	GDP
fact_population_monthly	6	population (animals)
fact_price_monthly	10	price
fact_production_monthly	6	production
fact_sales_monthly	6	sales
fact_slaughter_monthly	8	slaughters, weight
fact_stocks_monthly	6	stocks
fact_supply_demand_monthly	7	supply and demand
fact_trade_monthly	12	weight, value

Table 3.8: Facts: biweekly, weekly and daily

Name	dims	measure
fact_price_biweekly	10	price
fact_price_supermarket_weekly	7	price
fact_price_weekly	6	price
fact_farm_sales_weekly	7	sales
fact_production_daily	6	production
fact_production_weekly	6	production
fact_slaughter_weekly	6	slaughters
fact_usda_all_info_weekly	6	slaughters, production

reuse and On-Demand ETL approaches. The queries range from very simple, to more complex and requiring a large number of sources to be combined to produce a result. We will evaluate our approach to fulfilling these queries by altering the extent to which previous queries and lake data will be used. The first case is our full cube reuse, where 100% of the data to fulfil the query comes from the store of data cubes. The final case is our full on-demand, where 100% of the data to fulfil the query comes from the data lake.

Case Study 1. Striploin prices

For the first requirement, we take a simple use-case of wishing to track the sales of striploins - a product manufactured by our industry partner - over time. In this case, the sales are measured in weight and in monetary value.

Query ID:Q001

RA: (date,product_desc,trade_weight, trade_value)

RV: product_desc:"Striploin"

FuncSpec: function_type:sum, attribute:price, group_bool:False, group_order:0

Case Study 2. Historical prices

It is often valuable to examine the prices of products between multiple countries, over a long period of time. This can be educational for noticing, for example, the difference between the cyclical seasonal trends between multiple countries as well as between products. For the second query, the user has a requirement to view price datasets from Austria, Germany and China, for all dates available.

Query ID:Q002

RA: (date,geo,price)

RV: geo:("AUSTRIA","GERMANY","CHINA")

Case Study 3. International trade

Agri data from international publishers is generally made available on a weekly or monthly basis, making it highly beneficial to have a large amount of historical data for attempting predictive analysis, whether one-step ahead or multi-step [7]. For the

third requirement, the user wishes to create a large historical record of the exports of Agri trade products, measured by weight and value, from Australia, the US, Canada, Ireland, France and Spain. This sort of large dataset requires data imports from multiple publishers of international trade data, and may be used to form a training dataset for predictive methods.

Query ID:Q003

RA: (yearmonth,reporter,partner,flow,product_code,product_desc,trade_weight,trade_value)
RV: reporter:("AUSTRALIA","UNITED STATES","CANADA","IRELAND","FRANCE", "SPAIN"), flow:("export")

Case Study 4. International trade, production & prices

Features of interest to Agri decision-makers often involves the interaction between several time-series datasets, such as which metrics influence each other, in which direction and over what time lags? For example, if the price of a product goes up, how quickly will that cause demand rates to decrease? Does this rate differ for each product and/or each country? For the fourth requirement, the user has a requirement to examine the relationship between the trade metrics used in the previous requirement, with the levels of slaughters and production, along with the price for each product, from the same countries in Case Study 3.

Query ID:Q004

RA: (yearmonth,reporter,partner,flow,product_code,product_desc,trade_weight,trade_value, slaughtering,price,production)
RV: reporter:("AUSTRALIA","UNITED STATES","CANADA","IRELAND","FRANCE", "SPAIN")

3.5.3 Datasets

In Table 3.9, we show details for some of the sources that will be used to resolve the queries in the previous section. These sources will be used at various stages in

Table 3.9: Agri Data Files

source	format	measure(s)	rows	attrs
statsnz	CSV	trade_weight, trade_value	942565	9
bordbia	HTML	slaughters	5635	4
bordbia	HTML	price	24990	5
clal	HTML	production	913	5
dairyaustralia	HTML	production	652	5
eurostat	CSV	trade_weight, trade_value	13484	10
kepak	SQL server DB	trade_weight, trade_value, yield, offcut_value	5000	15
pig333	CSV	price	14636	5
statcan	CSV	trade_weight, trade_value	2560	10
usda	CSV	trade_weight, trade_value	95725	10

our evaluation. Of the sources listed in Table 3.9, 4 are HTML web pages, usually displayed to the users as tables, 5 are CSV files obtained from an API such as the USDA QuickStats portal, and one is a SQL Server database. Kepak is our industry partner for this work. The measures listed are the main metrics being published in the data. Also shown is the number of attributes, i.e. columns or headers, found in the data. It can be seen that multiple datasets can be imported from one source provider, such as Bord Bia which publishes both price and animal slaughter datasets.

In Chapter 4, we will show how these datasets are processed. In following chapters, we will show how these sources are used to resolve the queries.

3.6 Summary

The purpose of this chapter was to provide a high level overview of the architecture that underpins our solution. Here we introduced the CATM - our common model and its dimensions, facts and vocabulary, along with some of the constructs used to match a source term to a canonical term. Following this, we provided an overview of the architecture we have constructed to produce our solution. This was split into two levels - the data processing layer and the query processing layer. The key components of the data processing layer such as those used for data storage, and the components in our template-based transformation process, were described.

A brief outline of our query reuse process was given. The architecture of each of these layers forms two of our three key contributions, the third being our final on-demand ETL solution that makes use of the architecture as a whole. We have also described our data sources and sample queries which will be used in our various experiments. The processes for the construction, population and usage and the data processing layer will be described in detail in Chapter 4, while the same for the query processing layer will be described in Chapter 5.

Chapter 4

An Extended ETL for Dynamic Data Marts

The first goal of this research is to achieve a more dynamic ETL system than the traditional approach by emphasising the processing of metadata over the data. In order to fulfil the research question of delivering dynamic data marts, we have devised a methodology for metadata components used to process incoming data sources to prepare the data for loading, as well as a way of creating data cubes outside of a warehouse environment. In order to deliver dynamic data marts, we require a construct to allow data to initially be imported into our system in such a way that the importation process does not need to be altered if the data source changes or when a new source needs to be added. Thus, we devised a way of organising the source metadata into an *Import Template*. The second requirement is a way of supplying the information needed to map each source's native schema to the target schema of the Common Data Model, without using a data warehouse. We have facilitated this process by using our second novel construct, the *DataMaps*, to store a set of transformations for each source.

In §4.1, we describe the process of importing a new data source to our ETL architecture. The key constructs that facilitate this process are introduced - the Common

Data Model, the Import Templates and the DataMaps, their purposes and their importance for the overall working of the system, and how they are built and populated. In §4.2, we describe how these components are utilised to transform the data and in §4.3, we show how we load the data to a data cube ready for querying.

As we have covered in Chapter 2, our approach to dynamic data marts is influenced by work that dynamically extends multidimensional cubes in terms of their schema or their instances or both [3]. Similarly, we have a goal of achieving a set of data marts that are constructed using previously unseen data sources and will be expanded with external data in response to queries. In order to achieve our approach to dynamic data marts, we create a number of metadata constructs that both facilitate the processing of previously unseen data sources into data cubes, and are later used to select files from these sources to integrate with existing data cubes in response to queries. Thus, we expand upon the work in [3] by using the metadata descriptions in multiple ways - both to provide a blueprint for data transformation and to select the data files for on-demand ETL.

4.1 Extended Data Extract

This section details what is involved in importing a new dataset from its source and in extracting it from the data lake. Figure 4.1 illustrates the important components and processes in our ETL system, with the processes marked in blue. In this workflow, the system has been initialised as the Common Data Model has been imported as a once-off action. The first process is that the data is Imported from external sources as well as enterprise data supplied to us. The data undergoes an initial Analyse process to produce the data lake metadata. It is stored in the data lake, from which it is Extracted as a set of attribute-value pairs.

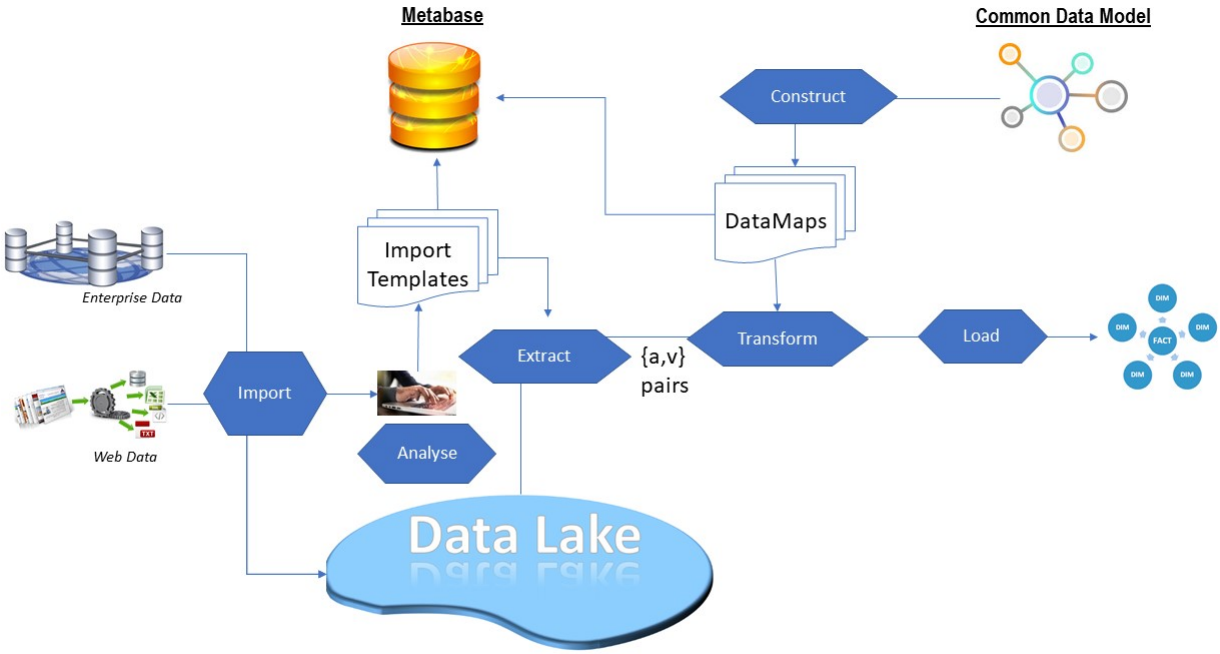

Figure 4.1: ETL processes for dynamic data marts

4.1.1 Data Importation

In this section, we present our methodology for importing the data to the data lake. As shown in Chapter 3, Section 3.4.4, a source provider record is specified for each data source required. From each of these sources, the source is accessed using the data address specified in the source provider record, and the data is imported from this source using Python and libraries specified for this purpose. The data is converted from its various formats to a set of CSV files. This is different from conventional data lakes and has the benefit over other forms of querying multi-model web-based data such as [48] in that, after the conversion to CSV, we can query these files using only one querying strategy rather than requiring each source to have its own querying language or method.

4.1.2 Dataset Analysis and Metadata Descriptions

Although the acquisition of metadata is common in ETL workflows, we present our specific approach to this - an exhaustive set of metadata descriptors called the Import Template. This is somewhat similar to the self-registration process in [56] but at the level of the individual dataset as opposed to the source provider. The Import Templates contain three key pieces of information: (i) source metadata, (ii) data mart management and (iii) data file layout. This is the only part of our ETL architecture that remains a manual process. However, this is a process of less than 5 minutes to gather the set of metadata descriptions and store an Import template for each source to be imported. An Import Template is formally defined in Definition 4.1.

Definition 4.1. An **Import Template** is a triple $IT = \langle SM, DMF, PM \rangle$
where SM is the Source Metadata, DMF is a set of Data Mart Flags and PM is the file metadata.

Definition 4.2. Source Metadata is a triple $SM = \langle N, M, D \rangle$ where N is a link to the source provider record, M is a set of measures, and D is the data import date.

In Definition 4.1, the Source Metadata is the set of metadata elements that describe the source of the data file, defined in Definition 4.2. N is a link to the source record provider in the `dim_source` dimension where the source provider information is stored, and D is the date of the most recent importation from this source.

Definition 4.3. Data Mart Flags are a set of Boolean variables $DMF = \langle Q, T, L \rangle$ where Q is whether or not the data has been Queried, T is Transformed, and L is Loaded.

Data Mart Flags, defined in Definition 4.3, are a set of Booleans that are set to False by default then changed to True when the data is queried, transformed and loaded. The importance of this will be seen when we show how our ETL process is triggered in Chapter 6.

In Definition 4.1, PM is a set of metadata elements that describe physical elements of the file:

- num_cols is the number of columns in the data file;
- num_valid_cols is the number of columns that will be used, as some columns in the native data may be superfluous;
- num_rows is the number of rows in the data file;
- header_start is the line number that contains the headers of the data, as some of the native files have superfluous information before the headers such as company logos;
- dim_col_list is a list of one or more column numbers that contain the dimensions;
- fact_col_list is the list of one or more column numbers that contain the measure values;
- skip_rows is a list of row numbers that can be skipped during the import of data from this file, again that are likely to contain superfluous information.

Table 4.1 shows two instances of Import Templates. The attributes `scrape_date, source, measure(s)` are the elements of the Source Metadata; the Data Mart Flags `transformed, loaded` and `queried` are all set to 'False' by default, then updated to 'True' as each dataset undergoes these processes; and the remaining meta-attributes refer to elements of the Physical Metadata: i.e. `num_rows` - the number of rows in the file, `skip_rows` - which, if any, rows in the file should be skipped while importing the file etc. The values in the right-hand cell of each table are the instances of the Import Templates, one for each data file.

The rest of the Analysis process involves the creating of a DataMap for each dataset that is imported, which is a metadata description of the transformations that take place to prepare the data for loading. This approach is similar to [6], where a set of

field	value
scrape_date	29_07_2019
source	eurostat
measure(s)	trade_weight,trade_value
transformed	False
loaded	False
queried	False
num_cols	15
num_valid_cols	8
num_rows	4384
header_start	0
dim_col_list	10,1,3,7,4,5,12
fact_col_list	14
skip_rows	null

field	value
scrape_date	13_02_2019
source	dairyaustralia
measure(s)	sales
transformed	False
loaded	False
queried	False
num_cols	7
num_valid_cols	5
num_rows	5000
header_start	0
dim_col_list	0,1,2,3
fact_col_list	4
skip_rows	null

Table 4.1: Import Templates

metadata templates provide the parameters for a highly generalisable transformation process. It also bears a resemblance to the model-weaving approach in [24] in that a set of mappings between source and target elements are specified in order to easily transform heterogeneous sources, except that our mappings are between a source and a common model, rather than between sources. These will be discussed in §4.2.

The result of this is a data lake with the Import Templates providing an interface through which the data can be accessed from the lake. The set of Import Templates are stored in the Metabase as seen in Figure 4.1. The Metabase is a storage component for all forms of metadata that are stored during our ETL and querying processes. It has multiple different models for the metadata contained within it, and is used at various points during our workflow.

4.1.3 Data Storage and Extraction

In the data lake, the data is encapsulated from the rest of the system, with the Import Templates providing an interface with which to access the data. Although the data files are in CSV format, they will still contain disparity between them in terms of the number of redundant columns, rows that do not contain usable data etc. The Physical Metadata fields of the Import Templates informs the process which rows and columns to extract.

This process of extracting a dataset from the data lake is presented in Algorithm 4.1 which takes as input IT the Import Template and F a file in the data lake. The correct columns are selected from the data file using the `dim_col_list` and `fact_col_list` fields of the Template. Next, the file is scanned column by column and row by row, beginning at the line number specified by the `header_start` field, and staged as a set of attribute-value pairs where the attribute is the name of the column in the CSV file and the values are the dimensional or measure data. The benefit of this format is genericity; the processes following do not need to be tailored to datasets of different dimensionality when all datasets to be processed are in this format.

Algorithm 4.1 Extract From Data Lake

```
1: function ExtractFromLake(IT, F)
2:     Initialise av_pairs=[]
3:     columns ← IT.dim_col_list + IT.fact_col_list
4:     if |columns| ≠ IT.num_valid_cols then
5:         Request user input
6:     end if
7:     for column ∈ F.columns do
8:         attribute ← column
9:         for row ∈ F.rows[IT.header_start :] do
10:            value ← row.cell
11:            av_pairs = av_pairs + (attribute, value)
12:        end for
13:    end for
14:    return av_pairs
15: end function
```

4.2 Data Transformation

While the system now has access to new data and sources, it is not yet usable as it is not represented in the model or structure of the CDM. Having been imported to a data lake with a metadata description, then extracted as a set of attribute-value pairs, the data is now considered staged for transformation and loading. In this section, we first detail the construction and population of a DataMap, which is a metadata description of the transformations that take place to prepare the data for

loading. This approach is similar to [6], where a set of metadata templates provide the parameters for a highly generalisable transformation process. It also bears a resemblance to the model-weaving approach in [24] in that a set of mappings between source and target elements are specified in order to easily transform heterogeneous sources, except that our mappings are between a source and a common model, rather than between sources.

The rest of this section is structured as follows: in §4.2.1, we define the DataMaps formally; in §4.2.2, §4.2.4, §4.2.4 and §4.2.5, we describe our data transformation strategy in a set of four functions that map the data from its native schema to the canonical form; in §4.2.6 we describe how we automated the process of creating the DataMaps and address the potential issues that may arise during the complicated process of schema matching. The errors found are categorised and we discuss the automation of the DataMap construction. Finally, in §4.2.7, we show how the DataMaps are used in the Transformation process.

4.2.1 DataMaps

Rather than a set of hard-coded transformations which must be manually updated every time a new source is to be imported, in our system we construct a *DataMap* to provide a blueprint for the transformation of data. For each of the attributes and values in the data, a DataMap supplies a canonical term from the canonical vocabulary, or a measure conversion function, for the data to be transformed to. Therefore, the construction of a DataMap involves the correct identification of that set of canonical terms. A similar approach to storing and reusing data transformations can be found in [74] but our approach does not require the user to be familiar with SPARQL or RDF. Instead, our queries are constructed from lists of canonical attributes and values, the mediated schema as described in [71].

A DataMap is defined in Definition 4.4.

Definition 4.4. A DataMap is a 4-tuple $TT = \langle S_s, CDM, S_t, M \rangle$ where: S_s is the set of source schemas, CDM is the canonical data model, S_t is the set of target

Table 4.2: DataMap Fields

Attribute	Type	Notes
attr_type	char	D(dimension) or F(fact)
supplement	Boolean	True (added term) or False
rule	int	Reference to measure conversion rule
source_term	String	Term in data source file
standard_term	String	Term converted to
dimension	String	Domain model's dimension name
dim_attr	String	Domain model's dimension attribute

schemas where $S_t \subseteq CDM$ and M is the suite of functions to map S_s to S_t.

Definition 4.5. $M = \langle AT, SM, DM, RA \rangle$ where AT is AttributeTyping, SM is SchemaMatch, DM is DataMatch and RA is RuleAssign.

Definition 4.5 defines a set of functions that are required to populate the DataMap. These four functions are:

- **AttributeTyping**: identify each S_s element as D or F (dimension or measure)
- **SchemaMatch**: identify S_t.
- **DataMatch**: create the mappings $SO \rightarrow ST$ where
  - SO is the source original term.
  - ST is the standard term from S_t.
- **RuleAssign**: assign a measure conversion rule $\langle ID, SO, ST, CONV \rangle$ where ID is a unique identifier, SO is the unit to be converted, ST is the unit to be mapped to, and $CONV$ is the formula to convert.

A new DataMap is initialised with the fields shown in Table 4.2. This occurs after the data has been imported to the data lake and the Import Template has been written for this dataset.

Algorithm 4.2 takes as inputs a ruleset R, a domain vocabulary O, an attribute_lookup AL, a common data model CDM and a data file F. The algorithm initialises a

blank DataMap (TT) with fields `attr_type`, `rule`, `source_term`, `standard_term`, `dimension` and `dim_attr` and demonstrates the process of populating each of these fields using a dataset with a set of attributes A= $\{a_1, ..., a_n\}$ and values V= $\{v_1, ..., v_n\}$ for each attribute.

4.2.2 The SchemaMatch Function

The process begins with an *assisted* schema matching function where the source schema is mapped to the target schema and the target schema is a subset of the elements in the CDM. Each element of the source schema is matched with an element of the target schema, using the subsection of the canonical vocabulary called the attribute_lookup. It may be matched with:

1. Case 1 - A single Domain Attribute
2. Case 2 - A single Measure
3. Case 3 - Multiple Domain Attributes
4. Case 4 - Multiple Measures
5. Case 5 - Both an attribute and measure
6. Case 6 - Unmatched

Cases 1 and 2 represent a straightforward 1-to-1 schema element matching and there is no action required. Cases 3 to 5 represent various types of *knowledge inconsistency* [108] that can arise in the matching, and the attribute_lookup will resolve these to a single measure or dimensional attribute. For the final case, either the element will not form part of the final data cube, or an additional element will be added to the vocabulary to serve as a match. At this stage, the schema is supplemented with any terms that are missing from the dataset in order to fulfil the set of required attributes of the Common Data Model. This is only done when the attribute will have a single value for every instance of the data. Two examples

Algorithm 4.2 Construct DataMap

```
1: function CONSTRUCT(A, R, O, AL, CDM, F)
2:     Initialise new DataMap DM
3:     for a ∈ F.A do
4:         source_term ← a
5:         get a′ from AL
6:         if |a′| <> 1 or a.supplement = True then
7:             Request user input    ▷ If <1, new term is added to CDM; if >1, get AL.dimension for correct term
8:         end if
9:         standard_term ← a′
10:        if a ∈ CDM.measures then
11:            attr_type ← F
12:            dimension ← null
13:            dim_attr ← null
14:        else
15:            attr_type ← D
16:            get dimension_cdm from CDM
17:            get dim_attr_cdm from CDM
18:            dimension ← dimension_cdm
19:            dim_attr ← dim_attr_cdm
20:            row = [dim_attr, source_term, standard_term, dimension, dim_attr]
21:            DM = DM+row
22:            get A.V
23:            for v ∈ V do
24:                source_term ← v
25:                try
26:                    get v′ from O
27:                catch v′ = null
28:                    standard_term ← null
29:                finally
30:                    standard_term ← v′
31:                end try
32:                dimension ← dimension_cdm
33:                dim_attr ← dim_attr_cdm
34:                if dimension_cdm=dim_unit then
35:                    get rule_id from R
36:                    rule ← rule_id
37:                end if
38:                row = [dim_attr, rule, source_term, standard_term, dimension, dim_attr]
39:                DM = DM+row
40:            end for
41:        end if
42:    end for
43:    return DM
44: end function
```

of this are seen in Example 4.2.1. From one of our sources, Pig333, the dataset published in its native format does not contain a `product` column, as the context of the website supplies the information that the dataset is the price of pigs. Similarly, the Statcan source provider does not specify a `reporter` attribute as the source only publishes Canadian data. In other cases, the information may be contained in the name of the file or database. In these cases, we supplement the data with this static attribute.

Example 4.2.1. Supplemented attributes.
Pig333: product=pig
Statcan: reporter='CANADA'

In lines 3-9 of Algorithm 4.2, **SchemaMatch** takes place whereby a dimension and its attributes are obtained from the common model and added to the DataMap. This is done for each dimension present in the source data. If an attribute raises one of the types of ambiguity mentioned above (cases 3-6), the user is required to add a correction. If the number of matchings generated is more than 1, one must be selected; while if no matchings are generated, the vocabulary is updated. A post-hoc validation identifies and allows for correction of any matching errors. The output from this process is the data cube definition, a *star schema view* derived from the domain schema.

4.2.3 The AttributeTyping Function

In lines 10-15, **AttributeTyping** takes place where it is determined whether the source attribute is a dimension or a measure. The attribute is searched against the canonical measure names in the Common Data Model. If the attribute is a measure, the `dimension` and `dim_attr` fields are left blank and `attr_type` is assigned the value 'F'; if it is a dimension, `attr_type` is set to 'D' the `dimension` and `dim_attr` fields are filled in with canonical terms from the common model. For each data attribute, a new record is added to the DataMap with each of the fields seen in Table 4.2 filled.

Table 4.3: Sample Rule Type

Attribute	Type	Notes
rule_id	int	I.D. number
source_term	String	Unit in data source file
standard_term	String	Unit from common model
conversion	$x' = f(x)$	Function to convert measure

4.2.4 The RuleAssign Function

In lines 34-37 of Algorithm 4.2, the **RuleAssign** takes place: if the value is a unit of measure, the system retrieves a conversion ruleset to enable a conversion for the measure data. The fields for a rule are seen in Table 4.3 where `rule_id` is a unique identifier to refer to the rule; `source_term` is the unit of measure used in the source original data; `standard_term` is the canonical unit of measure, and `conversion` is the formula to convert `source_term` to `standard_term`. During the process of populating the DataMaps, the `rule_id` is extracted and used to refer to the conversion rule.

An example of a rule to convert Tons to KG is seen in Example 4.2.2. The function retrieves an identifier of the rule to be used based on the source unit and the target unit so the formula to convert the measure values can be used during the transformation process. This identifier is inserted into the `rule` fields of the DataMap.

Example 4.2.2. Ton_KG Sample Rule
rule_id:R023
source_term: Ton
standard_term: KG
conversion: $x' = x * 1000$

4.2.5 The DataMatch Function

The final process in creating the DataMaps is DataMatch, wherein each dimensional value in the source original data is mapped to a standard term from the canonical vocabulary. Similar to the process of matching a target schema to the source schema,

attr_type	rule	source_original	standard_term	dimension	dim_attr
D	-	Ireland, Rep. of	IRELAND	dim_geo	geo
D	R023	Ton	KG	dim_unit	unit
F	-	amt	price	-	-

Figure 4.2: Populated DataMap Sample

data values are extracted from the vocabulary and a mapping is generated for each term. This is a similar problem-space to that found in ontology matching research, and has some similar challenges. For example, this can also lead to term ambiguity where certain dimensional values may belong to more than one dimension. To use an Agri-specific example, "North America" is the name of a country and is intuitively part of the `geo` dimension. However, it is also the name of a breed of cow, therefore is also part of the `product` dimension. Hence, this necessitates a post-hoc check of the DataMaps to identify and correct cases of ambiguity.

The **DataMatch** process is captured in lines 22-33 of Algorithm 4.2, where $v \in V$ represents the set of values associated with the attribute being mapped. Between lines 22-30, the system uses the vocabulary to find a canonical term in the data model to which the value v can be mapped. This step sets the `standard_term` attribute to the appropriate canonical term. If one is not found, the NULL value assigned at line 27 is detected during a later step, indicating an action is required. In lines 38 and 39, a new row is added to the DataMap for each dimensional value.

The output of the DataMatch process is a fully populated DataMap, the first three lines of which can be seen in Figure 4.2 where the `rule` seen in the second row references the rule seen in Example 4.2.2.

4.2.6 Automating the DataMaps

In previous versions of our ETL system, the DataMaps were initially a manual annotation process similar to that found in [38], wherein the data was manually matched with terms from the vocabulary. When creating them for datasets of any

considerable size, it quickly began to take several hours to create the DataMaps and apply the four processes - AttributeTyping, SchemaMatch, RuleAssign and DataMatch - for each dataset. We resolved this by automating the process of selecting the matching terms using the steps shown in Algorithm 4.2. When this process was being automated, the challenge was in evaluating and reducing the potential loss of accuracy when the four processes were being done by a function implemented in Python, as opposed to by a human. This work was published in [69], [67] and [66].

There are several issues that arise during the process of mapping one dataset to another, both at the schema and the data level. In [57], the authors point out the difficulty of schema matching and data integration, while comparing the two common strategies. Global-as-View has the downside of assuming that the datasets are static, while Local-as-View does not. But Local-as-View approaches are more difficult to query the end result. In our approach, we use a global schema to which all datasets will be mapped, but still assume the datasets will not be static. It will be seen in Chapter 6 how the query rewriting process that is required to address this, is facilitated by the DataMaps.

The authors of [5] point out the difficulty of heterogeneous attribute names between two or more sources, that represent the same semantic concept. We see the knowledge inconsistency problem in action in the six possible outcomes of the SchemaMatch process in §4.2.2, that the process can sometimes present with ambiguous results - more than one possible target term for a source term, or no target available. Our evaluation of the automated DataMaps, when compared to those created by human participants, will be seen in Chapter 7. The automation of the DataMap generation reduced the time taken to create and populate the DataMap from 6 hours to a few seconds, while an accuracy of 85.55% was found. A full analysis of the speed and accuracy of the DataMap generation process, compared to the manual equivalent, is discussed in §7.2. We observed and classified any errors in the accuracy of the SchemaMatch, RuleAssign and DataMatch processes as follows:

- Error Type A: A one off mistake. This is when a `source_term` is mapped to

a `standard_term` that is an incorrect mapping, as a once-off. This is seen in Example 4.2.3 where the `source_term` "OFL SWN-ED-F/CH" is mapped to a `standard_term` "Swine livers- edible offal- frozen", which is incorrect.

- Error Type B: An errant vocabulary term. This is when there is an error in the vocabulary, leading to repeated cases of the same `source_term` being mapped to a wrong `standard_term`.
- Error Type C: A missing vocabulary term. Where a `source_term` is incorrectly mapped to Null because no suitable term can be found for it in the vocabulary.
  - Error Type C1: Missing rule. A specific case of C where the term to be mapped is a *unit of measure*.

Error Type A represents human error. It occurred during the manual process of creating the DataMaps, but not when this process was automated. Type B and C errors provided information for updates made to the vocabulary or ruleset, when a matching for a term was missing. The impact of these errors may be that the data cannot be queried if the attribute names are not mapped, that the data may give incorrect results if the terms are mapped to a wrong term, or that the measure data may be incorrectly converted, leading to incorrect insights. Hence, as can be seen in Algorithm 4.2, it was necessary to make this process semi-automated rather than a fully automatic process. The system may request assistance from the user if necessary. Therefore, the final stage of preparing the DataMap is a quick human validation process to assess and correct any errors that occurred during the creation of the DataMap.

Example 4.2.3. Error type A
OFL SWN-ED-F/CH → Swine livers- edible offal- frozen
Correction:
OFL SWN-ED-F/CH → Swine edible offal- fresh or chilled

4.2.7 Applying the DataMaps

When applying the DataMaps, the datasets to be integrated into the system are extracted from the data lake and staged as a set of attribute-value pairs, and for each there is a fully populated DataMap to supply the mappings to transform the data from its source terms to the canonical terms of the domain-specific data model.

In order to do this, we use the algorithm shown in Algorithm 4.3. This algorithm takes as input a set of attribute-value pairs AV, the DataMap DM which is the output from Algorithm 4.2, and R - a set of rules for measure conversion. The first step is replacing the source attributes with the canonical attributes. The attribute is checked to see if it is a dimension by checking the `attr_type` field in that instance of the DataMap. If it is a dimension, there will be a canonical term to map the source term to. Next, the attribute is checked to see if it is a date. If so, the format of the date is converted to the canonical format, such as converting MMDDYYYY to YYYYMMDD. If the attribute is a measure, the conversion function is extracted from the ruleset by checking the `rule_assign` in that instance of the DataMap, and the function is applied to the value. If there are multiple units of measure in the data, the dataset is split by the unit of measure and, for each unit, the relevant conversion rule is extracted from the ruleset and applied. After these processes have taken place, the data is now composed of terms that conform to the CDM and, therefore, the existing data cubes in the system.

4.3 Data Loading

In this section, we describe the process of loading the data to a data cube. The data has at this point been extracted and transformed. The next step is to use the description of the Common Data Model stored in the Metabase to determine the structure of the data mart for the data. Hence, the data cube is composed of dimensional attribute names, canonical measure names, and canonical dimensional values for each attribute.

Algorithm 4.3 Data Transformation

```
1: function TRANSFORM((A,V),DM, R)
2:     while n do Get next(A,V):
3:         Get A.
4:         Get standard_term(A′) from DM
5:         Get V
6:         if A = dimension then:
7:             Get standard_term(V′) from DM
8:             if |V′| = 0 then:
9:                 Request user input
10:                V′ ← standard_term(V)
11:            end if
12:        else if A = date then:
13:            V′ ← formatdate(V)
14:        else if A = measure then:
15:            Get rule_id from DM
16:            Get conversion from R
17:            V′ ← V ∗ conversion
18:        end if
19:        Return A′, V′
20:    end while
21: end function
```

The InitialiseCube function uses the Common Data Model and an Import Template. It begins by extracting the measure or list of measures from the Import Template IT. This measure is used to search the description of the Common Data Model in the Metabase to determine the fact table that holds this measure. If more than one fact is returned at this point, it will be because they are facts that have the same measure names at different frequencies. Hence, the frequency for the data is determined by taking the source name in the Import Template and checking that name in the `dim_source` dimension and extracting the source provider record SPR for that source, which will give the frequency with which that source updates. A new data cube is initialised with the measures and dimensions found in that fact. Finally, it calls the function PopulateCube before updating the `Loaded` Data Mart Flag to True.

The PopulateCube function now takes the newly created data cube as input, along with the transformed data (A', V') and its associated DataMap TT. The attribute

Algorithm 4.4 Data Load

```
1: function InitialiseCube(CDM,IT)
2:     Get IT.measures
3:     fact ← CDM.Fact where Fact.measure == IT.measures
4:     if |fact| > 1 then
5:         Get CDM.dim_source
6:         Get dim_source.SPR where SPR.name == IT.source
7:         get SPR.frequency
8:         fact ← CDM.Fact where Fact.measure == IT.measures and
   Fact.frequency == SPR.frequency
9:     end if
10:    Initialise new Cube C = ⟨Facts, Dimensions⟩
11:    C.Facts ← Fact
12:    C.Dimensions ← Fact.Dimensions
13:    PopulateCube
14:    IT.Loaded ← True
15: end function

16: function PopulateCube((A',V'),DM,C)
17:    while A', V' do
18:        Get next A', V'
19:        if A' ∈ C.measures then
20:            Insert V'
21:        else
22:            Get DM.dimension where DM.standard_term == A'
23:            Get C.dimension where C.dimension == DM.dimension
24:            Get dimension.sk where dimension.value == V'
25:            Insert dimension.sk
26:        end if
27:    end while
28: end function
```

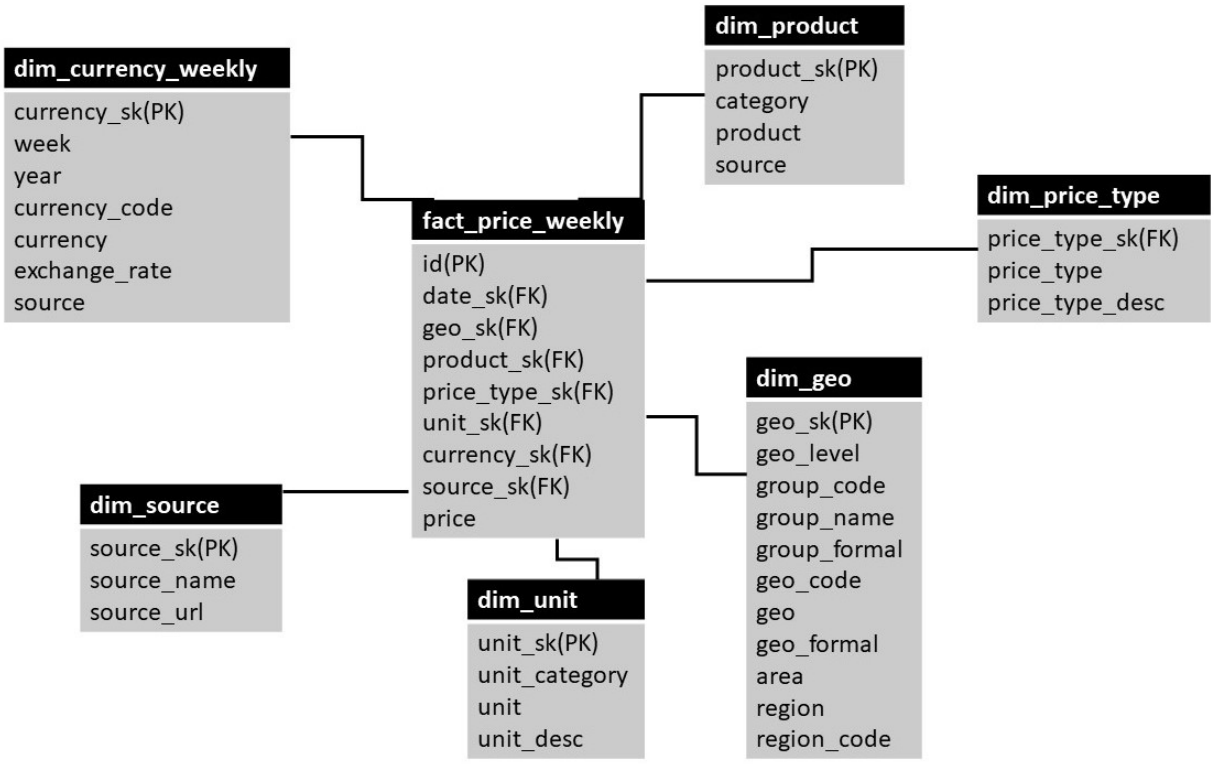

Figure 4.3: Price Weekly Data Mart

is checked to see if it is the name of a cube measure. If so, the value is inserted into that measure. For each attribute that is a dimensional attribute in the data, the DataMap is searched for the correct dimension. The foreign key for this dimensional value is extracted from the dimension and inserted into the cube.

An example of the structure of our data cubes is shown in the Star Schema seen in Figure 4.3. In this schema, the fact_price_weekly fact table contains a single measure - price - and foreign key links to 6 dimensions: dim_currency_weekly, dim_source, dim_unit, dim_geo, dim_product and dim_price_type. The dimensions are connected to the fact table by primary key-foreign key links.

Table 4.4: Data Cubes

c_id	source	measure(s)	rows	attrs	t-forms	convs
C001	statsnz	trade_weight, trade_value	942565	9	3827381	57121
C002	bordbia	slaughters	5635	4	11270	0
C003	bordbia	price	24990	5	49980	24990
C004	clal	production	913	5	2480	827
C005	dairyaus.	production	652	5	1304	610
C006	eurostat	trade_weight, trade_value	13484	10	94395	26970
C007	kepak	trade_weight, trade_value, yield, offcut_value	5000	15	3314	0
C008	pig333	price	14635	5	29272	14636
C009	statcan	trade_weight, trade_value	2559	10	15359	0
C010	usda	trade_weight, trade_value	95725	10	478625	95725

4.4 Evaluation - Sample Dynamic Data Darts

We begin by loading data to a set of 10 data cubes. Table 4.4 gives an assigned unique identifier **c_id**, the source name, the measure(s), the number of rows, the number of attributes, the number of term mappings **t-forms** and the number of measure conversions **convs** for ten data cubes. The measures are the names of canonical measures from the CATM. In the case where the measures are already expressed in the unit specified by the canonical model, no measure conversions need take place. However, if the name of the unit is mapped to a canonical term, this will still count as a dimension **transform**. In some cases, where the data has multiple measures, one may need to be converted while the other does not. Similarly, some dimensions may need to be transformed to canonical terms while others are already expressed in these terms.

These data cubes are stored in a data cube repository ready for querying. This is distinct from traditional ETL, which usually employs a more resource-heavy data warehouse, which is queried to produce data marts. In the next chapter, we assume that all of the components described thus far have been implemented and we have a working extended ETL system, namely: (i) our Common Data Model, the CATM has been imported, (ii) a data lake has been populated with several files from several data sources, (iii) Import Templates (Appendix B) and DataMaps (Appendix C)

for each have been written and added to the metabase, (iv) for a small number of these data files, they have been transformed and loaded to a set of data cubes. From that beginning data environment, we describe the components and processes that underpin our query-fulfilment system. These data cubes will also be used in several sets of experiments in our evaluations chapter.

4.5 Summary

We have established in earlier chapters that traditional ETL is unsuitable for modern-day requirements with changing data sources. In this chapter, we presented an ETL architecture that reflects traditional ETL processes but has extra features to provide a more dynamic means of integrating previously unseen data.

We detailed our main constructs in our extended ETL - the Import Templates that contain the main metadata elements for our data lake and facilitate the extraction of the data as a set of attribute-value pairs, and the DataMaps which provide a set of canonical terms and conversion rules from the canonical vocabulary for the transformation process. We have shown how these elements of our architecture facilitate the importation of previously unseen datasets by providing sets of parameters to generic processes of importing and transforming the data. These constructs are stored in a multi-model metabase which contains multiple forms of metadata, some of which have been described thus far and rest will be described in Chapters 5 and 6. We outlined our transformation process and how the transformed data is loaded to a data cube. The result of this process is a set of dynamic data cubes, i.e. data cubes loaded from both enterprise and web-based sources, that will be assumed to change frequently and will be used as a starting point for our query reuse system.

For moving onto the next chapter, we assume that the system has described thus far has been implemented. Our domain-specific Common Data Model has been initialised and we have a data lake populated with a large number of files, from multiple heterogeneous data sources. A subset of these will have undergone the

ETL process and are now stored as a set of data cubes, defined by the all cube (*). In the next chapter, we will describe our query processing methodology wherein these cubes are utilised to fulfil a query.

Chapter 5

Query Matching

Our second research question is how to make use of query reuse using dynamic data cubes. The next step in our research is to exploit the novel architecture and data cube construction presented in Chapter 4 to develop our method for processing queries for query reuse. The challenge is to select the correct cube or cubes to address the query for our query reuse system, in order to avoid queries having to be recomputed when there is a match or partial match available. In order to deliver this research question, we require a method of comparing incoming queries with existing data cubes. Thus, the cubes and queries must be in a format to make direct comparisons. We present our two main novel constructs in this chapter - the *CubeMaps* and the *QueryMaps*, which allow for queries and cubes to be compared in order to determine whether or not query reuse is possible for each query. We also present our *Cube Matrix* which facilitates the integration of partial matches, as this is a common occurrence in query reuse systems.

We begin with an overview in §5.1 to introduce the requirements needed for query processing over a dynamic ETL architecture. We then describe in §5.2 how the CubeMaps are populated from a set of existing data cubes. We also describe how the Cube Matrix is populated using the Common Data Model and the existing CubeMaps. In §5.3, we show how the Cube Matrix facilitates matching new queries with data cubes in a query reuse process, which prevents the need to re-compute a

query result from scratch.

5.1 Overview

In brief, the main components of our query processing architecture are:

- Common Data Model - The Common Data Model used in our research is the CATM, which represents the agriculture domain and is described in detail in Chapter 3. The CATM is a Constellation schema of Facts and Dimensions.
- The Data Lake - This low-overhead data repository is described in Chapters 3 and 4.
- Data Cubes - These are the result of loading the transformed data, in the form of a star schema of a single Fact and multiple Dimensions.
- CubeMaps - CubeMap are abstract representations of data cubes, containing Cube Vectors - components of the data cube.
- Queries - These are expressed by the user using a portal to the Common Data Model.
- QueryMaps - These have an identical structure to the CubeMaps but are abstract representations of queries.
- The Cube Matrix - This is a map of the facts and dimensions from the CDM and their availability in the existing cubes.

The interaction between each of these elements in response to a query is represented in Figure 5.1. A query is created by the user interacting with the CDM, which in this case is the CATM, representative of the Agricultural domain. The query is parsed as a QueryMap and can be compared side-by-side with each CubeMap to offer a possible match. In this chapter, we focus on the elements of the architecture used in our query processing - the CubeMap, which is a representation of a data

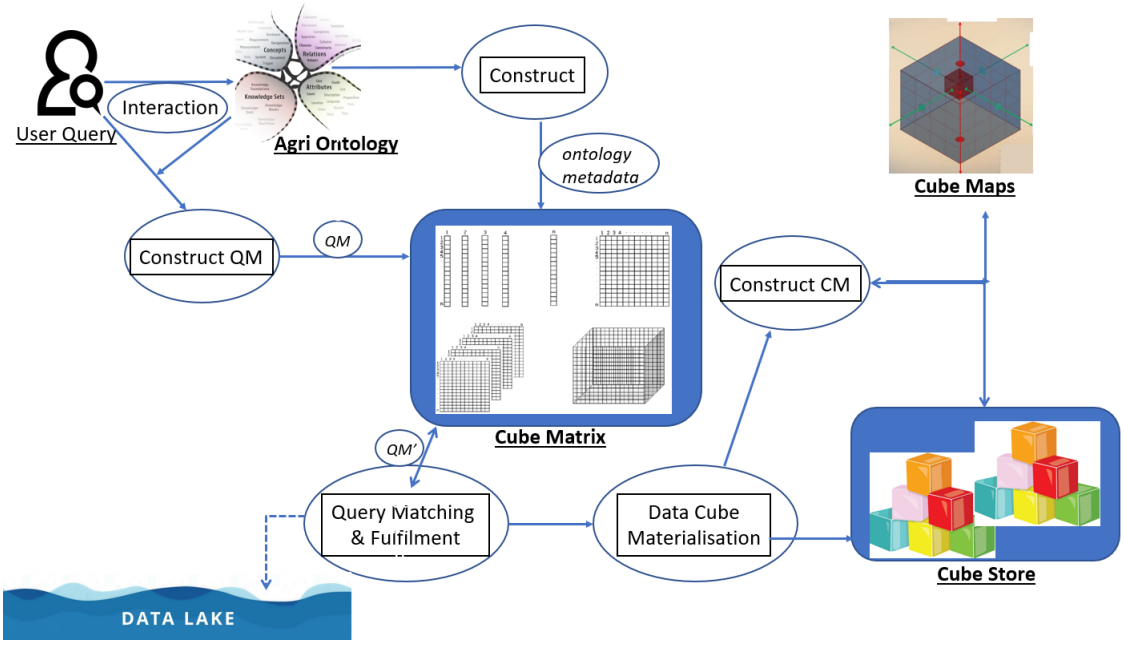

Figure 5.1: Query Processing

cube, i.e. the attributes and data types it contains; statistics on the continuous variables in the cube; as well as metadata; the QueryMap, which is a representation of a query; and the Cube Matrix, which provides a map for query fulfilment.

In Chapter 2, we describe the approach to on-demand ETL in [8] where an abstraction called a *dice* is used to capture the information available in a data cube, providing information on the suitability of the cube for answering an incoming query and determining the elements of the query that cannot be answered. A dice management process provides a map of the cubes that are currently available. Our CubeMap construct describes a data cube at the most fine-grained level of detail, and therefore is somewhat similar to a dice as in [8]. We expand this work by acquiring the key components of a query into the QueryMap, in order to perform a series of checks of the relationship between the query and the cube.

For this chapter, our methodology has the following goals:

i. Our first goal is to acquire the metadata for each cube and generate a CubeMap for each.

ii. Next, we use the Cube Matrix to show the extent to which the values of the Common Data Model are captured by the cubes in the cube store.

iii. When a query is launched, the query is fragmented, i.e. converted into a set of fine-grained requirements.

iv. We acquire the metadata of this query into a QueryMap.

v. We have a process to fetch information from the Cube Matrix on which cubes can be used to fulfil the query, and whether or not additional data is required from the lake.

To demonstrate our approach, we assume that we have acquired data from a large amount of sources and stored them in a data lake. Now, a query has been launched that resulted in the extraction, transformation and loading of 10 files from the lake, meaning we have the 10 data cubes in Table 4.4 currently in the cube store.

5.2 Acquire Cube Metadata

In order to examine the results of previous queries so that they can be inspected to see if a previous query result can be reused for the incoming query, we need to acquire and store the metadata of existing cubes in a new structure. In Definition 5.1, a CubeMap is formally defined as having a unique identifier, a set of CubeVectors, which are defined in Definition 5.2 and a Function, which is defined in Definition 5.3.

Definition 5.1. A CubeMap is a triple $CM = \langle I, CV, FS \rangle$ where: I is a unique identifier, CV is a set of CubeVectors and FS is an aggregation function.

The CubeMap contains a set of Cube Vectors, which have a set of fields that capture the metadata of the attributes in the cube plus some statistics of the measures and dates. The statistical fields fulfil the same purpose as the start and end nodes of the SchemaGuide in [60], which is to check the containment of one fragment within

another. A CubeMap has a one-to-one relationship with a data cube and a one-to-many relationship with a CubeVector. Each of the CubeVectors has a one-to-one relationship with an attribute of the cube.

In Definition 5.2, there will be a one-to-one relationship between N the name of a CubeVector, and an attribute in the data cube. T the data type will be one of three values: date, which is any attribute belonging to one of the data dimensions in the CDM; number, which is for any measure attributes such as price or weights; and string which is any dimensional attribute which is not a date. H is a Boolean that indicates whether the range of values contained in the attribute includes Nulls. If T is either a date or a number, R is the range of values contained in this attribute, specified by the min and max, and VS will be Null. If T is a string, R is null and VS is the set of unique values for this attribute.

Definition 5.2. A CubeVector is a 5-tuple $CV = \langle N, T, H, R, VS \rangle$ where: N is the name of the CubeVector, T is the data type, H is a Boolean, R is the range of values, and VS is the valueset.

In Definition 5.3, FT is the type of function e.g. sum, average. CV is the CubeVector on which the function will be performed (the T of the CubeVector must be a number). G is a Boolean that indicates whether the data will be grouped while this function is performed. If G is True, GO is the name of the attribute or attributes on which the data will be grouped.

Definition 5.3. A Function is a 4-tuple $CV = \langle FT, CV, G, GO \rangle$ where: FT is the function type, CV is the name of a CubeVector, G is a Boolean and GO is a list of CubeVectors.

5.2.1 Create CubeMap

A CubeMap is a novel construct, the closest comparable work being a SchemaGuide in [60]. We briefly describe the process of creating a CubeMap from a data cube below. In the remainder of this section, we will present the structure of a CubeMap

in detail followed by the process of constructing these from existing data cubes 5.2.1.

At a high level, the CubeMaps are created as follows. Given a data cube with a unique identifier and a set of attributes, which each has a set of values, the 6-step process has the following goals:

1. A CubeMap is initialised and named with a unique identifier for the data cube.
2. The set of attributes, i.e. the measure names and dimensional attributes, are extracted from the cube. These populate the `names` of the CubeVectors.
3. For each attribute, the data type is identified as one of three types: (i) a string, which most dimensional values will be, (ii) number, i.e. mostly used for measures - price, sales, weights, (iii) date. These populate the `types` of the CubeVectors.
4. For each of the attributes in the cube, the unique set of values for that cube is extracted. If the data type is a number or a date, the range of values, i.e. the `minimum` and `maximum`, populate the CubeMap. If the data type is a string, the unique list of values is saved as the `valueset`.
5. It is recorded whether or not each attribute contains null values. This is recorded as True or False for each CubeVector.
6. If the cube has been defined by an aggregation function (e.g. sum or mean), details of this are stored in the CubeMap.

We will use the cube referenced as C003 in Table 4.4 as a working example. A short snapshot of this data cube is shown in Figure 5.2.

Applying the 6 steps listed above:

1. A new CubeMap is initialised for cube C003.
2. The attributes of the cube are extracted (ignoring the index): `date, geo, product, unit, price`.

date	geo	price	product	unit
07-01-95	AUSTRIA	106.3	Pig	EUR/KG
07-01-95	BELGIUM	110.94	Pig	EUR/KG
07-01-95	BRAZIL	NULL	Pig	EUR/KG
07-01-95	DENMARK	97.24	Pig	EUR/KG
07-01-95	EU	104.73	Pig	EUR/KG
07-01-95	FINLAND	103.84	Pig	EUR/KG
07-01-95	FRANCE	104.77	Pig	EUR/KG
07-01-95	GERMANY	102.38	Pig	EUR/KG
07-01-95	GREAT BRITAIN	NULL	Pig	EUR/KG
07-01-95	GREECE	122.77	Pig	EUR/KG

Figure 5.2: C003 Data Cube

3. The `price` attribute has the type 'number'; the `date` attribute has the type 'date'; `geo, product, unit` all have type 'string'.

4. The minimum and maximum values for attributes `date` and `price` are extracted. The set of unique values for `geo, product, unit` are extracted into valuesets.

5. The `price` attribute contains nulls.

6. In this case, there are no aggregations (roll-ups) used to define this cube.

The resulting CubeMap instance is shown in Figure 5.3, where C003 is the identifier of the cube and the `valueset` is a link to the full set of dimensional or fact values for that attribute. The `price` measure has a missing value, hence the **has_nulls** field in the CubeMap, for the `price` attribute, is set to True, while the rest of the attributes are set to False. It can also be seen that attributes whose type is a string are linked to the list of unique attribute values, while those that are numerical or a date instead have the range of values (i.e. the **min** and **max**) populated.

The implementation of the method to create a CubeMap is found in Algorithm 4.1 in Appendix D.

In Table 5.1, we provide a condensed version of three CubeMaps: `C001, C003` and `C005`. The Cube Vectors are represented by the `CV Name, type, nulls, min` and `max` columns, while the valuesets are presented as a set. As these cubes are intended to provide a baseline for our query reuse system, all have been created with a simple

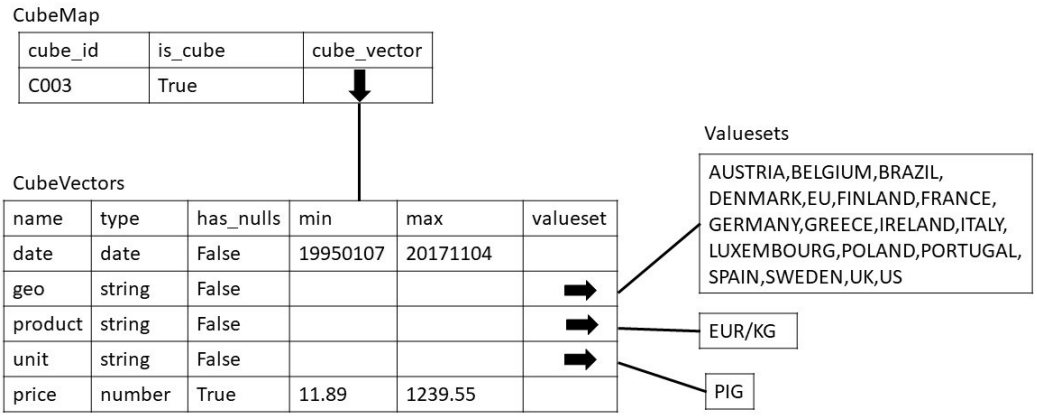

Figure 5.3: CubeMap Instance

`SELECT *` query and no aggregate functions. From this point, new incoming queries may contain such functions.

The CubeMaps are stored in the metabase along with all metadata constructs, and provide quick means of identifying which data cubes are available for answering queries, without having to perform the query on all the available data.

5.2.2 Cube Matrix

The next construct in our query processing architecture provides describes the information stored in the CubeMaps and can be used to see whether or not an incoming query can be fulfilled. The Cube Matrix is a map of the cubes available in the cube store, so that it provides both an overview of which elements of the Common Data Model are available in the current set of cubes and which are missing, as well as allows for fast access to the complete set of CubeMaps so they can be compared against an incoming query. This section will show how the Cube Matrix is populated.

At the stage of system initialisation, the Cube Matrix is created with the following fields:

- dimension - the name of the attribute as it appears in the dimension

Table 5.1: Sample CubeMaps

cm_id	**CV name**	**type**	**has_nulls**	**min**	**max**	**valueset**
C001	yearmonth	date	F	201202	201401	null
C001	reporter	string	F	null	null	{CANADA}
C001	product	string	F	null	null	{pigs}
C001	trade_weight	number	F	0	534066	null
C003	date	date	F	1995-01-07	2017-11-04	null
C003	geo	string	F	null	null	{AUSTRIA, BELGIUM, BRAZIL, DENMARK, EU, FINLAND, FRANCE, GERMANY, GREECE, IRELAND, ITALY, LUXEMBOURG, NETHERLANDS, POLAND , PORTUGAL, SPAIN , SWEDEN, UK, US}
C003	product	string	F	null	null	{pig}
C003	unit	string	F	null	null	{cent/KG}
C003	price	number	T	11.89	1239.55	null
C005	yearmonth	date	F	200701	201706	null
C005	geo	string	F	null	null	{AUSTRALIA}
C005	product	string	F	null	null	{milk, butter, smp, wmp}
C005	unit	string	F	null	null	{KG,Litres}
C005	production	number	T	5.315	39.19	null

- dimension_attribute - the name of the attribute as it appears in the dimension
- dimension_valueset - a valueset of canonical dimensional values.
- cube_id - the unique identifier of a CubeMap
- cube_vector - the name of the attribute as it appears in the CubeMap
- intersect - the subset of the valueset of the dimension that is contained within this cube.

This process is shown in the `CreateMatrix` Function in Algorithm 4.4. Following this, the `Dimension`, `dimension_attribute` and `dimension_valueset` are populated by the Common Data Model prior to the creation of any cubes or CubeMaps.

The steps to populate these fields are as follows:

1. From the Common Data Model, the list of dimension names is extracted. These populate the **dimension** field.
2. For each dimension, the full list of attributes of that dimension is extracted, which populate the **dimension_attribute** field.
3. The complete set of dimensional values, i.e. the canonical vocabulary, is added to the **dimension_valueset** field.

Following the population from the data model, the following fields are populated from the existing cubes:

- `Cube_id` is populated with the unique identifier for each CubeMap.
- `cube_vector` is populated with the name of each Cube Vector represented in a CubeMap.
- `intersect` is populated with the valueset that is the intersection between each dimension's valueset and the Cube Vector's valueset.

These processes are shown in Algorithm 4.4 in Appendix D.

Figure 5.4 shows a CubeMap with four attributes, each with a valueset, a dimension dim_A, with three attributes, each with a set of values. Within the Cube Matrix is the intersection between the valuesets of the dimension and the valuesets of the CubeMap. In this way, the Cube Matrix shows which CubeMaps contain which values of the Common Data Model.

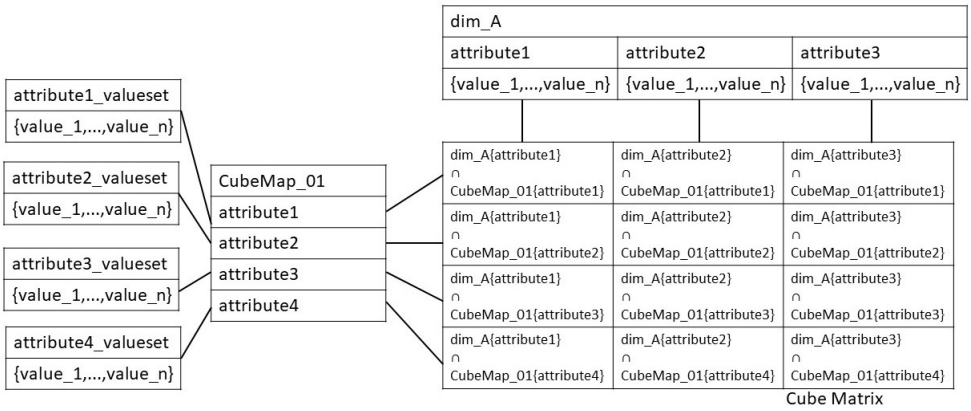

Figure 5.4: Cube Matrix

5.3 Query Reuse

The purpose of query reuse is to examine if the results of materialised queries can be used to satisfy incoming queries either in part or as a whole. The results of previous queries are stored as data cubes in the cube store and each will have an associated CubeMap.

Figure 5.5 illustrates the work flow of our query reuse process. The process begins with the expressing of a query, shown in green. The processes of checking the Cube Matrix and querying the Lake, shown in yellow, are each a multi-step process of their own, to be described later in this chapter and the next. The more specific steps shown in blue will each be given as an Algorithm. In orange are some of the possible outcomes, which will indicate a strategy for returning the resulting data cube.

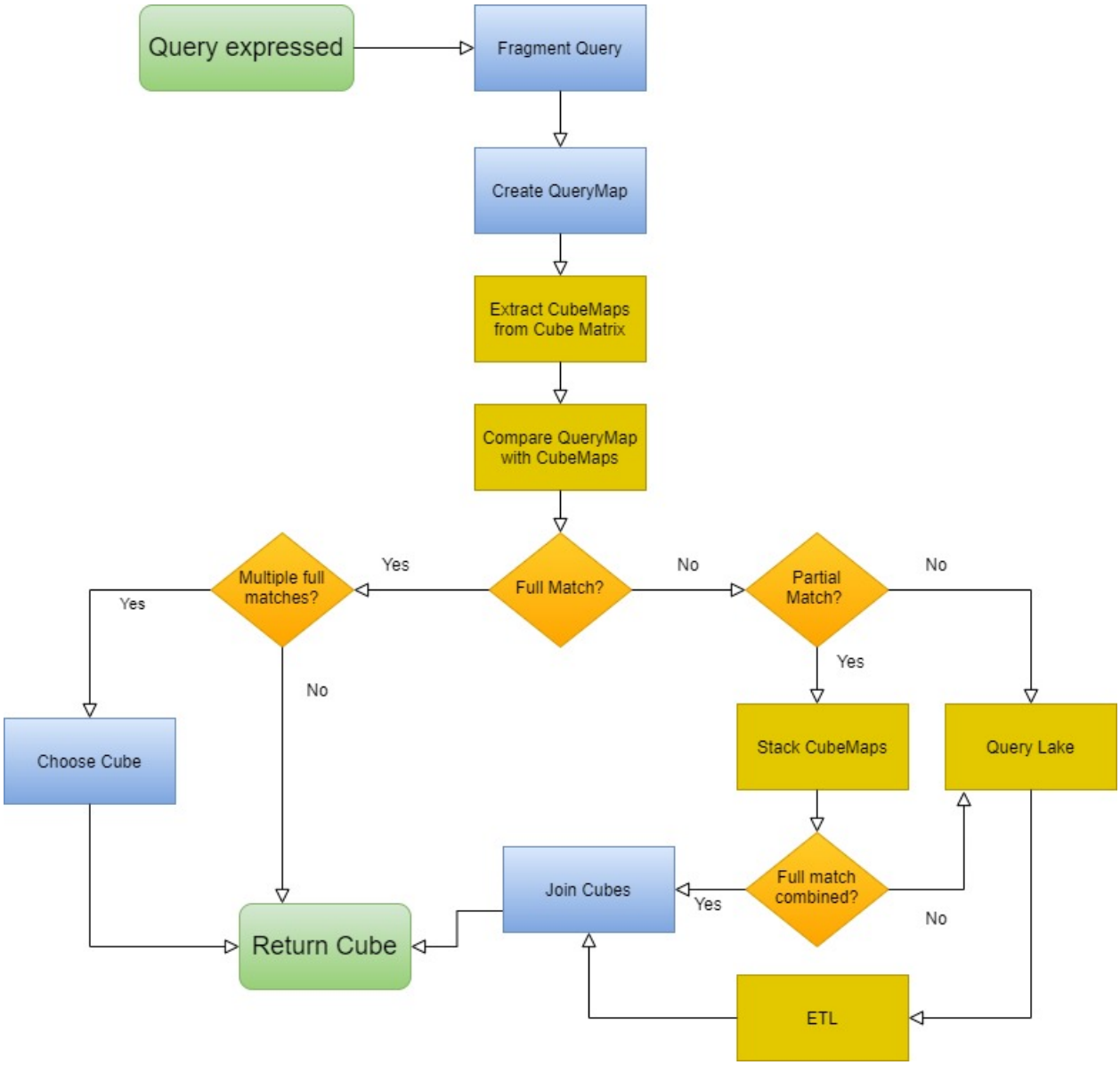

Figure 5.5: Query Processing

At a high level, the steps involved are:

i. Fragment query: The query is deconstructed into the most fine-grained elements.

ii. Create QueryMap: A QueryMap is created for the query.

iii. Search existing cubes for fragment matches: From the Cube Matrix, extract information of the existing CubeMaps.

iv. Compare QueryMap with each CubeMap to identify the fragments that can be fulfilled by the cubes in the cube store.

v. Select candidate cubes: a candidate cube is the name of a data cube that can contribute to the fulfilment of a query.

vi. If any full matches are found, partial matches may be disregarded. Otherwise:

vii. Query lake: The files that can fulfil the missing fragments of the query are selected for processing into new data cubes

viii. Return resultset: Partial matches are refined, subsets removed. The candidate cubes are joined with the new data cubes. Constraints and functions of query are applied.

There are three possible outcomes when examining if a query can be reused: 1. Full Match, 2. Partial Match, 3. No Match. Within each of these outcomes, there are a number of possible circumstances. We will show how the outcome of the matching process will determine the strategy for selecting and combining the data that matches each fragment of the query.

1. Full match - where a previous query can fulfil a new query.
    - Single full match - A single data cube can fulfil the query.
    - Multiple full matches - More than one data cube is a full match for the query.

2. Partial match - One or more cubes can partially fulfil the query.
    - There are multiple partial matches which, combined, can fulfil the query.
    - Single partial match.
    - There are multiple cubes which, combined, partially fulfil the query.
3. No match - No existing data cubes match any part of the query.

A single full match is a straightforward case of query reuse - a data cube can be reused to fulfil an incoming query. The second sub-type of full match, when there is more than one data cube which can fulfil the query, raises the obvious question of which to choose. When there are multiple matches which, when combined, will produce a full match, raises a case of the knapsack problem or subset sum problem [63], i.e. given multiple possible partial matches, how to choose the partial matches (or subsets) that provide a full match but with the smallest possible data cube. If there are fragments that remain unmatched after attempting query reuse, these are passed to the data lake. In Chapter 6, we will examine the cases where additional data is required to fulfil the query.

5.3.1 Input and Fragment Query

We will begin by providing the format in which a query must be expressed. As shown in Chapter 3, a query has four elements: a unique identifier, a set of one or more required attributes, a set of zero or more constraints and may or may not contain an aggregate function.

Therefore, a query populates the following set expression:
Q:(RA=(),RV={},F={})
Where RA is a list of attributes, RV is the constraints applied on those attributes, and F is any aggregate functions the user wishes to apply. RA will be a list of attributes found in Common Data Model. RV is a list of constraints where each constraint has an attribute and a value for that attribute, a list of possible values for that attribute, or a statement that nulls should be excluded. F has the same

properties as the Function in Definition 5.3: a function type, an attribute on which the function is to be performed (where the attribute must be a measure), a Boolean to indicate whether the dataset is to be grouped, and which attribute or attributes to group by.

The queries posed must be expressed in the terms of the Common Data Model, meaning the attributes in RA must be the names of dimensional attributes or measures from the Common Data Model, and any constraints set on dimensional values in RV must be within the set of dimensional values of the Common Data Model. In order to provide a working example, Figure 5.6 shows a prototype of a GUI which forces the user to express their queries in canonical terms. In this case, the user is specifying her required attributes and values for Query 002 from Chapter 3. The user begins by selecting one or more measures. In the example shown, the user has selected `price` from the list of possible choices. This has the effect of manipulating the list of available dimensions such that only the dimensions which are associated with the chosen fact will be available. The user has selected the dimensions `date` and `geo`.

For each selected dimension, the user has the choice to define a range of possible values or to choose between two different versions of a `select all`. On the GUI, if the `select all` option is checked, we assume that the user wishes to search for all possible canonical values for this dimension, even if there are thousands of values. Thus we call this a `greedy select all`. If the user wishes to use this option, they are instructed that this will represent a large query and may result in a longer run-time. If they wish to view all available values in the Common Data Model, they may proceed. If the user does not specify this option, we assume this to mean the system will search for the values that have been specified for other dimensions, while ignoring the values for this dimension. We refer to this as a `lazy select all`. The searches for measure values are a `lazy select all` by default unless the user specifies a range of values to search for. The user may also specify that they wish to exclude nulls from the final dataset.

In Figure 5.6, the user has chosen `lazy select all` for `date`, and is in the process

of selecting options for the `geo` dimension. Finally, the user has the same choices to narrow down the range of measure values. Here, the user may again do a `greedy select all`, allow a `lazy select all`, specify not null, or set a minimum and maximum value.

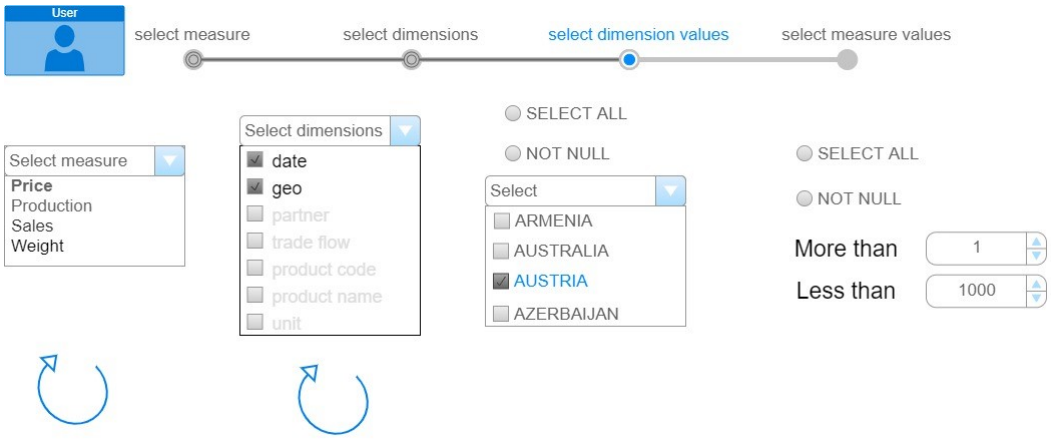

Figure 5.6: QueryMap instance from user portal

The measure and the dimensions selected by the user then populate the RA of the query. The range of values set as a constraint on `geo` populate RV, while `date` and `price` are both `lazy select all`. The user has not requested a sum, mean or other aggregations, so F is empty. Thus, the query expression is now populated in Example 5.3.1.

Example 5.3.1. Populated Q002

Q002:(RA=(date,geo,price),RV={geo:(AUSTRIA,GERMANY,CHINA)},F={})

We have seen in Chapter 2 how the fragment-based approach is often used in improving the performance of query materialisation [12, 59]. In our approach to query fragmentation, we convert a query into QF a set of attribute-value pairs. For each constraint specified in RV of the query, a fragment is the attribute of the constraint plus a single value. For each attribute of RA for which a `lazy select all` is required, a fragment is the name of the attribute with an '*' as the value. For each dimensional attribute of RA for which a `greedy select all` is required, a set of fragments are created by extracting the full set of possible canonical values for that

dimensional attribute from the Cube Matrix. This allows us to progress the existing approach to query fragmentation by creating a set of highly specific, granular queries which can be answered from multiple sources, identifying both full and partial matches. Using the query in Example 5.3.1 as a case, the output is shown in Example 5.3.2.

Example 5.3.2. Q002 Fragments
QF(Q002) = [(geo:CHINA), (geo:AUSTRIA), (geo:GERMANY),(date:*),(price:*)]

This process is shown in Algorithm 4.3 in Appendix D.

5.3.2 Create QueryMap

Queries and Cubes have an identical structure - a set of facts and associated dimensions with ranges of values. The novelty in our work on CubeMaps is extended by creating a metadata capture of a query called a QueryMap. The structure of this is identical to the CubeMap which facilitates a side-by-side comparison of the two structures.

The process of converting a query to a QueryMap is as follows. This process takes the query as input and uses the Common Data Model to supply information on which attributes are measures, which are dimensions and which are dates.

1. Extract RA, the list of the attributes required to fulfil the query. Populate the `CubeVector` field with each of these.
2. For each constraint in RV:
   (a) If it is specified that the attribute may not contain Null values, set `has_nulls` to False, otherwise set to True.
   (b) Where the clause specifies a specific set or range of values, set `min` to the minimum value and `max` to the maximum value. Populate the valueset with RV - the full set of required dimensional values.

(c) If there is no set of values specified for this attribute, set `valueset` to null.

This process is shown in Algorithm 4.2 in Appendix D.

The QueryMap of the query in Example 5.3.1 can be seen in Figure 5.7. In the resulting QueryMap, there are three CubeVectors: `date`, of type date; `geo`, of type string; and `price`, which is numerical. The query does not contain a `not null` constraint, so `has_nulls` is set to True by default. The lack of specifications for the `min` and `max` for `date` indicates an equivalent of a `lazy select all` for this attribute. If the user had specified a `greedy select all` for `date`, the full range of 73414 values available from the `date` dimension would be extracted and the `min` and `max` would be populated. The `geo` dimension has a link to a valueset containing the required values. This QueryMap, its attributes, ranges and valuesets can now be compared with those of the store of CubeMaps.

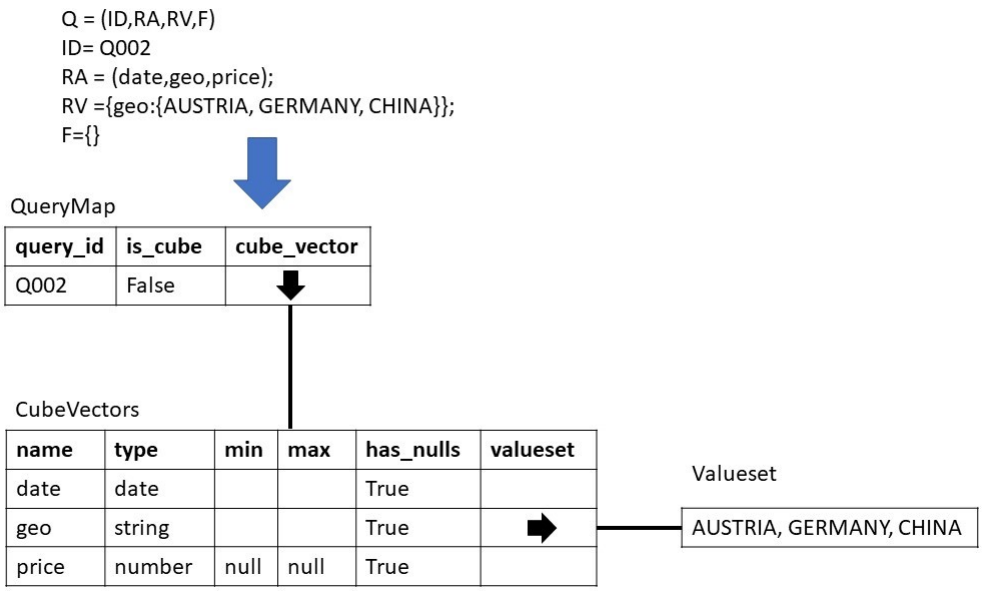

Figure 5.7: Constructing a QueryMap

5.3.3 Cube-Query Matching

We now describe the process for matching queries against a set of cubes, to provide the best query fulfilment strategy. In [60], a relationship between a cube and a query might be *is contained in*, *contains*, *equivalent* and *incomparability* (disjoint). We use similar definitions of relationships between the two constructs, wherein the identical structure of the CubeMaps and QueryMaps allows for a one-to-one comparison, to see if a query is identical to an existing cube, contained within an existing cube, has some intersection with an existing cube or is disjoint.

The strategy for matching CubeMaps with QueryMaps is (i) we begin with QF the set of query fragments, (ii) examine the relationship between the QueryMap and the set of CubeMaps and find the degree of containment or intersection between the sets of attributes and values, (iii) find $C_{candidate}$, the set of CubeMaps which can fulfil some or all of QF, (iv) split QF into QFF, the set of query fragments that can be fulfilled by $C_{candidate}$, and QFU the set of query fragments that require external data.

The first step is to determine whether there is a CubeMap which can fully fulfil, i.e. contains or is equivalent to, the QueryMap. This is to eliminate the need for redundant query processing. The `InspectMatrix` function determines whether there is a full match, one or more partial matches, or no match. At this step, we will also identify the $C_{candidate}$, the set of candidate cubes. The aim at this stage is to *greedily* select as many cubes as possible to the set of candidate cubes. We will see in Chapter 6 how we assign stricter thresholds to select the more useful matches.

The inputs to the `InspectMatrix` function are Q - a Query, CDM the Common Data Model and $Matrix$ the Cube Matrix. The query is first fragmented into QF before a QueryMap is created. This is followed by extracting the set of CubeMaps and performing a set of `CheckContainment` functions which check whether the query can be contained within an existing cube:

1. The CubeMap is disregarded if the cube is defined using an aggregation which is different from any aggregation used in the query, i.e. if the type of aggregation required in the query is the same as that in the cube or if it uses the same attributes to group by and in the same order. If any of the variables in the aggregations are different, the containment check fails.

2. For each CubeMap, the list of CubeVector names is compared with the CubeVector names in the QueryMap. The CubeMap is also disregarded if the CubeVectors do not have any overlap with RA the required attributes of the query. If there is overlap, the `CheckContainment` functions are used to determine whether the values required by the query are contained within the values of the CubeMap's valuesets and value ranges.

3. If the attribute is a measure or date: if the range of values is contained within the range of the cube, this attribute passes the containment check.

4. If the attribute is a dimension: if the valueset of the query is a subset of the valueset of the cube, the attribute passes containment.

If all containment checks pass for a CubeMap, this CubeMap is added to $C_{candidate}$ with a flag that indicates it is a full match for the query. If all containment checks fail, the cube is disregarded.

Our strategy for checking the containment of an incoming query within a cube is a methodology of three steps. For each attribute of the query:

1. If the attribute is a measure or date: if the range of values is contained within the range of the cube, this attribute passes the containment check.

2. If the attribute is a dimension: if the valueset of the query is a subset of the valueset of the cube, the attribute passes containment.

3. If there are any aggregation functions in either the cube or the query: checks are run that the type of aggregation required in the query is the same as that in the cube, that it uses the same attributes to group by and in the same order.

Algorithm 5.1 Inspect Cube Matrix

```
1: function InspectMatrix(Matrix, Q, CDM)
2:     QF ← FragmentQuery(Q)
3:     Initialise C_candidate=[]
4:     QM ← CreateQueryMap(Q)
5:     for CubeMap ∈ Matrix do
6:         CheckContainmentFunc(CubeMap, Q.F)
7:         if CheckContainmentFunc=False then
8:             Continue
9:         end if
10:        if CubeMap.CubeVectors ∩ Q.RA = ∅ then
11:            Continue
12:        else
13:            for CubeVector_i ∈ QM do
14:                if CubeVector_i ∈ CDM.measures or ∈ CDM.dates then
15:                    CheckContainmentCont(CubeMap, CubeVector_i.valueset)
16:                else
17:                    CheckContainmentDim(CubeMap, CubeVector_i.valueset)
18:                end if
19:            end for
20:            if For all CubeVectors: CheckContainment=True then
21:                C_candidate ← [CubeMap, FullMatch]
22:            else if For all CubeVectors: CheckContainment=False then
23:                Continue
24:            else
25:                C_candidate ← [CubeMap, PartialMatch]
26:            end if
27:        end if
28:    end for
29:    if C_candidate=[] then
30:        QFU ← QF
31:        QFF ← ∅
32:    else if |C_candidate, FullMatch| ≥ 1 then
33:        QFF ← QF
34:        QFU ← ∅
35:    else
36:        FindQFU(Q, C_candidate)
37:    end if
38:    ResolveQuery(QFF,QFU,C_candidate)
39: end function
```

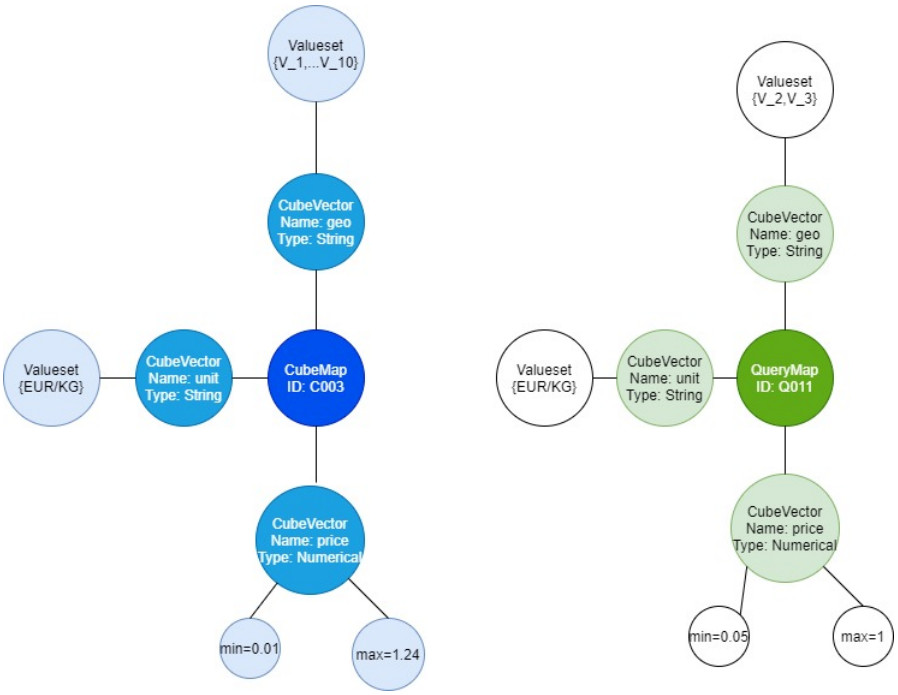

Figure 5.8: CubeMap-QueryMap Comparison

If any of the variables in the aggregations are different, the containment check fails.

An example of a cube containing a query is shown in Figures 5.8 and 5.9. In Figure 5.8, a CubeMap, in blue, is shown as having a set of CubeVectors `geo, price` and `unit`, each with a valueset in the case of CubeVectors with `type=string` and a range of values defined by the min and max for CubeVectors with `type=numerical`. A QueryMap, in green, has a set of CubeVectors also with valuesets or a range of numerical values.

In Figure 5.9, it is shown that the QueryMap can be contained by the CubeMap. It can be seen that, for each CubeVector, the set of values in the queries is equal to or a subset of the range of values in the cube. For the CubeVector `geo`, the set of values in the query are contained by the set of values in the `geo` CubeVector of the cube. For `price`, the range of values in the query are contained within the range of values in the cube. For the `unit`, the valuesets are equivalent.

The implementation of the containment checks strategy are shown in the Check

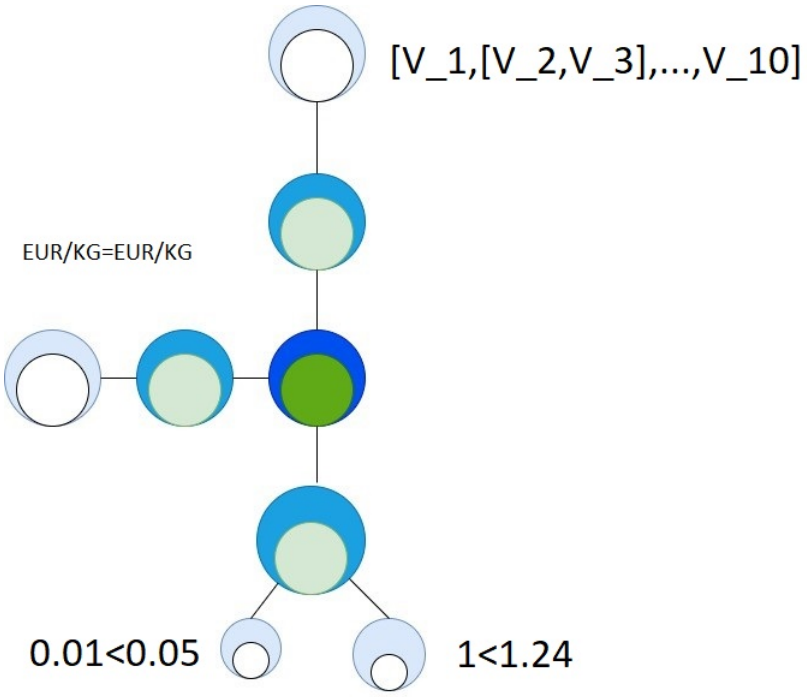

Figure 5.9: CubeMap-QueryMap Containment

Containment algorithm in Algorithm 4.5 in Appendix D.

At the end of the examination process, if no candidate cubes have been identified, the result is a No Match and all query fragments now belong to QFU. If any full matches have been found, QFU is empty. Otherwise,`FindQFU` identifies QFU, the set of query fragments that cannot be identified by the partial match. In the worst case scenario, this will be the full set of query fragments. It takes as inputs QF a set of fragments, and $C_{candidate}$ a set of CubeMaps, which it stacks to form $MultiCM$, a superset CubeMap. For each fragment, if the attribute of the fragment is not found in the CubeVector names or if the value of the fragment is not found in the valueset, it is added to QFU. At this point, we know whether all fragments of the query can be resolved using the data in the cube store, or whether there will be additional data required.

In Table 5.2, we have launched the four queries found in Chapter 3 over the set of data cubes loaded in Chapter 4. The table shows how many of the existing data cubes have been selected as candidate cubes and $|QFU|$ the number of query

Algorithm 5.2 Find Missing Fragments

```
1: function FindQFU(QF, C_candidate)
2:     Initialise MultiCM = combine(C_candidate)
3:     CM_attributes ← MultiCM.CubeVectors
4:     CM_values ← MultiCM.valuesets
5:     Initialise QFU = []
6:     for QF_i ∈ QF do
7:         if QF_i.attribute ∈ CM_attributes and QF_i.value ∈ CM_values then
8:             Continue
9:         else
10:            QFU = QFU + QF_i
11:        end if
12:    end for
13:    return QFU
14: end function
```

Table 5.2: Case study candidate cubes

Query ID	**candidate cubes**	$\|QFU\|$
Q001	10	0
Q002	6	0
Q003	7	1
Q004	10	3

fragments that will be passed to the data lake. We can see that for queries with ID Q001 and Q002, neither query found a single cube that could fully match the query, but fragment matches were found from multiple cubes which, when combined, will answer the query. For these two queries, there were no QFU. On the other hand, for queries Q003 and Q004, the combination of partial matches will require additional data from the data lake to fulfil.

With the outcome of the matching process now specified, the result is passed to the `ResolveQuery` function in Algorithm 5.3. This function can then take the appropriate action. The three things that determine the pipeline for query resolution are (i) QFF the fragments that are fulfilled by data cubes, (ii) QFU the fragments that are unmatched by any cubes, and (iii) $C_{candidate}$ the cubes selected as matches.

If QFU is an empty set and QFF is not, and there is only one candidate cube, this is a single full match and the cube can be reused for this query. If there are multiple full matches, the cubes that can answer the query are passed to a function to choose

Algorithm 5.3 Resolve Query

```
1: function ResolveQuery(QFF, QFU, C_candidate)
2:     if QFF ≠ ∅ and QFU = ∅ and |C_candidate| = 1 then
3:         MaterialiseCube(C_candidate)
4:     else if QFF ≠ ∅ and QFU = ∅ and |C_candidate| > 1 then
5:         ChooseCube(C_candidate)
6:     else if QFF = ∅ then
7:         QueryLake(QFU)
8:     else if QFF ≠ ∅ and QFU ≠ ∅ then
9:         QueryLake(QFU)
10:    end if
11: end function
```

the best possible match.

If all the fragments can be fulfilled by combining a number of partial matches, the cubes that can partially answer the query are joined into a multi-source cube, which is returned as the query result.

If there are query fragments that cannot be fulfilled by the existing data cubes, these fragments are now passed to the lake for querying. This processes of querying the lake, of selecting from multiple matches and of materialising the data cube will be described in Chapter 6.

5.4 Summary

In this chapter we have shown the architecture and process for our query re-use. We have seen how our system components - CubeMaps, QueryMaps and the Cube Matrix - are set up and populated, then how they are used to reuse previously fulfilled queries, i.e. cubes, to fulfil new incoming queries.

We have presented how we examine the store of data cubes for query fulfilment and identify a full match, partial match or no match, depending on the extent to which the data to fulfil the query comes from reuse of the cubes. We have shown some of the key functions that identify the ETL strategy used to fulfil the query, and how cubes that contain the results to incoming queries are identified and selected.

5.4.1 Case study summary

Throughout this chapter we have demonstrated key stages of the process. In Figure 5.2 and 5.3, we show the gathering of the metadata of a cube into a CubeMap. In Table 5.1, we provide a brief summary of the CubeMaps for all ten of our data cubes currently in the cube store. From this point, we assume that a user has entered a query, demonstrated in Figure 5.6, resulting in the query seen in Example 5.3.1. This first undergoes fragmentation as seen in Example 5.3.2. Figure 5.7 shows the resulting QueryMap. Table 5.2 shows the set of candidate cubes selected by the containment checking process. These cubes and the remaining unfulfilled fragments now move onto the next stage.

In the next chapter, we will show our on-demand ETL, which uses the architecture from both this and the previous chapter. This will include the process of querying the data lake to fulfil the queries when they cannot be fulfilled using the existing data cubes.

Chapter 6

On-Demand ETL

In previous chapters, we introduced the components of our ETL architecture and dynamic data marts (Chapter 4), and our query reuse methodology (Chapter 5). In this chapter, we will show how these systems are used to produce on-demand ETL solutions. This is our final research question, to show that our approach to lake querying can successfully fulfil the gaps in the query after the query reuse process. We use the data cubes created by our processes previously demonstrated, and the lake metadata constructs, to examine whether on-demand ETL can successfully fulfil queries in conjunction with our query reuse methodology.

In §6.1, we show our process for identifying the sources in the data lake that can answer the query, our method of query re-writing and how data from the lake is extracted for transformation and loading. In §6.2, we show how the data cubes are returned to the user, how they are joined when necessary and how we go about selecting a result when there is more than one possible way to fulfil the query.

At this stage in the query fulfilment process, we assume that the user has launched a query, interacting with the CATM so that the query is expressed in canonical terms. The query has been compared with the existing CubeMaps by way of the Cube Matrix, and QFU a set of fragments have been identified as requiring lake data to fulfil. The fragments that are not fulfilled at the end of this process, we

assume cannot be fulfilled using the data available to our system.

As an overview of the methodology presented in this chapter, we provide the following set of steps:

i. Select lake sources - QFU is used to identify sources from the data lake as matches, $F_{candidate}$.

ii. Process lake data - $F_{candidate}$ are converted to data cubes and included in $C_{candidate}$.

iii. Select fragment matches - $C_{candidate}$ undergo a filtering process to identify the most useful cubes.

iv. Combine fragment matches - The remaining cubes are integrated.

v. Materialise resultset - The resulting data cube undergoes a post-processing step we call Materialise to apply the constraints and aggregations required by the query. The final resultset is returned.

6.1 On-Demand Query Fulfilment

Our on-demand query fulfilment begins with the selection of the files in the data lake that can provide a match for the fragments in QFU. In this section, we present two stages in this methodology:
(i) Identify lake files - We describe our method for selecting the files containing fragment matches for QFU from a large repository of files stored in their disparate vocabularies in the data lake.
(ii) Process lake files - We show how we use our extended ETL architecture to process the selected files into a set of data cubes.

Our methodology for achieving this goal is influenced by the Lazy ETL approach found in [46] and in [45], wherein their methodology is to have separate loading strategies for metadata and actual data. On acquiring data, metadata is loaded

upfront while the loading of actual data is delayed until required for a query. The benefit of this is in allowing the metadata to act as an interface for the actual data, offering quick results over querying the data as well as the correct files being selected. When a query is launched, the metadata provides the information to the Lazy Extraction process of which data files are required to fulfil the query. The worst case scenario is that the query requires all the data available to be loaded, the best case scenario is that the data has already been loaded as a previous query. This is similar to our approach to querying the data lake, in that the metadata constructs created upfront at the point of data acquisition will supply the information to the file selection process, although we do not load this metadata to a data warehouse. Instead, these metadata constructs - the Import Templates and DataMaps - now provide an interface to the data lake, which facilitates the process of identifying the correct files to undergo the ETL process. In this section, we present our methodology for identifying the files with the required attributes and values using these constructs.

6.1.1 Identify Lake Files

From the DataMaps, the set of values in the `standard_term` field is extracted from each. For any DataMap that contains the attributes and values in the query, the data file with which it is associated is flagged for querying. Using the DataMaps, the query is transformed from the canonical vocabulary to the local set of terms, so the query can then be used to directly query the data lake. Therefore, no processing need be done on the data until it has been selected for materialising as a data cube.

We first give an overview of our approach to select the files from the lake that can fulfil the missing query fragments. Algorithm 6.1 demonstrates this methodology. Similar to the process of identifying candidate cubes, the aim is to greedily gather as many potential matches as possible.

1. The state space is first reduced by identifying from the Import Templates which files have already been loaded to a cube, and eliminating these. Those that have been loaded may be disregarded as they will have already been

checked for matches in the query reuse system. This leaves a set of files that have not yet been loaded to a cube environment, and only these will be examined further.

2. Of the remaining files, the DataMaps are searched. The `standard_term` field for each DataMap is extracted.

3. If any the fragments in QFU are present in the `standard_term`, this is a candidate file. The list of candidate files is a set of 2-tuples $\langle dm, QFU_i \rangle$ where dm is the name of a file and QFU_i is the list of query fragments that are a match in this file. This set of candidate files are returned to `QueryLake`.

4. The list of candidate files is filtered to find the best matches. For each file:

   (a) the function checks whether the matched fragments by this file have already been found. If so, this file is disregarded. Otherwise:

   (b) The matching fragments for this file are translated to the local vocabulary of the file.

   (c) The function checks if the attributes of the query are in the file as well as if the values in the query are found under the correct attribute. If so, the query fragment is appended to the processed_fragments so it will not be processed a second time.

With the query being expressed in Common Data Model vocabulary, it is unsuitable for querying the files in the data lake, which are in their native terms. Therefore, the query is rewritten so that it can be launched on the lake files. This is seen in Algorithm 6.3 where QFU is the unfulfilled query fragments and f_d is a DataMap. This process is used for each individual file that will be used to fulfil the query, as each file will have a different set of native source terms. The query will be expressed in the terms of the CDM so will therefore be found in the `standard_term` field of the DataMap. For each term in the query, an entry is found in the DataMap where the `standard_term` matches the term. The `source_term` from that entry is extracted and replaces the term in the query. The transformed query can then be returned to the `QueryLake` function.

Algorithm 6.1 Find Lake Files

```
1: function FindLakeFiles(QFU, DM, IT)
2:     Initialise files = DM
3:     Initialise skip_files
4:     for i ∈ IT do
5:         if i.Loaded = True then
6:             skip_files = +i.data_address
7:         end if
8:     end for
9:     files = files − skip_files
10:    Sort files descending
11:    Initialise candidate_files
12:    for dm ∈ files do
13:        standard_term ← dm.standard_term
14:        for QFU_i ∈ QFU do
15:            if QFU_i ∈ standard_term then
16:                candidate_files =+ {dm, QFU_i}
17:            end if
18:        end for
19:    end for
20:    return candidate_files
21: end function
```

Algorithm 6.2 Query Lake

```
1: function QueryLake(QFU, DM, IT, R, Cubes)
2:     F_candidate ← FindLakeFiles(QFU, DM)
3:     Initialise processed_fragments=[]
4:     for c ∈ F_candidate do
5:         if c[fragments] ⊄ processed_fragments then
6:             QFU′ ← TranslateQuery(c[fragments], c[file])
7:             Open c[file]
8:             Get file.column = fragment.attribute
9:             if fragment.value ⊄ file.column then
10:                F_candidate = F_candidate − c
11:            end if
12:            processed_fragments = processed_fragments + c[fragments]
13:        end if
14:    end for
15:    Cubes ← ETL(F_candidate, DM, IT, R)
16:    if |Cubes| > 1 then
17:        JoinCubes(Cubes, C)
18:    end if
19:    MaterialiseCube
20: end function
```

Algorithm 6.3 Query Translation

```
1: function TranslateQuery(QFU, f_d)
2:     for a ∈ QFU.attributes do
3:         a' ← f_d.source_term(a)
4:     end for
5:     for v ∈ QFU.values do
6:         v' ← f_d.source_term(v)
7:     end for
8:     return QFU.a', QFU.v'
9: end function
```

We have seen in the previous chapter that query Q003 will require lake data to be fulfilled. Example 6.1.1 shows (i) QFU for Q003, (ii) the candidate file found which can fulfil QFU and (iii) the translation of QFU into the source terms of the candidate file. Of the existing data cubes in the system, none contained trade data reported from Australia. However, in the data lake there is a file from UN Comtrade, which reports Australian trade data, among others.

Example 6.1.1. Query translation

(i) QFU={reporter:AUSTRALIA}

(ii) Candidate lake file = comtrade.csv

(iii) QFU' = {reporter:Aus.}

6.1.2 Process Lake Data

At this point, we have a set of candidate files that can satisfy each of the fragments in the query that can be fulfilled by the data available to our system. The next step is to bring these data files inside the data cube environment in such a way that the matches found in query reuse and the matches found in the lake can be treated the same way. The ETL processes found in Algorithms 4.1 to 4.4 are used. These processes are managed by Algorithm 6.4.

For each of the files to be processed, the data is extracted as a set of attribute-value pairs, which are passed to the Transform function, along with the DataMaps and ruleset which that function takes as input. A new data cube is initialised and the

transformed data is passed to the function to populate the cube. In lines 5, 8 and 11, the Data Mart Flags of the Import Template are updated.

Algorithm 6.4 Process Lake Data

```
1: function ETL(F_candidate, DM, IT, R)
2:     for FC_i ∈ F_candidate do
3:         (attributes, values) = ExtractFromLake(IT, FC_i)
4:         Set IT.Queried = True
5:         (attributes', values') = Transform((attributes, values), DM, R)
6:         Set IT.Transformed = True
7:         Set C = InitialiseCube(CDM, IT)
8:         PopulateCube((attributes', values'), DM, C)
9:         Set IT.Loaded = True
10:        CreateCubeMap(C)
11:        C_candidate = +C
12:    end for
13:    return C_candidate
14: end function
```

6.2 Data Cube Materialisation

When the data has been selected from the lake file, it has now undergone the Extraction, Transformation and Loading processes described in Chapter 4, and will be a set of data cubes in the cube store. In this section, we provide our methodology to:

(i) Select between multiple possible full or partial matches.

(ii) Join multiple partial matches, whether from cubes or the data lake or both.

(iii) Apply constraints to the resulting multi-source cube and returning the results to the user.

6.2.1 Select Fragment Matches

More than one match can be returned by the matching process. The challenge is to design an approach to selecting the best possible combination of fragments to fulfil the query. We have two methods for selecting between multiple candidate matches, depending on whether we are selecting between multiple full matches or multiple

partial matches. The first method requires low overhead processing but delivers faster results. The second requires a method to selectively remove matches that are a subset of another match as well as a method to join the cubes that remain.

6.2.1.1 Full Containment Match Selection

If all containment tests pass for more than one cube, there are multiple full matches and the aim is to find the most efficient data cube to select as the result. The metrics used are the number of dimensions and the cube size. All cubes found to be full matches will contain all the dimensions required by the query, but may have additional dimensions. For each cube, if the number of dimensions is less than the number of dimensions of the smallest cube found so far, this cube becomes the smallest cube. If the two cubes have the same number of dimensions, we calculate the total cube size in each cube by multiplying the number of columns by the number of rows, and select the smaller of the two. This is demonstrated in Algorithm 6.5.

Algorithm 6.5 Select Full Match

Input: Set of data cubes
Output: A data cube

```
1: Initialise Cube_m                                          ▷ minimum cube
2: for cube_i ∈ cubes do
3:     Get cube_i.dims
4:     if Cube_m.dims > cube_i.dims then
5:         Cube_m = cube_i
6:     else if Cube_m.dims = cube_i.dims then
7:         Get cube_size = |cube_i.rows| * |cube_i.dims|
8:         if cube_i.cube_size < Cube_m.cube_size then
9:             Cube_m = cube_i
10:        end if
11:    end if
12: end for
```

6.2.1.2 Partial Containment Match Selection

When there are multiple partial matches such that there is more than one possible way to combine the matches to fulfil the query, this is an example of the knapsack

problem, more specifically the subset-sum problem [63]. This problem refers to finding the subset of a set of elements that combine to form the desired outcome, where there is a relationship between the benefit and cost of each element. In programming problems, the desired outcome is usually a single value; in our case, the desired outcome is the set of attributes and values that resolve the query. In our case, we are not attempting to materialise all available partial matches. Therefore, we need to use some heuristics to select the cubes to be combined. Typical approaches to this problem generally involve recursively creating pairs of subsets and eliminating the one that does not create the desired outcome. Thus, our methodology is to examine each partial match compared to another to see which ones are subsets of the other, and which ones satisfy a constraint of the query.

We assume at this point that we have $C_{candidate}$ a set of possible cubes, that none of these cubes are a full match. Our strategies are (i) to remove any candidate cubes where the contribution of that cube to the query is a subset of the contribution of another cube to the query, and (ii) to remove any candidate cubes that do not meet any of the constraints of the query. If two cubes contribute the same set of constraints, we defer to selecting the smaller cube by computing cube size as seen in Algorithm 6.5.

The `RemoveByConstraint` function examines the CubeMaps associated with each cube in set of candidate cubes and the RV of the query. For each CubeMap, the function checks two criteria: (i) that the measures found in the query overlap with the measures in the cube. If not, the CubeMap is removed; (ii) the intersection between the dimensions in the CubeMap and the dimensions searched for in RV. For each of these matching dimensions, the overlap between the valuesets is checked to see if it is above a certain threshold (or over 0 by default). If so, it is removed from the list of candidate cubes.

The `RemoveSubsets` function is used to discard any candidate cubes for which the contribution is a subset of the contribution of another cube. The cubes are divided into pairs and CubeVector names of each pair are extracted, where the CubeVector names are attributes required in RA of the query. Next, the CubeVector lists of the

Algorithm 6.6 Select Partial Matches

```
1: function RemoveByConstraint(cubes, Q.RV)
2:     for cube ∈ cubes do
3:         if RV.measures then
4:             Get cube.measures ∩ RV.measures
5:             if cube.measures ∩ RV.measures = ∅ then
6:                 Remove cube
7:             end if
8:         end if
9:         match_dims ← cube.dimensions ∩ RV.dimensions
10:        for dim ∈ match_dims do
11:            Get cube.valueset
12:            Get RV.valueset
13:            if cube.valueset ∩ RV.valueset = ∅ then
14:                Remove cube
15:            end if
16:        end for
17:    end for
18: end function
```

two cubes are compared to see if one is a *proper* subset of the other, i.e. a subset which is not of equal length [75]. The lengths of the CubeVector lists are compared and, if they are the same length, the one with the smaller number of overall cells is selected. Otherwise, the subset cube is removed from the list of cubes. Finally, the function returns the list of cubes of which none are a subset of the other. At this stage, we have reduced the number of potential matches from a *greedy* selection of any fragment matches, to a set of cubes which each has data contributing to the query, and the contribution of one cube is not a subset of the contribution of another.

In Table 6.1, we show how the candidates cubes found to fulfil each query, shown in Table 5.2, have been narrowed down by these functions. The `verification` column shows the result of a manual check that the cubes that remain in the set of candidate cubes fulfil the criteria as partial matches, i.e. each should contain overlap with the query, none should be eligible as a full match, and none should be a subset of another. Initial results showed that, for query Q003, one cube was retained in the list of cubes that should have been eliminated as a subset, which

Algorithm 6.7 Select Partial Matches

```
1: function RemoveSubsets(cubes, Q.RA)
2:     for i ∈ 0, ..., |cubes| do
3:         for j ∈ 1, ..., |cubes| do
4:             Cube_a ← cubes[i]
5:             Cube_b ← cubes[j]
6:             CV_a ← Cube_a.CubeVectors ∩ Q.RA
7:             CV_b ← Cube_b.CubeVectors ∩ Q.RA
8:             if CV_a ⊂ CV_b then
9:                 if |CV_a| = |CV_b| then
10:                    ChooseSmallCube((Cube_a, Cube_b))
11:                else
12:                    cubes = cubes − Cube_a
13:                end if
14:            else if CV_b ⊂ CV_a then
15:                if |CV_a| = |CV_b| then
16:                    ChooseSmallCube((Cube_a, Cube_b))
17:                else
18:                    cubes = cubes − Cube_b
19:                end if
20:            end if
21:        end for
22:    end for
23:    return cubes
24: end function
```

Table 6.1: Case study candidate cubes selected

Query ID	**candidate cubes initial**	**candidate cubes post-selecting**	**verification**
Q001	10	2	correct
Q002	6	2	correct
Q003	7	5	adjusted
Q004	10	7	correct

was rectified by a correction in the `RemoveSubsets` function.

6.2.2 Combine Match Fragments

Having now narrowed down the list of candidate cubes to only those that satisfy constraints and those that are not a subset of another, we now move onto the process of joining the fragment matches to fulfil the query insofar as it is possible to be fulfilled with the data within the cube store, including those which have been extracted from the data lake. The challenge at this point is to create a method of combining these fragment matches. It must be able to intelligently integrate a number of data cubes without the function knowing how many cubes to be integrated nor on which attributes to join the data cubes. We need to be able to do this in such a way that all the data cubes nominated as candidates for this process are joined - none should be omitted unless they have already been found to be a subset of another. Our fragment integration process needs to arrive at a complete plan for combining the candidate cubes, without any prior knowledge of the cubes pre-programmed, and avoiding the user needing to supply information.

Our approach is to create an *Integration Plan* to combine the current set of cubes into a single cube, which is the resultset which will undergo post-processing and returned to the user. The integration plan is checked for completeness in a separate process of checks, before being applied. In order to build an integration plan, we refer to a data cube as a *Node*. The nodes will be joined by finding the *Link* between nodes, where the link is the set of attributes that are shared by any two nodes that are also required by RA. The Cube Matrix is used to provide this information. For

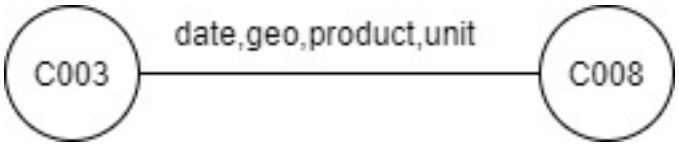

Figure 6.1: Integration Links

example, in Figure 6.1, the cubes C003 and C008 can be joined by their shared attributes date,geo,product,unit.

The cubes will be joined by the `JoinCubes` function shown in Algorithm 6.8. For each CubeMap, a node is created and nodes are passed to an `AddLink` function in pairs to determine the link between two nodes. The set of links are passed to a function to create an integration plan, or set of steps, to combine all nodes.

Algorithm 6.8 Join Cubes

```
1: function JoinCubes(cube_ids, Q.ID)
2:     Initialise Links =[]
3:     for i ∈ 0, ..., |cubes| do
4:         for j ∈ 1, ..., |cubes| do
5:             Cube_a ← AddNode(cubes[i])
6:             Cube_b ← AddNode(cubes[j])
7:             link_attrs ← AddLink(Cube_a, Cube_b)
8:             Links = Links + (Cube_a, Cube_b, link_attrs)
9:         end for
10:    end for
11:    multi-cube ← CreateIntegrationPlan(IP)
12:    MaterialiseCube(multi-cube)
13: end function
```

The next task is to identify the attributes on which to join each of the cubes. The `AddLink` function finds the set of shared attributes between two cubes. It checks the attributes of each node, ignoring the measure names, and finds the attributes in common between the two cubes. There are a series of error checks at this point. If the two cubes are identical, if one of the cubes is empty, or if no attributes are found in common, this represents some error that has occurred at an earlier point in the process. If this is the case, the cubes cannot be joined and the user must perform some sort of intervention such as removing an empty cube. Otherwise, a link is returned that contains the two nodes and a set of common attributes. These

are then passed to the `CreateIntegrationPlan` function.

Algorithm 6.9 Add Node and Links to Integration Plan

```
1: function AddNode(cube_id, Matrix)
2:     cube ← Matrix.cube_id
3:     return cube.CubeVector
4: end function

5: function AddLink(Cube_a, Cube_b)
6:     cube_a_attrs ← Cube_a.CubeVectors
7:     cube_a_attrs = cube_a_attrs − CDM.measures
8:     cube_b_attrs ← Cube_b.CubeVectors
9:     cube_b_attrs = cube_b_attrs − CDM.measures
10:    try
11:        link_attrs ← cube_a_attrs ∩ cube_b_attrs
12:    catch Errors
13:        if link_attrs = ∅ then
14:            User Intervenes
15:            return
16:        else if Cube_a = Cube_b then
17:            User Intervenes
18:            return
19:        else if Cube_a = ∅ or Cube_b = ∅ then
20:            User Intervenes
21:            return
22:        end if
23:    end try
24:    return (Cube_a, Cube_b, link_attrs)
25: end function
```

The `CreateIntegrationPlan` function in Algorithm 6.10 identifies the steps required to combine all the nodes. At a high level, these steps are:

1. Test Integration Plan for completeness: The integration plan is first checked for completeness. The `TestForCompleteness` function performs a depth-first search to show that there are links between all nodes and thus, the plan as a whole can be integrated. If not, the user must intervene.

2. Consolidate links into new nodes: Each of the links is used to create a new node, where each new node will contain $\langle node_a, node_b, new\_node, link\_attrs \rangle$ where $node_a$ and $node_b$ are the previous nodes of the link, new_node is a temporary name for the combined node, and $link\_attrs$ is the set of attributes

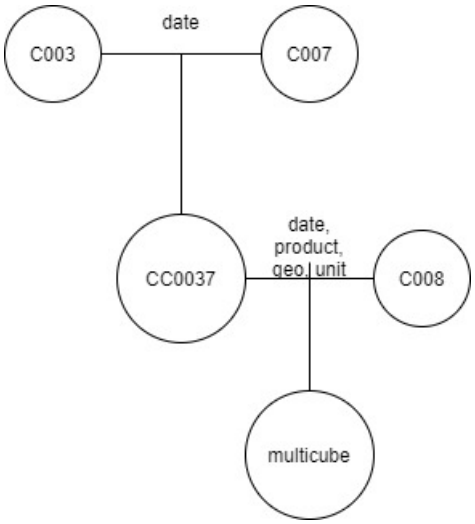

Figure 6.2: Integration Plan for Q001

to join $node_a$ and $node_b$.. Following this, the link is removed. This is performed recursively until all links have been consolidated.

3. Return results. Once all links have been consolidated, the resulting data cube is passed to the post-processing stage.

Figure 6.2 shows the integration process between the three cubes required to produce the multi-cube used for query fulfilment. Cubes C003 and C007 are joined on the Link ['date'], the result of which is joined with cube C008 on the Link ['date', 'product', 'geo', 'unit'].

6.2.3 Materialise Results

The final step in our methodology is the post-processing of the resultset. This occurs because the multi-cube returned from the Integration Plan will likely contain data that is extraneous to the requirements of the query. Thus there is a process of removing the extraneous columns to only those required by RA, then filtering the remaining columns to those specified in RV, before finally applying any aggregations defined in F. For example, for query Q002, at this point the `geo` column will be narrowed down to the valueset ("AUSTRIA","GERMANY", "CHINA").

Algorithm 6.10 Create Integration Plan

```
1: function CreateIntegrationPlan(Links)
2:     TestForCompleteness(Links)
3:     if TestForCompleteness=True then
4:         Initialise IntegrationPlan
5:         for link ∈ Links do
6:             new_node = link.node_a + link.node_b
7:             IntegrationPlan+ = (link.node_a, link.node_b, new_node, link.join_attrs)
8:             Links = Links − link
9:         end for
10:    end if
11:    Initialise multi-cube
12:    for i ∈ IntegrationPlan do
13:        cube_a ← i.node_a
14:        cube_b ← i.node_b
15:        i.new_node ← cube_a + cube_b on i.join_attrs
16:        multi-cube =+ new_node
17:    end for
18:    return multi-cube
19: end function

20: function TestForCompleteness(Links)
21:    nodes ← Links.nodes
22:    Initialise visited_links=[]
23:    for e ∈ Links do
24:        if e.node_1 not in visited_links then
25:            visited_links =+ e.node_1
26:        end if
27:        if e.node_2 not in visited_links then
28:            visited_links =+ e.node_2
29:        end if
30:    end for
31:    if |nodes| = |visited_links| then
32:        return True
33:    else
34:        return False
35:    end if
36: end function
```

For query Q003, the `reporter` column will first be filtered to the valueset ("AUSTRALIA","UNITED STATES","CANADA","IRELAND","FRANCE","SPAIN"), before filtering by the value `flow`='export'. If the result of this filtering process is an empty cube, the cube from the previous step is returned to the user. This may happen if, for example, the cube can complete an outer join on shared attributes but the valuesets are mutually exclusive. Returning the unfiltered cube to the user allows them to gain access to the data they need even if there may be extraneous columns and values.

The `MaterialiseCube` function takes as input *cube_id* the identifier for a single data cube, and Q the query. The function begins by saving a temporary version of the data cube. The function continues with a working copy and applies constraints. The cube is then narrowed down by the attributes that were specified in the RA of the query. If RV is not empty, the relevant valuesets are applied to each column in the order in which they are specified. Finally, the aggregate functions are applied, if any. For example, if the function type is `sum`, the measure in the data will be summed before returning the cube.

6.3 Summary

We have now presented our methodology for an on-demand ETL architecture which supports query reuse of a set of dynamic data cubes. In previous chapters, we presented our methodology for extended ETL and data cube reuse. In this chapter, we presented how our system deals with the case where the system requires data outside of the data cube environment to fulfil a query. The main processes presented were how to select files in the data lake, how to rewrite our queries to query the data lake, and how to combine lake data with cube data. We presented our approach to selecting the best query fulfilment strategy when there are multiple full matches or partial matches available.

Algorithm 6.11 Materialise Cube

```
1: function MaterialiseCube(cube_id, Q)
2:     Cube_t ← cube_id
3:     Cube ← cube_id
4:     Cube = Cube[Q.RA]
5:     if Q.RV ≠ None then
6:         for ri ∈ RV do
7:             col ← rv.key
8:             col.valueset ← rv.valueset
9:         end for
10:    end if
11:    if |Cube| = 0 then
12:        return Cube_t
13:    end if
14:    if Q.funcs ≠ None then
15:        attribute ← funcs.attribute
16:        function ← funcs.func_type                ▷ e.g. sum, mean
17:        apply function(attribute)
18:    end if
19:    ConstructCubeMap(Cube)
20:    return Cube
21: end function
```

6.3.1 Case study summary

In this chapter we have shown the results of key stages of the lake querying and post-processing of the resultset. In Example 6.1.1 we have demonstrated the query translation of query Q003 as well as the selecting of a file that provides a match for the missing query fragments. In Table 6.1, we show the output of the filtering process conducted on the candidate cubes. Finally in Figure 6.2, we show the integration plan of a number of partial matches.

In the next chapter, we will show our evaluation methods used at various stages in building this system.

Chapter 7

Evaluation

The goal of the research presented in this dissertation is to deliver an on-demand ETL methodology, which can use a set of dynamic data cubes constructed from both web and enterprise data and reuse these cubes for new incoming queries, as well as supplement these cubes with data from outside of the cube environment when necessary. There are many elements of this system that require an evaluation. When considering the use of standard metrics for evaluating ETL [99], our main focuses were to evaluate the data quality and performance. We observed that much of the existing research in the area of On-Demand ETL focused on performance as their main metric, as the small number of structured sources used in our comparable works meant a lower level of risk from the point of view of data quality, with the exception of [111] on account of the probabilistic nature of their work. However, we wished to also examine the fitness for use of the final resultset for each of the case study queries and therefore examine additional indicators of the quality of the results as well as time taken to produce them. We also categorised the errors found during the ETL process based on our observations of the errors.

In order to ensure our process to construct our dynamic data cubes was successful, we completed a query-based evaluation on our ETL architecture. The purpose of this is to ensure that a query run on the data cubes produces the data expected, based on the source data. This is presented in Section 7.1. The task of automating

the creation of the DataMaps required its own validation process to investigate the possibility of a loss of accuracy in favour of speed. Our experiments for this element of the system are presented in Section 7.2. Finally, the evaluation of our on-demand ETL solution can be found in Section 7.3. All tests were carried out on an Intel desktop (3.4 GHz, 32 GB RAM, Windows 8-64 bit) and used Python v2.7 and MySQL 5.7.18.

7.1 Dynamic ETL

In this section, we present our evaluation of the architecture that we have designed to answer RQ1, to create dynamic data cubes. The approaches most similar to ours in terms of the problem space, such as [3,17,101], generally demonstrated their work using case studies, as we have done in Chapters 4-6. However, we used a query-based evaluation in order to validate our ETL workflow in terms of the data quality principles outlined in [99]. The aim of this evaluation was to ensure that the data had not been lost, duplicated or altered in any unexpected ways during the ETL processes. In order to achieve this certainty, we put together a suite of queries to run some descriptive statistical tests on the data cubes. The same tests were manually conducted on the CSV files in the data lake. The test passes if the results are identical. This section begins with our earliest version of this evaluation, and finishes with a repeat of the same experiment, after several upgrades were made to the system as a result of the insights gained in the first version.

7.1.1 Experiment Setup

Definition 7.1 shows the suite of tests run on each of the data cubes. Test L1 is a measure of the principle of *data completeness* from [99] - it checks that the number of rows/instances of the dataset has not changed. This is a reasonable indication that no data has been dropped or duplicated. D1 and D2 are measures of *data consistency* run on each dimensional attribute in the data cube. D1 is a test of

whether the number of distinct terms in the source and data cube are the same, indicating that no two `source_terms` were matched to the same `standard_term`, nor that one `source_term` was matched to more than one `standard_terms`. For each file in the data lake, there should be a one-to-one mapping of `source_term` to `standard_term`. D2 checks the count of each individual dimensional value term in the dataset. If, for each dimensional value, the number of instances of that term in the data cube matches the number of instances of the equivalent term in the source data, it is an indication that the mapping of each term was done consistently. Tests M1-M3 are tests of *data accuracy* to ensure the conversion process correctly transformed the measure data.

Definition 7.1. Query Test Definitions

```
L1. SELECT COUNT(*) FROM <table>;
D1. SELECT COUNT(DISTINCT <dimension_attribute>) FROM <table>;
D2. SELECT <dimension_attribute>,COUNT(1) as count FROM <table>
GROUP BY <dimension_attribute> ORDER BY count DESC;
M1. SELECT sum(<measure>) FROM <table>;
M2. SELECT avg(<measure>) FROM <table>;
M3. SELECT std(<measure>) FROM <table>;
```

Test L1 is performed on each data cube as a whole. The results of a `SELECT COUNT(*)` should match the number of rows in the `num_valid_rows` in the Import Template for this file. D1 is performed on all dimension attributes, while D2 is run on all but the `date` dimension. The reason for this is that the `date` dimensions do not cause ambiguity issues such as can happen with the `product` or `geo` dimensions. M1, M2 and M3 are each performed on each measure variable in the data. Each of these tests will be said to pass if it gives a result identical to the source data file. In the case that the measures were converted using the ruleset during the transformation process, the values are manually converted using the same conversion function, as shown in Equation 7.1.

$$
\begin{aligned}
O &= originaldata \\
T &= transformeddata \\
\alpha &= Function(O) \\
output &= \begin{cases}
\{\mu(O) = \mu(\alpha T) & \\
\Sigma(O) = \Sigma(\alpha T) & \\
\sigma(O) = \sigma(\alpha T)\} & pass \\
\text{otherwise} & fail
\end{cases}
\end{aligned} \tag{7.1}
$$

The following data sources were used for the evaluation:

- The United States Department of Agriculture (USDA) [77] publishes Agri trade data figures which can be downloaded in bulk. Two different datasets were used from this source.
- StatCan [18] is the Canadian National Statistics agency and publishes economic, social and census data.
- Comtrade [22] is the U.N. international trade statistics database.
- Kepak Group [35] are an Irish Agri company and have provided sales data from their internal data warehouse.
- Bord Bia: the Irish Food Board [14]
- CLAL: An Italian advisory board for dairy and food products [20]

These sources represent a variety of data file sizes, types and the data mart used to represent the target dataset. Data from these sources was extracted and underwent the Importation, Analysis, Extraction, Transformation and Loading processes detailed in Chapter 4. The resulting data cubes and their ID's are found in Table 7.1 where `cube_id` is the unique identifier of the cube and `source` shows the source

Table 7.1: Data cubes for ETL evaluation

cube_id	source	cols	rows	measures	dim
C003	Bord Bia	5	24990	price	dim_date_daily dim_geo dim_product dim_unit
C007	Kepak Group	10	5000	trade_weight trade_value yield_to_spec1 offcut_value	dim_date_daily dim_product dim_org dim_unit
C009	Statcan	9	2559	trade_weight trade_value	dim_date_monthly dim_geo dim_trade_product dim_trade_flow dim_unit
C011	CLAL	5	3024	production	dim_date_monthly dim_geo dim_product dim_unit
C012	Comtrade	10	1694	trade_weight trade_value	dim_date_monthly dim_geo dim_trade_product dim_trade_flow dim_unit
C013	USDA	4	10092	stocks	dim_date_daily dim_geo dim_product
C014	USDA	10	59028	trade_weight trade_value	dim_date_monthly dim_geo dim_trade_product dim_trade_flow dim_unit

of the data where all sources are web-based except for Kepak; `cols` is the number of columns in the source data; `rows` is the number of rows or instances; `measures` is the list of measures and `dims` is the list of dimensions found in the data. A number of these cubes are from the set of ten data cubes found in Table 4.4, while others are newly imported to evaluate the ETL processes with unseen data.

7.1.2 Results

The results of our initial evaluation on the data cubes are presented in Tables 7.2 and 7.3. Note that it is often the case that a dimension may be linked to more than one attribute of a source, e.g. if a data source contains two columns whose values are geographical information, `reporter` and `partner`, both have values to be stored in the `dim_geo` dimension. This is the reason for the high number of tests run on the cubes C007, C009 and C012, compared to the number of dimensions in

Table 7.2: Pass:Fail ratios

cube_id	**pass:fail**
C003	9:2
C007	18:2
C009	13:6
C011	11:0
C012	20:2
C013	6:1
C014	17:5

the data. Conversely, the dimensional tests D1 and D2 were not run on dimensions which only has one value for each row, such as the geo dimension for C013 or C014, which was always 'US'.

Table 7.2 shows the ratio of passes to fails for each cube, where the `pass:fail` column shows the number of queries run on each data cube that yielded identical results to source data queries (i.e. passed), as compared to those that failed. As we mentioned in Chapter 4, the impact of these errors may affect both the query results and the way in which the query could be expressed. Therefore, we did not consider these results to be sufficient and made changes to the components of our system, described in the next section, before conducting another evaluation.

We selected a single cube to show the results of each individual test for, which is shown in Table 7.3. It can be seen that the cardinality test (L1) passed, meaning the dataset length did not change. The sum, average and standard deviation tests (M1, M2, M3) also passed, so the measure data was correctly transformed, but D1 and D2 both failed for the unit dimension, which means there was an error while mapping the name of the unit (rows 5 and 8 in the table).

7.1.3 Analysis

When analysing the results in Table 7.3, the `unit` dimension failed both the D1 and D2 tests. We discovered the reason was that the data source provider uses the unit 'cent' to publish its data, which caused ambiguity as more than one currency uses

Table 7.3: Detailed C003 results

attribute (data type)	**test**	**pass**
dataset length	L1	Y
date (date)	D1	Y
geo (dimension)	D1	Y
product (dimension)	D1	Y
unit (dimension)	D1	N
geo (dimension)	D2	Y
product (dimension)	D2	Y
unit (dimension)	D2	N
price (measure)	M1	Y
price (measure)	M2	Y
price (measure)	M3	Y

the *cent* as a denomination: Euro, US dollars, AUS dollars, etc. The source of the data for this cube, Bord Bia, is an Irish website so uses Euro as the unit of currency. Therefore, this required the user to clarify in the DataMap that the term refers to cent as a denomination of Euro, as opposed to that of another currency. This led to investigation into how the system handles ambiguity, allowing us to categorise the outcomes of our SchemaMatch function (4.2.2).

We found all passes for the L1 test and for the M1, M2 and M3 tests for all datasets; all the fails found were from the D1 or D2 tests. This means that any flaws in our system are found in the manner of transforming dimensional values. Among the trade datasets (C002, C003, C005), the most common issue was product codes in the `dim_trade_product` dimension not being correctly mapped. The reason is that these sources use the Harmonized Commodity Description and Coding Systems [97] but each display these codes in different ways. For example, the HS code 20120 represents the products included in 'Bovine cuts bone in, fresh or chilled', but the sources may display this code in their data in different ways: 020120, 02012000, 2012000000 etc.

This discovery of the errors found in our ETL process led to a number of additions to our approach that had not yet been added at this stage:

- We adapted our vocabulary by selecting a format to display HS codes at the

finest level of detail i.e. 02012000 and to provide a mapping process to map each way of displaying the code to this standard format. On investigating this product coding system further and including it in the CATM over a period of time, we observed that products also sometimes gets reclassified under the HS system, where their code changes. Each time this happens, the domain vocabulary needs to be extended to map the previous product code to the new one.

- We added the annotation of our vocabulary to indicate which dimension a term may be mapped to, as the *same* dimensional value can be used in more than one dimension. For example 'North America' is both a geographical place and a breed of cow.

On analysing the causes of errors found in the evaluation of our data cubes, we created a more comprehensive classification errors which could occur during the transformation process.

- **Cardinality error**: a difference in the number of rows between the source data and the data cube.
- **Attribute error**: a source attribute name not matched to a dimensional attribute from the CDM.
- **Value mismatch error**: a source dimensional or measure value mapped to an incorrect value.
- **Null value error**: a source dimensional value not matched to a dimensional value from the CDM.

However, the results also highlighted the need for a manual post-processing check after the SchemaMatch and DataMatch functions. Following these extensions to our approach and the re-classification of the ETL errors, we repeated this set of experiments with a new set of data cubes which are presented as V2 of this evaluation.

Table 7.4: Data cubes for ETL evaluation V2

cube_id	source	cols	rows	measures	dims
C003	Bord Bia	5	24990	price	dim_date_daily dim_geo dim_product dim_unit
C007	Kepak Group	10	5000	trade_weight trade_value yield_to_spec1 offcut_value	dim_date_daily dim_product dim_org dim_unit
C008	Pig333	4	14635	price	dim_date_daily dim_geo dim_product
C009	Statcan	9	2559	trade_weight trade_value	dim_date_monthly dim_geo dim_trade_product dim_trade_flow dim_unit
C011	CLAL	5	3024	production	dim_date_monthly dim_geo dim_product dim_unit
C012	Comtrade	10	1694	trade_weight trade_value	dim_date_monthly dim_geo dim_trade_product dim_trade_flow dim_unit

7.1.4 Dynamic ETL Evaluation V2

The setup for the second version of the dynamic data cube evaluation is the same as the first but with a number of extensions to the existing architecture. For this version of the experiment, we also replaced a previously used dataset for a new one to see if a new dataset caused any errors not previously found in our categorisation. One of the USDA datasets was replaced with a dataset from an international publisher of pig price data, Pig333.com [80]. Table 7.4 shows details of all datasets used in the second version of this experiment. The previous data cubes were dropped and re-extracted, transformed and loaded from the data lake.

The updated results are shown in Table 7.5. It can be seen that, following the results of the previous set of experiments and the resulting changes in our methodology, overall numbers of errors between the source file and the final data cube were reduced considerably. The addition of the notation of the vocabulary to include dimension and dimension attribute for terms meant that the ambiguity was reduced, for example, when there was a term that could conceivably be part of more than one

Table 7.5: Data cubes evaluation V2 overview

cube_id	total tests	fails	error(s)
C001	11	3	Value mismatch errors
C002	11	0	
C003	19	1	Null value error
C004	9	0	
C005	22	0	
C006	11	0	
C007	24	0	

dimension such as "North America". The three fails for cube C001 indicate that accuracy is still not at 100%.

7.2 Automated DataMap construction

Continuing our evaluation of RQ1, we examine the key component required to produce dynamic data marts. The data accuracy of the data cubes created with our ETL process depends on the accuracy of the DataMaps used to transform the data. This necessitated an investigation into the extent to which automating the process of creating the DataMaps resulted in a loss of accuracy. We examined the differences in the speed and accuracy of DataMaps created by our automated process, with those created by human participants. The key metrics for this evaluation are speed and accuracy. Speed is measured in seconds and minutes. Accuracy is measured by similarity to a "golden DataMap". To determine this, a DataMap was created manually by a domain expert, by assigning a term from the CATM to each term from the three files used in this evaluation, as well as identifying a conversion rule for each file. We make the assumption that these assignments are correct that therefore an accuracy percentage can be calculated for each of the following attributes of the DataMaps: `attr_type`, `rule`, `standard_term`, `dimension`, `dim_attr`. It is difficult to draw comparisons between the results of this process and other works. This is partly because the DataMap is a novel construct, although there are works with similarities. However, we consider a self-assessment to be the correct approach because the comparison is between the manual and automated methods to creating

Table 7.6: Source CSV files

source	num_rows	num_cols
USDA	4999	10
Statcan	2559	9
Bord Bia	24990	5

our DataMaps, as opposed to comparing the DataMaps with another approach.

7.2.1 Experiment Setup

For the experimental setup of the manual validation, we engaged three participants of varying levels of expertise in the areas of databases, data engineering and ETL, using tools built using Microsoft Excel and MySQL. The sources used were USDA, Statcan and Bord Bia. Details on the size of the files is found in Table 7.6. As the participants were not familiar with the domain (agriculture), this would add to time taken to create DataMaps. Thus, for these experiments, we selected datasets that were not overly large in size in order to avoid fatiguing our participants.

For each source data file, testers were required to extract the full set of the attribute names and the unique list of all dimensional values. Each of these terms will be called a source_term. For each source_term, the participants populated the fields of the DataMap as shown in Chapter 4, where `attr_type` denotes whether the term is a Dimension (D) or measure (F); `rule` is the unique identifier of a rule for converting measure data based on the unit of measure; `standard_term` is the specific dimensional value to which the source_term is mapped; `dimension` is the dimension from the canonical data model where the `standard_term` is found; and `dim_attr` is the specific dimensional attribute.

The participants were presented with a data file from each of the three sources, and three blank DataMaps. Participants were not permitted to write a program in any scripting language, nor to use any MySQL commands other than `SELECT`. Additionally, they were not provided with indications of which dimension each term in the data might belong to. The participants were also provided with:

- The Ruleset.
- The canonical vocabulary - the list of `source_terms` and `standard_terms`.
- An incognito browsing window to look up help pages for Excel or MySQL, as needed. This was to avoid the users inadvertently using their own browser history for assistance, as they were using their own work stations to complete this experiments. Their work stations were of the same specifications as that used for all other experiments.

7.2.2 Multi-test Results

DataMap Time to Construct Table 7.7 presents the time required for each participant to construct each DataMap (build time) and the number of mapping instances (row count) created for each. In this table, P1 represents a participant who was a beginner to ETL and data warehousing, P2 was intermediate level, and P3 was an expert. As expected, the system performs quickest and the expert user was quickest among the 3 testers. The intermediate participant was the slowest. Comparing the time of the automated method against the average of the three human participants, the automated method took 0.08% of the average human time to create a DataMap.

DataMap Accuracy. In Table 7.8, the accuracy of each DataMap for each participant when compared to the system generated DataMaps, is shown. In this table, each of the fields in the DataMap that the human users and system are to accurately assign - attr_type, rule, standard_term, dimension (dim.) and dim_attr - are shown, with the percentage of accurate guesses for each user for each data source. For example, when selecting an attr_type for the attributes of USDA, users P2 and P3 achieved 100% accuracy as did the automated system, while P1 scored 3.29% accuracy.

Although participant P2 was the slowest, as seen in Table 7.7, their accuracy was by far the highest - 92.64% overall. The beginner, P1's, accuracy was lowest at

Table 7.7: Time to build DataMaps

Tester	Source	build time	row count
P1	USDA	105 mins	304
P1	Statcan	31 mins	228
P1	Bord Bia	55 mins	1825
P1	Total	191 mins	2357
P2	USDA	215 mins	304
P2	Statcan	112 mins	230
P2	Bord Bia	25 mins	1218
P2	Total	352 mins	1752
P3	USDA	36 mins	307
P3	Statcan	35 mins	230
P3	Bord Bia	37 mins	1218
P3	Total	108 mins	1755
auto	USDA	3 secs	304
auto	Statcan	5 secs	231
auto	Bord Bia	3 secs	1218
auto	Total	11 secs	1753

58.16%. Meanwhile, the expert's accuracy was high at 72.45% but lower than the intermediate's.

7.2.3 Analysis

Although the automated system-built templates did not provide 100% accuracy, it performed favourably, giving an overall accuracy rating of 85.55% as shown in Table 7.8. This compares favourably with the accuracy ratings of the two more experienced participants. However, it is clear that there were some errors in the accuracy of both the participants and the automated system. The categorisation of these errors found in §4.2.6 was as a result of this set of experiments. For reference:

- Error Type A: A one off mistake in mapping a `source_term` to a `standard_term`.
- Error Type B: Repeated cases of the same `source_term` being mapped to a wrong `standard_term`.
- Error Type C: No mapping available for a `source_term`, resulting in a mapping

Table 7.8: DataMap Accuracy

Tester	Source	attr_type	rule	standard_term	dim.	dim_attr	Overall
P1	USDA	3.29%	0%	85.85%	100%	100%	
P1	Statcan	3.95%	0%	92.98%	98.68%	98.68%	
P1	Bord Bia	0.41%	0%	88.51%	100%	100%	
AVG		2.55%	0 %	89.11%	99.56%	99.56%	58.16%
P2	USDA	100%	50%	96.05%	99.67%	99.34%	
P2	Statcan	99.57%	50 %	98.7%	98.27%	98.27%	
P2	Bord Bia	100%	100%	99.92%	99.92%	99.92%	
AVG		100%	66.67%	98.22%	99.28%	99.18%	92.64%
P3	USDA	100%	0%	100%	45.4%	45.4%	
P3	Statcan	99.57%	0%	97.84%	99.13%	99.57%	
P3	Bord Bia	100%	0%	100%	100%	99.92%	
AVG		99.85%	0%	99.28%	81.51%	81.63%	72.45%
auto	USDA	100%	50%	99.01%	80.59%	80.59%	
auto	Statcan	99.57%	100%	99.13%	74.89%	99.57%	
auto	Bord Bia	100%	0%	99.92%	100%	100%	
AVG		99.86%	50%	99.35%	85.16%	93.39%	85.55%

to Null.

– Error Type C1: Missing rule. A specific case of C where the term to be mapped is a *unit of measure*.

The system did not make Type A errors - these were examples of human error. However, the automated system did make Types B, C and C1 errors. For Type B errors, they were often the result of an incorrect decision made between two or more possible `standard_terms` for a `source_term`. Some additional causes of errors in the manual versions included:

- Term duplication: The DataMaps should be a unique list of terms and their matches; two out of three human participants produced DataMaps containing duplicates.
- On the contrary, users may also accidentally omit terms that should be included in the DataMap. This, as well as term duplication, can be seen in Table 7.7 where the `row count` differs between users.
- Due to possible reasons such as fatigue, the participants occasionally missed a

step in the constructing of the DataMaps, e.g. forgot to assign an `Attr Type` to every term in the DataMap.

- Making an incorrect guess. When the automated system was presented with this ambiguity, it would instead map the term to null, allowing a user to easily see where intervention is needed.

It is interesting to note in Tables 7.7 and 7.8 the differences in the speed and accuracy between the human participants. Between human participants, there is a trade-off observed between speed and accuracy. It is also noteworthy that humans suffer from lower accuracy in cognitive tasks when they are fatigued. In psychology, researchers have studied this relationship in terms of the effect of fatigue on both decision-making and response time [62,96]. This effect is also shown to be consistent regardless of the perceived difficulty of the task itself [86].

For example, P3 (the expert participant) had a lower accuracy than the intermediate tester, P2 - 72.45% as opposed to 92.64% - but was considerably faster. The intermediate user, P2, while slower, had the highest accuracy. The beginner, P1, was faster than P2 but P1's DataMap contained the most errors. Meanwhile, the automation was obviously the fastest approach and also achieved significantly higher accuracy than 2 of the 3 testers. Similarly, our goal is to achieve a "sweet spot" in terms of losing as little accuracy as possible while allowing for less than 100% accuracy when the time savings are sufficient. The system's accuracy was close to that of the most accurate human participant.

When creating the DataMap for USDA, our system ran into an issue of term ambiguity that it solved incorrectly. It was presented with a term that may have been one of two domain attributes (Case 3 as mentioned in §4) and incorrectly mapped this term to the wrong dimension - `dim_product` as opposed to `dim_trade_product`. This lowered the overall accuracy for this source and was the reason why the system had lower accuracy than the human users, as both the `Dimension` and `Dim. Attr.` fields in the template were wrong. The human users were able to distinguish between these two dimensions but the system found them ambiguous. The two dimensions

appear similar but the `dim_trade_product` dimension has additional features, as mentioned before the HS coding system. Thus, the transformation process was adapted to recognise other distinctive features of international trade datasets, i.e. the presence of a `flow` attribute and two `geo` attributes instead of one, and use the correct one of the two product dimensions when they are found.

In cases where either a participant or the system got none (0%) of the attribute value mappings correct, the reason is that it is a single-value attribute and the system/user did not correctly map this single value. For example, this happens frequently with attributes `unit_weight` and `unit_value`. These attributes each have a single value for the entire dataset, e.g. 'Tonnes', 'Dollars', etc. Thus, if the user or system does not find a match for these terms, the accuracy for that attribute will be 0. However, fixing this in the DataMap would be straightforward as it is a Case 1 error from the error list at the beginning of this section.

Overall, the accuracy of the automated approach to building DataMaps compared favourably with those manually built by people with high levels of skill in the areas of data warehousing and ETL, while reducing the time spent on build time from hours to seconds.

7.3 Query Reuse and On-Demand ETL

To evaluate our approaches used to produce RQ2 - query reuse using dynamic data marts, and RQ3 - on-demand ETL, we use the ten data cubes in Table 4.4. In Table 7.9, we show some statistics on the dimensions in our Common Data Model, the CATM, and the extent to which the values can be found in the set of ten data cubes. Table 7.9 shows some insights about the data cubes and the sources which contribute to them. The `dim_geo` dimension was initially specified as a simple list of all the countries, continents and trading blocs in the world. However, on conducting an analysis of our large number of data sources, only a subset of these are of significance to the Agri domain.

There are some features found in international trade datasets not found in other datasets. For example, it can be seen that cubes C001, C006, C009 and C010 have values from the `dim_trade_flow` dimension while the others do not. Additionally, these cubes will have two attributes which are links to the `dim_geo` dimension while the rest have one. One of these link is the reporter of the data while the other is their trading partner, and the trade flow determines whether each transaction is an import, export, re-import or re-export. We also differentiate between two product dimensions, because products used in trade datasets have a HS code and a description. These are held in the `dim_trade_product` dimension. The `dim_product` dimension has simpler descriptions but also contains a large quantity of supermarket products.

7.3.1 Experiment Setup

To evaluate our On-Demand ETL architecture, our experiments use elements found in the evaluation of [8] in that our focuses are (i) full reuse of previous queries to fulfil new incoming queries, (ii) partial reuse of previous queries and partial new data. These are categorised by the extent to which the data used to fulfil the query is a reuse of a previously materialised query. At one end of the scale, there is full query reuse, where 100% of the data is gathered from existing data cubes. At the other end of the scale is full on-demand ETL, where 100% of the data comes from the data lake and must undergo the full ETL processes to fulfil the query. In between is our combined approach to On-Demand ETL with query reuse. We are assuming this will be the most commonly occurring scenario, with a set of fragments from the query being passed to the data lake.

Our methodology for evaluating these processes is to run the four queries found in §3.5 against the ten data cubes, followed by systematically removing the data cubes and re-running the queries. We apply two tests to the resultset that is returned to the user. As seen in Equation 7.2, **Fulfilled** passes if all the fragments of QF for the query are fulfilled.

Table 7.9: CATM Contained in Data Cubes

Dimension	**Values**	**No. Model values contained by cubes**
dim_geo	321	C001: 254 C002: 6 C003: 21 C004: 7 C005: 6 C006: 215 C007: 9 C008: 24 C009: 103 C010: 189
dim_product	71871	C002: 3 C003: 1 C004: 6 C005: 4 C007: 513 C008: 1
dim_trade_product	1227	C001: 138 C006: 219 C009: 54 C010: 166
dim_unit	30	C001: 5 C002: 1 C003: 1 C004: 2 C005: 2 C006: 2 C007: 5 C008: 10 C009: 2 C010: 2
dim_trade_flow	6	C001: 2 C006: 1 C009: 1 C010: 2

$$
\begin{aligned}
QF &= queryfragments \\
QFF &= fulfilledqueryfragments \\
QFU &= unfulfilledqueryfragments \\
output &= \begin{cases} \{QFF = QF & \\ QFU = \emptyset\} & pass \\ \text{otherwise} & fail \end{cases}
\end{aligned}
\tag{7.2}
$$

The criteria for the **Constraints** test are:
(i) there should be no values in the cube that are not required to fulfil the query; e.g. for the `geo` dimension in the resultset for query Q002, there should not be any data for which the value of `geo` is 'UNITED STATES' as Q002 has specified a constraint where the value for `geo` is in ("AUSTRIA","GERMANY","CHINA").
(ii) only the attributes specified by RA of the query should be shown in the cube.
If the **Constraints** test has failed, the user will still have access to the data with the extraneous attributes and/or values.

7.3.2 Results

We first provide the results of some key steps in the query matching process of a single query. We will highlight any challenges that arose during this process, what insights we gained and where we found that the system required an improvement before we could continue. Following this, we will show the overall results of some key metrics - runtime and completeness. Again, we will show what insights we gained during this process before conducting a second version of this overall experiment. We will then provide a breakdown of each task in terms of runtime. Finally, we will conclude with our insights gained after the iterations of this evaluation.

Using Q003 as a case study, we will show the output of

1. Query fragmentation
2. Selecting of candidate cubes
3. Filtering candidate cubes
4. Identification of missing fragments
5. Identification of matching lake fragments
6. Fragment integration

For reference, the specification for query Q003 is shown in Example 7.3.1.

Example 7.3.1. Q003
RA: (yearmonth,reporter,partner,flow,product_code,product_desc,trade_weight,trade_value)
RV: {reporter:(AUSTRALIA,UNITED STATES,CANADA,IRELAND,FRANCE,SPAIN),
flow:(export)}
F: null

1. Query Fragmentation

Fragmenting the query results in a set of query fragments where each value searched for has its own attribute-value pair fragment. Each `lazy select all` is shown as the attribute plus the character *. This is to distinguish it from a `greedy select all`, which would result in a an attribute-value pair for every possible value in the Common Model for that attribute.

Example 7.3.2. QF(Q003) = [(flow:export), (reporter:CANADA), (reporter:AUSTRALIA), (reporter:FRANCE), (reporter:UNITED STATES), (reporter:IRELAND), (reporter:SPAIN), (yearmonth:*), (partner:*), (product_code:*),(product_desc:*),(trade_weight:*),(trade_value;*)]

2. Candidate Cube Selection

Table 7.10 shows the results of the selection process which flagged the list cubes as candidate cubes by matching them to fragments shown.

Table 7.10: Query Fragment Matching

Cube ID	Matching fragments
C004	(yearmonth:*)
C005	(yearmonth:*)
C006	(yearmonth:*),(partner:*),(product_code:*) (product_desc:*),(flow:export), (trade_weight:*),(trade_value:*),(reporter:FRANCE), (reporter:IRELAND),(reporter:SPAIN)
C007	(yearmonth:*),(partner:*)
C009	(yearmonth:*),(partner:*),(product_code:*) (product_desc:*),(flow:export), (trade_weight:*),(trade_value:*),(reporter:CANADA)
C010	(yearmonth:*),(partner:*),(product_code:*) (product_desc:*),(flow:export), (trade_weight:*),(trade_value:*),(reporter:UNITED STATES)

Table 7.11: Filtered Candidate Cubes

Cube ID	Action
C004	Removed as subset of C006
C005	Removed as subset of C006
C006	Kept as candidate cube
C007	Removed as subset of C006
C009	Removed as subset of C010
C010	Kept as candidate cube

3. Cube Filtering

On filtering the candidate cubes, the outcome is shown in Table 7.11. Cubes C004, C005 and C007 were removed as subsets of C006, as the set of matching attributes found in each of these cubes were a subset of those in C006. However, C009 was removed as a subset of C010 for the same reason. The reason for this was that previous versions of our system identified identical sets as subsets which resulted in a failed **Fulfilled** test. We resolved this by removing only proper subsets during the filtering process.

4. Missing Fragment Identification

On creating the Multi-CM, the stacked CubeMaps of all remaining candidate cubes, only one fragment was still found to be missing from the results and was passed to the lake, shown in Example 7.3.3.

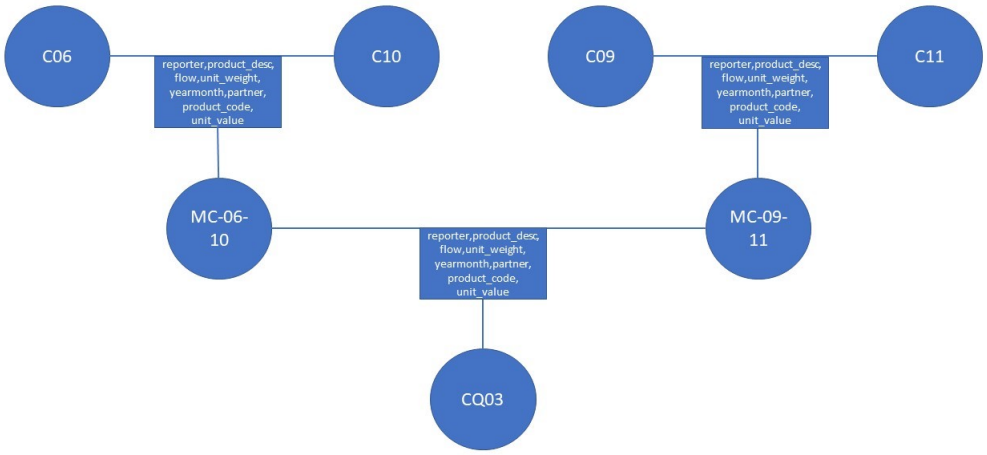

Figure 7.1: Integration Plan Q003

Example 7.3.3. QFU(Q003) = [(reporter:AUSTRALIA)]

5. Lake Fragment Matching

The DataMaps were searched to find the files that contained both the attribute and the value of the missing fragment. Three files were found that contained the attribute 'reporter', which had a value 'AUSTRALIA' for this attribute. These were sorted by size and the largest one processed into a data cube with cube_id C011.

6. Fragment Integration

The integration plan for joining these cubes is seen in Figure 7.1. Between the cubes with identifiers C006 and C010 in Table 7.11, the link is found to combine these nodes, and the same for C009 and C011. Each of these combined nodes then forms a new node, and the linking attributes between these new nodes are found again. The output of this process is a new data cube, which is renamed. This undergoes the post-processing stage of applying the constraints required by the query before being returned to the user.

In Table 7.12, we show the results of running each of our queries with query reuse only, i.e. the system combines the partial matches in the existing cubes to fulfil the query insofar as the data in the cubes can, without searching the data lake. In this table, `Query` is the unique identifier of the query, `Cubes` is the cubes that contribute to fulfilling the query, `Link` is the integration link, i.e. the set of attributes used

Table 7.12: Query Reuse

Query	Cubes	Link	Rows	Cols	Runtime	Constr.s	Fulfil
Q001	C007 C008	date	32457	5	15.7s	n	y
Q002	C003 C008	date,geo, product,unit	3682	4	15.3s	y	y
Q003	C006 C009 C010	product_desc, reporter,flow, unit_weight, yearmonth, partner, unit_value, product_code	84663	12	29.7s	y	n
Q004	C002 C004 C005 C006 C009 C010	product_desc, reporter,flow, unit_weight, yearmonth, partner, unit_value, product_code	692828	15	2min44s	y	n

to join all the cubes, `Rows` and `Cols` are the number of rows and columns of the resulting data cube, `Runtime` is the time in seconds to materialise the resulting data cube to fulfil the query, `Constr.s` is the result of the **Constraints** test, and `Fulfil` is the result of the **Fulfilled** test.

For queries Q001 and Q002, the cube initialised would be conforming to the `fact_price_weekly` data mart as they are both joined on date and contain the price measure. Queries Q003 and Q004 are fulfilled using primarily data from sources that publish international trade figures. Therefore, the sources are joined using attributes that are elements of the `fact_trade_monthly` data mart in the CATM. The data cube initialised loading processes would therefore conform to that model. However, it would need to be joined with additional measures.

Table 7.13 shows the setup of our On-Demand ETL evaluation. For Q003 and Q004, we begin with ten data cubes and a populated data lake. From this beginning setup, we remove the data cubes two at a time and rerun all four queries. From the beginning ten cubes, we remove the cubes in order of the unique identifier of the

Table 7.13: On-Demand ETL Experiment setup

Exp_id	**Cubes in store**	**Cubes dropped**
Exp1	10	0
Exp2	8	C001 C002
Exp3	6	C003 C004
Exp4	4	C005 C006
Exp5	2	C007 C008
Exp6	1	C009
Exp7	0	C010

cubes. When dropping a cube, we remove it entirely from the system: we drop the cube itself from the cube store, remove the associated CubeMap and valuesets, and remove the cube from the Cube Matrix.

In Table 7.14, `Query` is the unique identifier of the query, `Exp_id` is the reference to the system setup in Table 7.13, `% reuse` is the percentage of data in the final data cube that was reused from previous queries, calculated in Equation 7.3 where $Cube_R$ is the number of rows of the data cube that were fragments reused from previous data cubes and $Cube_T$ is the size of the full data cube, `files` is the number of files that were processed from the data lake, `Transforms` is the number of transformations - dimension mappings or measure conversions - for taken place during the ETL process of the lake files found for fulfilling this query and `Runtime` is the time taken to materialise the resulting data cube to fulfil the query. The criteria for the `Constr.s` and `Fulfil` columns remain the same. When the `Cubes in store` is equal to ten, we omit the results for Q001 and Q002 as these results can be found in Table 7.12 and no additional data is required from the data lake to fulfil these queries.

$$PercentReuse = \frac{|Cube_R|}{|Cube_T|} \times 100 \tag{7.3}$$

Similar to our experiments on the ETL process, we ran a first version of this set of

Table 7.14: On-Demand ETL Partial reuse V1

Query	Exp_id	% reuse	files	Transforms	Runtime	Constr.s	Fulfil
Q003	Exp1	81	1	231310	1min2s	y	y
Q004	Exp1	99.3	1	231310	3mins 40s	y	y
Q001	Exp2	100	0	0	20s	n	y
Q002	Exp2	100	0	0	21.2s	y	y
Q003	Exp2	81	1	231310	59s	y	y
Q004	Exp2	99.3	2	360910	9mins15s	n	y
Q001	Exp3	100	0	0	20s	n	y
Q002	Exp3	67	1	124950	36s	y	y
Q003	Exp3	81	1	231310	1min5s	y	y
Q004	Exp3	99.3	2	360910	9mins15s	n	y
Q001	Exp4	100	0	0	20s	n	y
Q002	Exp4	67	1	124950	36s	y	y
Q003	Exp4	62.9	2	1677270	2min58s	y	y
Q004	Exp4	21	4	2054480	9mins20s	y	n
Q001	Exp5	0	2	174950	33s	n	y
Q002	Exp5	0	3	260630	50.8s	y	y
Q003	Exp5	62.9	2	1677270	2min58s	y	y
Q004	Exp5	21	4	2054480	9mins20s	y	n
Q001	Exp6	0	2	174950	33s	n	y
Q002	Exp6	0	3	260630	50.8s	y	y
Q003	Exp6	60.9	4	2069010	2mins 53s	y	y
Q004	Exp6	52.9	8	2222866	3mins 21s	y	n
Q001	Exp7	0	2	174950	33s	n	y
Q002	Exp7	0	3	260630	50.8s	y	y
Q003	Exp7	0	5	2914930	3mins 28s	y	n
Q004	Exp7	0	6	3049770	3mins 23s	y	n

experiments and analysed the results. We will identify and categorise the problems that emerged, describe the improvements made to the system based on these results, then show the results of our second version.

The following observations were made when running the evaluations shown in Table 7.14: **Exp1**

In Exp1, both query Q003 and Q004 found the same file, a file from Comtrade, which supplied the Australian data that the results were missing in the query reuse experiments.

Exp2

For Q003, again the Comtrade data was used to fulfil the query along with the remaining cubes, so the results are approximately the same as for Exp1, apart from small differences in processing time due to the smaller number of CubeMaps and the smaller Cube Matrix. For Q004, the cube that supplied the animal slaughters data (C003) no longer existed. The process to query the lake found a different source for this data, a Eurostat source. This is because the functions to query the lake favour larger lake files over smaller ones. However, the resulting dataset could not be joined on date values, so the possible resultsets are the cube without the constraints applied, or the results of constraints applied but without the slaughters data.

Exp3

For Q002, the remaining cube contained data for Germany and China, but the cube that supplied the Austrian data has now been removed. The query lake process found the same file in the lake and produced the same cube as before the cubes had been removed, with a small addition to the processing time. For Q003, the results are again the same as for Exp1 as the Australian data from Comtrade was all that was required to fulfil the query. The results for Q004 were identical to Q003

Exp4

For queries Q001 and Q002, the results were identical to Exp3. However, queries Q003 and Q004 now only have two of their contributing cubes. This run of experiments was the first time we had a situation where a single lake file was used to fulfil more than one query fragment, as the Eurostat data reports trade figures for multiple countries while USDA and Statcan do not. This necessitated a fix in the program to ensure processing time was not used loading a single lake file multiple times.

It was found that, for query Q004, again the prioritising of larger files caused a considerable increase in processing time while not ensuring the files found were ideal to merge with the existing cubes.

Exp5

This set of experiments is the first instance of a No Match - a query being run for which no cubes are available, as cubes C007 and C008 are now removed. Therefore, queries Q001 and Q002 now need to be fulfilled entirely from finding the correct lake files. Q001 was fulfilled using the same files as were used to create the cubes that originally fulfilled it, again with a small increase in processing time. Q002 was fulfilled with the two cubes that previous fulfilled it plus an additional cube that also fulfilled the criteria. This additional file is of grain prices while the other two were pig prices, however, the query does not specify a product. The results for Q003 and Q004 were identical to Exp5.

Exp6

The results for queries Q001 and Q002 were identical to those for Exp5. For Q003, the results were similar to Exp5 as the cube removed between the two sets of experiments made up only a small portion of the data for this query. The system found the correct data files as well as an additional one which also fulfilled the criteria.

For Q004, several of the files processed from the lake were dropped during the process of creating integration edges, leading to an incomplete data cube, which accounts for the sudden drop in runtime as well as the increase in the amount of data in the final cube which was reused. The resulting cube was small and only contained a small amount of the data required for the query.

Exp7

The results for queries Q001 and Q002 were identical to those for Exp5. For Q003, the system again found additional files but did not result in a complete integration. The resulting cube for Q004 did not contain all the measures required by the query. The system found files that had the correct measures, but these were not filtered by dimensional values, meaning that although they were the correct measure, they may not share countries, dates or products on which to join, and thus, are eliminated during the integration stage.

Categorising the errors found in this set of experiments, i.e. whenever either constraints were not applied to the resulting data cube or the cube did not fulfil the

query, the following issues were found listed:

- In query Q004 in Exp2 and again in Exp4, the prioritising of larger files caused issues with speed while not improving the result. It was decided that, in the second version of the experiments, the process of querying the lake would be altered so that it prioritised number of unique individual terms over file size.
- In Exps 4 and 5, queries Q003 and Q004 behaved quite differently in that Q003 needed to go to the lake for missing data on the value-level only, while Q004 required more than one additional measure that could not be found in the cubes available.
- In some cases, the file in its source original structure was not a structure that the system was expecting, leading to the transformation process transforming the wrong columns of the file. For example, the transformation process expects that the file will have a column that shows the unit of measure, and another which is the measure values. However, in some cases the file used the name of the unit as the column header for the measure values, or contained the unit of measure in the product description. These were fixed by the addition of static variables to the data during the transformation process. However, before this was discovered, it led to incomplete integration plans such as in Exp6, query Q004.

7.3.3 On-Demand ETL V2

The results of the second version of our on-demand ETL evaluation are found in Table 7.15. For this final version of the evaluation, we present a more detailed exploration of the **Fulfil** and **Constraints** tests, as opposed to a pass/fail. As seen in Equation 7.4, **Fulfil** is a measure of the degree to which all query fragments QF are fulfilled where `QF` is the set of query fragments and `QFU'` are the set of fragments that remain unfulfilled at the end of the full on-demand ETL process.

$$Fulfil = 1 - \frac{|QFU'|}{|QF|} \tag{7.4}$$

Constraints is a measure of the percentage of a materialised cube that was not requested in a query. This can be seen in 7.5 where `EC` is the set of additional columns obtained during the query and `ER` the set of additional rows as a result of the inclusion of extraneous data, `CA` is the total number of attributes in a materialised cube and `CR` is the total number of rows for said cube.This figure is derived from manual inspection of a cube after materialisation.

$$Constraints = 1 - \frac{|EC| * ER}{|CA| * CR} \tag{7.5}$$

We found an improvement in the runtime of the queries over the previous version of experiments. However, it was still sometimes necessary to provide the user with the table before constraints were applied. We found that, with experiment runs such as query Q004 for Exp 4-7, the system did not have enough information to select a file from the lake that was sure to successfully integrate with the existing data, or with the other lake files found. The system performs well when it required a specific value to integrate with cube data, but less so when there was a measure required from the files. This highlights the necessity to make queries highly specific when using on-demand ETL.

The total runtime can be broken down into eight basic tasks, the details of which are shown in Table 7.16. The runtime for launching and fragmenting the query was static at approximately 0.01 of a second for each query for each experiment. In Figure 7.2, we show details of the runtimes shown in Table 7.15. In this figure, the runtimes of the tasks associated with on-demand ETL only are shown, enabling us to draw comparisons with other approaches in §7.3.4.

Table 7.15: On-Demand ETL Partial reuse V2

Query	**Exp_id**	**% reuse**	**files**	**Transforms**	**Runtime**	**Constr.s**	**Fulfil**
Q001	Exp1	100%	0	0	8.5s	83.3%	100%
Q002	Exp1	100%	0	0	8.8s	100%	100%
Q003	Exp1	81%	1	231310	47.9s	100%	100%
Q004	Exp1	99.3%	1	231310	1min29s	53.5%	100%
Q001	Exp2	100%	0	0	6.7s	83.3%	100%
Q002	Exp2	100%	0	0	6.96s	100%	100%
Q003	Exp2	81%	1	231310	46.4s	100%	100%
Q004	Exp2	81.3%	2	360910	8mins27s	74.6%	93.75%
Q001	Exp3	100%	0	0	8.5s	76.9%	100%
Q002	Exp3	67%	1	124950	18.3s	100%	100%
Q003	Exp3	81%	1	231310	46s	100%	100%
Q004	Exp3	69%	2	360910	1min20s	86.6%	100%
Q001	Exp4	100%	0	0	8.7s	76.9%	100%
Q002	Exp4	67%	1	124950	19.2s	100%	100%
Q003	Exp4	76%	2	366150	1min 3s	100%	100%
Q004	Exp4	27%	4	1670092	3min 6s	82.4%	75%
Q001	Exp5	0%	2	74990	23.8s	83.3%	75%
Q002	Exp5	0%	3	260630	33s	100%	100%
Q003	Exp5	76%	2	366150	1min 5s	100%	100%
Q004	Exp5	37%	6	592690	2mins 50s	82.4%	75%
Q001	Exp6	0%	2	74990	23.8s	83.3%	75%
Q002	Exp6	0%	3	260630	33s	100%	100%
Q003	Exp6	60%	3	391740	2mins	100%	100%
Q004	Exp6	12%	7	2250650	4mins 40s	82.6%	75%
Q001	Exp7	0 %	2	74990	24.7s	83.3%	75%
Q002	Exp7	0%	3	260630	43.6s	100%	100%
Q003	Exp7	0%	4	2981350	2mins 3s	100%	100%
Q004	Exp7	0%	5	3004870	2mins 30s	82.6%	75%

Table 7.16: On-Demand ETL Tasks (% of runtime)

Task	Task description	Q001	Q002	Q003	Q004
Query Match 1	Inspect Cube Matrix, find candidate cubes	4.4%	5.96%	1.77%	0.8%
Query Match 2	Filter partial matches, remove subsets, create QueryMap, combine partially matching CubeMaps, find QFU	2.8%	1.07%	0.5%	0.4%
Search lake	Filter DataMaps, find matching lake files	9.6%	7.21%	9.93%	4.5%
Extract from lake	Read file from data lake, translate query	1.4%	2.14%	1.15%	0.49%
Transform	SchemaMatch, DataMatch, Measure conversion, apply static variables	14.3%	16.9%	7.8%	7.07%
Load	Initialise and populate data cubes	35.2%	34.5%	29.17%	16.9%
Join	Create integration plan, perform joins	25.38%	22.8%	29.17%	43.2%
Materialise	Apply constraints, functions, return to user	6.6%	9.4%	20.55%	26.5%

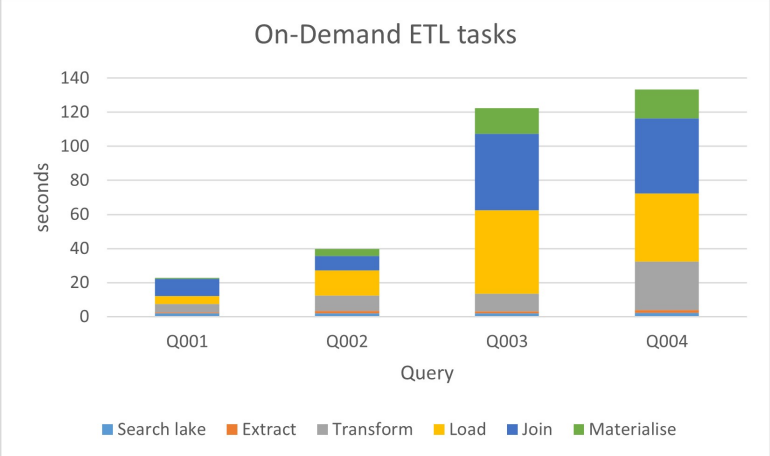

Figure 7.2: Runtime by task - avg

7.3.4 On-Demand ETL Analysis

The overall success of the on-demand query fulfilment hinges on the robustness of the underlying ETL architecture. Because of the steps taken at the point of data acquisition - the metadata captured in the Import Templates and DataMaps - the curation of data from the data lake when it was required was a very quick process. The system could treat lake fragments and cube fragments in the same way because of the metadata constructs supplying the information required to all processing steps. We have shown the improvements we made in several stages between our experiment iterations in Tables 7.12, 7.14 and 7.15.

The time taken to resolve a query generally increased with the complexity of the steps involved in resolving that query - the number of cubes to be processed, the number of fragments to be matched by the lake and the time taken to join all partial matches. The runtime for query Q004 is highly variable compared to the other 3 queries. This effect was seen in Exps where the majority or all of the data was to be fetched from the lake as opposed to reused. For example, in Table 7.15, in Exp3 for query Q004, 69% of the data in the final resultset is reused and the time taken is 1 minute 20 seconds. In the next set, Exp4, 27% of the data in the resultset is reused and the time taken increased to just over 3 mins. The reason for this variability is frequently due to both the number and complexity of the transforms that must take place - see for instance the increase in runtime and the increase in transforms for query Q004 between Exp5 and Exp6. However, it should be noted that the relationship between number of transforms and time taken to transform is also not perfectly linear. For example, transforming large quantities of measure data takes longer than transforming files with only a single measure and a large number of dimensions. Additionally, the complexity of the task of joining the candidate cubes has a number of variables that may slow down the runtime besides simply how many cubes to be joined, such as the number of attributes on which to join and the number of individual values for those attributes. The results of query Q004 in Exp2 are an example of a file being found which could not be integrated with the remaining cubes, i.e. the integration plan failed because it was incomplete. This is

the reason for the high runtime as well as the $< 100\%$ `Fulfil` score.

However, this time saving must be weighed against the savings in not having pre-loaded that large portion of data before it was needed. In terms of the comparison between our results and those of the most similar approach [8], although their runtimes were faster than ours, it must be considered that ours has tasks to complete that are not seen in the previous work, to account for the greater complexity of our search and integration processes when using more uncontrolled, disparate sources.

Completeness tests

Examining the results of the latest set of experiments, the results shown in Table 7.15, the `Constr.s` test is less than 100% for half of the results, meaning that the final resultset had extraneous data that was not needed for the query. A `Constr.s` test of less than 100% meant that, after the matching fragments had all been combined and the constraints were applied, the resulting data cube was empty. This happened in cases where the cubes selected to fulfil the query shared an attribute to link on but the values did not match. For example, for query Q001, both of the sources that contributed to this query shared the attribute 'date', but one of the data sources published their figures at the start of each week while the other source published its figures at the end of the week. In this case, we provide the user with access to the result of the integration without applying the constraints.

The results of the `Fulfil` test were less than 100%, 28.5% of the time, as seen in Table 7.15. This test was a measure of how many of the fragments in QF were fulfilled at the end of the querying process. In the cases where the `Fulfil` field in Table 7.15 was less than 100%, the reason was that one of the cubes had not successfully been integrated and thus, was excluded in the integration plan, thus the fragments required by the query that were supplied by that cube, were not fulfilled. We found that this happened when the system was searching the lake for a measure not found in the cubes, with a `lazy select all`. In Exp1, when there are ten cubes in the cube store, all the data needed to complete queries Q001 and Q002 is available in the cubes and thus, the `Fulfil` score is 100%. For queries Q003 and Q004, all the measures and dimensional attributes required to fulfil the query are

also in the cube store, and only new dimensional values are required from the lake, which the process successfully found and thus, the `Fulfil` score is 100%. For the $< 100\%$ `Fulfil` scores, they are similarly a result of the number of fragments that were fulfilled, out of the total number of fragments for each query - four for query Q001 and 16 for Q004. Both of these queries require the contributing cubes to be extended by additional measures as opposed to new values.

We can see that the process of querying the data lake is more effective when the query is more specific. The queries that required additional measures to be fetched from the lake, as opposed to just dimensional values, performed worst.

It can be seen in Tables 7.12 to 7.15 that, for query Q001, the `Constr.`, i.e. constraint results are always negative. The two cubes identified as partial matches for Q001 were successfully integrated. However, when the constraints were applied in the final materialisation of the cube, the result was an empty cube. This turned out to be due to the fact that, although the two data sources reported weekly and data is available from the same time periods, the specific days on which each reported data were different. This may occur when one source reports their data at the end of each week and another at the start of each week, for example. Thus, the cubes share a 'date' column and therefore an outer join will not fail, but the cubes do not share specific values on which they can be joined. It is for this reason that the results of the outer join are returned to the user if this is the case.

Time taken per task

It can be seen from Figure 7.2 that the formation and application of the integration plan for the data cubes is the most resource-heavy task, along with the applying of constraints. Unsurprisingly, the fewer data cubes in the cube store at the time of the experiment, the more resources were used to transform and load data. the time taken to select and filter the candidate cubes was both very small and almost static, regardless of the number of cubes being searched. We consider this to be a good indication that the CubeMaps and Cube Matrix are a very efficient way to search existing data cubes for matches to queries. During the fragment matching tasks, the actual data in the cubes is never iterated and the time taken to do this

was rarely more than one second.

Although results may take two or three minutes when data is being fetched from the data lake, compared with faster times in comparable approaches such as [8], our work also has a higher level of risk in that we have more data sources and a higher level of heterogeneity between sources. Therefore, our lake querying method is considerably more complex in terms of selecting the optimal file from a large number of options. Overall, we feel that our methodology produces considerable time savings over traditional ETL while retaining a high level of query fulfilment. Unsurprisingly, the joining and post-processing of data cubes was faster than the full ETL processing of lake data, and the experiments that included a number of the pre-computed data cubes were faster than those that relied entirely or mostly on lake data, i.e. Exp 6 and Exp 7. However, pre-computing these cubes also has a cost in terms of time and, as expected, a subset of the cubes were not used. This suggests that there is a "sweet spot" in terms of the reuse of cubes that are used frequently and therefore worth pre-loading, and supplementing that data with lake data only as needed.

7.4 Summary

This evaluation had three main goals: to evaluate the ETL architecture which underpins our solution by ensuring that it does not alter (transform, duplicate or remove) data in erroneous ways; to specifically examine the accuracy and speed of automating the DataMaps; and to provide a validation of our approach to query reuse and on-demand ETL using the case studies presented in Chapter 3. Where possible, we have aimed to provide more than one version of our experiments in order to demonstrate the research process and document the changes that were made when there were insights to be gleaned from examining these different versions.

We found a high amount of difference in the reliability of the ETL process between versions one and two, resulting in a high degree of accuracy in the final data cubes.

The automation of the DataMaps produced highly favourable results, with the automated system producing these constructs considerably faster than a human and with a very low loss of accuracy compared to an expert user.

For our final set of evaluation experiments, we examined the speed of our On-Demand ETL process. It was seen that the bulk of the runtime was spent on loading and integrating the data. We also found that the runtime generally increased as more data files were added from the lake. However, this must be weighed against the time savings in avoiding the upfront processing of data in traditional ETL, as well as the benefits of query reuse over query re-computation.

Chapter 8

Conclusions

As we arrive at a conclusion to this dissertation, we now provide a summary of our methodology, together with a reminder of the evaluation strategy and results in Section 8.1. We follow this with a discussion on a number of possibilities for further developments in Section 8.2.

8.1 Thesis Summary

We opened this dissertation with a discussion on data warehousing, warehouse schemas and data cubes. We introduced the ETL process which brings data from multiple sources into a format where they can be viewed globally. However, in an environment where data sources are frequently changing, disappearing and new sources appearing, the traditional approach to ETL and data warehousing is too rigid and resource-heavy to be suitable without some extensions. In Chapter 1, we motivated our research by identifying our problem area and presented our hypothesis. Our research goal was to produce an On-Demand ETL architecture that could support dynamic data cubes and query reuse. When conducting our study on the state of the art, our focus was on existing approaches to extending traditional ETL, such as ontology-based ETL and right-time data warehousing; creating and maintaining dynamic data cubes, i.e. data cubes formed from frequently chang-

ing sources; and approaches to query reuse and methods of processing queries so that reuse can be conducted efficiently. Finally, we examined existing On-Demand ETL methodologies and were influenced by some of the key studies in this area. However, we identified a gap in the existing literature and considered the types of extensions necessary to accommodate both our large quantity of sources and our diversity between sources.

In Chapter 3, we presented our methodology and some of the key components. Our Common Data Model was presented - the dimensions and facts used in our data cubes - along with our vocabulary and ruleset. In this chapter, we also presented some of our main data sources along with some case studies which formed a basis for our evaluation.

In Chapter 4, we presented our extended ETL architecture for the formation of dynamic data cubes. We introduced two of our main constructs that facilitate the seamless importing of unseen data sources - our Import Templates which provide a metadata layer and API for our data lake, and our DataMaps which provide a set of mappings for our transformation stage. We found our first real setback in the creating our DataMaps - for data sources of any considerable size, the manual crafting of the DataMaps did not scale. A solution was found in automating this process, with only a small input of a post-hoc check afterwards, and found significant time savings while retaining a high level of accuracy. We showed how these constructs are utilised in our main processes in acquiring, preparing and loading data, and finished with some starts on a set of data cubes that would be used in the following chapters.

These cubes were used in the research presented in Chapter 5 where we showed our approach to query reuse, and in Chapter 6 where we introduced a methodology for answering queries that require data contained outside of the cube store. The significant constructs required in this methodology were the CubeMaps, QueryMaps and the Cube Matrix. The Cube Matrix provides a metadata layer to the cube store environment and a map of those elements of the Common Model that are contained within existing cubes. The CubeMaps and QueryMaps represent the information

contained in a cube and the information required by a query, respectively. These are identically structured to allow an easy side-by-side comparison. We showed our verification of the containment of a query inside a data cube, identifying three possibilities: the full match, the partial match and no match, and the various sub-types. In Chapter 6, we presented our fully realised On-Demand ETL. This showed how data from the data lake was selected, processed and combined to present a resultset to the user.

In Chapter 7, we began by evaluating if our ETL architecture was sound, focusing especially on the transformation process. We categorised the errors that may occur in the transferring and changing of data between one structure to another, and found that these errors were rare cases and were resolved by extensions to the canonical vocabulary. We specifically wanted to ensure we were not introducing errors in the automating of our DataMaps, and showed high levels of accuracy along with significant time savings.

We carried out our query reuse and on-demand ETL methodology using four case studies. We began with a set of ten data cubes, assumed to have been produced using previous queries, and examined the results of reusing these to fulfil new queries. We then removed these in order to examine the results of partial query reuse with partial on-demand ETL, finally resulting in full OD-ETL. We found that the queries which required higher numbers of files to be processed had a longer runtime, but was still only a matter of a few minutes, while saving large amounts of processing time before the data was required. We also found that, the majority of the time, the fitness-for-use tests on the resultsets passed.

8.2 Limitations

The scope of this research was carefully defined, and certain assumptions were made in terms of the timeliness and the schema of the data. It was assumed that the data lake was as up to date as possible in terms of the timeliness of the data source

provider. We also assume, as mentioned in Chapter 3 that data sources to be integrated into our cube environment will contain a basic level of compatibility with the Common Data Model used in implementation. In this case, the Common Agri Trade Model contains a certain number of dimensions and measures. The mapping of a source to this model requires a basic amount of semantic overlap with these dimensions and measures. We are conscious that the process of ontology-building is a difficult task in itself but, for the purpose of this work, we make the assumption that we are using a Model that is comprehensive and that its creation is outside the scope of this work.

Although we used a large number of data sources, the individual datasets from each source represented relatively small data updates from each source, as opposed to a large-scale data extraction. This was done based on the assumption that on-demand ETL is best suited for cases where the user requires only enough data to answer a specific query or gain a key insight. However, we are conscious that there are remaining questions about the scalability of this approach in a Big Data requirement.

A limitation was found during the process of creating the DataMaps in that, for the data sources we used, there must be a one-to-one relationship between a data source and a DataMap. After we had the DataMaps generated as an assisted automated process, we next investigated the possibility of reusing the DataMaps. Our aim was to produce further time savings by selecting a small number of data sources for which we created DataMaps, and attempted to use these in the transformation process of previously unseen data sources. We experimented with similarity thresholds to see if the DataMap would sufficiently transform the data if the DataMap contained, for instance, 90% of the correct terms. However, the similarity between sources was too low to be successful. We found that the DataMaps were too dissimilar when trying to use one created for one source to transform data from another source. Often, the only terms the data sources had in common were dates, and even these may be in different formats. The sources were usually reported from different countries so did not share many values from the geographical dimension, and we had observed

previously that the descriptions of products were very dissimilar. The effort involved in adapting a DataMap for use for a different data source meant that much of the time saved by automating the process of creating them was lost. It was decided that generating one DataMap for each data source is very low in terms of overheads.

8.3 Future Work

Future work may involve investigating some possible strategies for further attempts to fulfil the query §8.3.1-8.3.2. Additionally, we consider how further optimisation could have been found within our methodology §8.3.3.

8.3.1 Probabilistic Approach

In our task of automating the generation of the DataMaps, we briefly discussed the concept of a "trade-off" between speed and accuracy, where the user may be satisfied with a small hit to the accuracy of the dataset, if it meant considerable savings in speed. This same concept is what underpins the use of non-deterministic databases and probabilistic data imputation. In the case where only a small amount of data is missing from the resultset returned by our system, it is possible that probabilistic forms of data imputation could be used to fill these gaps.

8.3.2 Natural Language Processing

There are a number of tools in our architecture that are an extensible set of some useful elements, such as a set of term mappings or measure conversion rules. At present, the addition of new mappings or rules is a manual process. Although this is a small input and infrequently done, and becomes less frequent as more sources are imported, the automation of this task would further optimise our system. In the case where a term mapping cannot be completed using the canonical vocabulary, the expansion of the vocabulary is a manual input. The use of Natural Language

Processing techniques to suggest a number of potential matches from the existing vocabulary is an avenue for further investigation. At present, this was not used because of the relatively short length of strings used in the data sources used in our case studies, compared to the document lengths usually used as a corpus in NLP research.

8.3.3 Dropping Policy

In this research, data cubes are created on the fly and may be reused as query results multiple times, often after a certain adaptation. An extension to this would be the intelligent dropping of cubes that are used less frequently. At present, the dropping of data cubes is dependent on the user performing this action. A system to regularly analyse the frequency with which cubes are reused to address queries would represent an optimisation, such as the Least Recently Used policy in [45].

Bibliography

[1] Mysqldb for python. `https://mysqlclient.readthedocs.io/user_guide.html`. Accessed: 2020-03-21.

[2] Selenium for python. `https://selenium-python.readthedocs.io/`. Accessed: 2020-03-21.

[3] Alberto Abelló, Jérôme Darmont, Lorena Etcheverry, Matteo Golfarelli, Jose-Norberto Mazón, Felix Naumann, Torben Bach Pedersen, Stefano Rizzi, Juan Trujillo, Panos Vassiliadis, and Gottfried Vossen. Fusion cubes: Towards self-service business intelligence. *IJDWM*, 9(2):66–88, 2013.

[4] I. A. Alvi. Etl vs. elt: Transform first or transform later? `https://datawarehouseinfo.com/etl-vs-elt-transform-first-or-transform-later`, 2018.

[5] Ali A Alwan, Azlin Nordin, Mogahed Alzeber, and Abedallah Zaid Abualkishik. A survey of schema matching research using database schemas and instances. *International Journal of Advanced Computer Science and Applications*, 8(10), 2017.

[6] Petr Aubrecht and Zdenek Kouba. Metadata driven data transformation. In *World Multiconference on Systemics, Cybernetics and Informatics, ISAS-SCIs 2001, July 22-25, 2001, Orlando, Florida, USA, Proceedings, Volume I: Information Systems Development*, pages 332–336, 2001.

[7] Fouad Bahrpeyma, Mark Roantree, and Andrew McCarren. Multi-resolution forecast aggregation for time series in agri datasets. In John McAuley and Susan McKeever, editors, *Proceedings of the 25th Irish Conference on Artificial Intelligence and Cognitive Science, Dublin, Ireland, December 7 - 8, 2017*, volume 2086 of *CEUR Workshop Proceedings*, pages 193–205. CEUR-WS.org, 2017.

[8] Lorenzo Baldacci, Matteo Golfarelli, Simone Graziani, and Stefano Rizzi. QETL: an approach to on-demand ETL from non-owned data sources. *Data Knowl. Eng.*, 112:17–37, 2017.

[9] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM computing surveys (CSUR)*, 18(4):323–364, 1986.

[10] Natércia A Batista, Michele A Brandão, Michele B Pinheiro, Daniel H Dalip, and Mirella M Moro. Dealing with data from multiple web sources. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, pages 3–6, 2018.

[11] BeautifulSoup. `https://pypi.org/project/beautifulsoup4/`, 2009.

[12] Zohra Bellahsene. View adaptation in the fragment-based approach. *IEEE Trans. Knowl. Data Eng.*, 16(11):1441–1455, 2004.

[13] Alain Berro, Imen Megdiche, and Olivier Teste. Graph-based ETL processes for warehousing statistical open data. In *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 1, Barcelona, Spain, 27-30 April, 2015*, pages 271–278, 2015.

[14] Bord Bia Irish Food Board. `https://www.bordbia.ie/Pages/Default.aspx`, 2019.

[15] Robert M. Bruckner, Beate List, and Josef Schiefer. Striving towards near real-time data integration for data warehouses. In *Data Warehousing and Knowledge Discovery, 4th International Conference, DaWaK 2002,*

Aix-en-Provence, France, September 4-6, 2002, Proceedings, pages 317–326, 2002.

[16] Peter Buneman. Semistructured data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA*, pages 117–121, 1997.

[17] Luca Cabibbo and Riccardo Torlone. On the integration of autonomous data marts. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece*, pages 223–231, 2004.

[18] Statistics Canada. `https://www.statcan.gc.ca/eng/start`, 2019.

[19] Rashmi Chhabra and Payal Pahwa. Data mart designing and integration approaches. 2016.

[20] CLAL. `https://www.clal.it/en/`, 2019.

[21] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[22] UN Comtrade. `https://comtrade.un.org/`, 2019.

[23] DairyAustralia. `https://www.dairyaustralia.com.au/industry/production-and-sales/latest-production-and-sales-statistics`, 2019.

[24] Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Patrick Valduriez, et al. Applying generic model management to data mapping. In *BDA*. Citeseer, 2005.

[25] Prasad Deshpande, Karthikeyan Ramasamy, Amit Shukla, and Jeffrey F. Naughton. Caching multidimensional queries using chunks. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 259–270, 1998.

[26] Eurostat. Eurostat: Your key to european statistics. `http://ec.europa.eu/eurostat/about/overview`. Accessed on 2020-03-20.

[27] Organisation for Economic Co-operation and Development. `https://data.oecd.org/`, 2019.

[28] Tobias Freudenreich, Pedro Furtado, Christian Koncilia, Maik Thiele, Florian Waas, and Robert Wrembel. An on-demand ELT architecture for real-time BI. In *Enabling Real-Time Business Intelligence - 6th International Workshop, BIRTE 2012, Held at the 38th International Conference on Very Large Databases, VLDB 2012, Istanbul, Turkey, August 27, 2012, Revised Selected Papers*, pages 50–59, 2012.

[29] International Monetary Fund. `https://www.imf.org/external/np/fin/data/rms_five.aspx`. Accessed: 2020-03-21.

[30] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[31] Michael Gertz, Tamer Ozsu, Gunter Saake, and Kai-Uwe Sattler. Data quality on the web. url="https://www.dagstuhl.de/Reports/03/03362.pdf", 2003.

[32] Matteo Golfarelly and Stefano Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill Education, 2009.

[33] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[34] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM,*

PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers, pages 278–296, 2006.

[35] Kepak Group. `https://www.kepak.com/`, 2019.

[36] Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[37] Ashish Gupta, Inderpal Singh Mumick, and Kenneth A. Ross. Adapting materialized views after redefinitions. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*, pages 211–222, 1995.

[38] Rashmi Gupta and Cathal Gurrin. Considering manual annotations in dynamic segmentation of multimodal lifelog data. In *IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019, Kyoto, Japan, March 11-15, 2019*, pages 34–39, 2019.

[39] David K. Hsiao. Federated databases and systems: Part I - A tutorial on their data sharing. *VLDB J.*, 1(1):127–179, 1992.

[40] Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. Here are my data files. here are my queries. where are my results? In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 57–68, 2011.

[41] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1: Reference Description, 2012.

[42] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[43] Thomas Jörg and Stefan Dessloch. Near real-time data warehousing using state-of-the-art ETL tools. In *Enabling Real-Time Business Intelligence -*

Third International Workshop, BIRTE 2009, Held at the 35th International Conference on Very Large Databases, VLDB 2009, Lyon, France, August 24, 2009, Revised Selected Papers, pages 100–117, 2009.

[44] Niranjan Kamat and Arnab Nandi. A session-based approach to fast-but-approximate interactive data cube exploration. *ACM Trans. Knowl. Discov. Data*, 12(1):9:1–9:26, 2018.

[45] Yagiz Karæz, Milena Ivanova, Ying Zhang, Stefan Manegold, and Martin L. Kersten. Lazy ETL in action: ETL technology dates scientific data. *PVLDB*, 6(12):1286–1289, 2013.

[46] Yagiz Kargin, Holger Pirk, Milena Ivanova, Stefan Manegold, and Martin L. Kersten. Instant-on scientific data warehouses - lazy ETL for data-intensive research. In *Enabling Real-Time Business Intelligence - 6th International Workshop, BIRTE 2012, Held at the 38th International Conference on Very Large Databases, VLDB 2012, Istanbul, Turkey, August 27, 2012, Revised Selected Papers*, pages 60–75, 2012.

[47] Yannis Katsis and Yannis Papakonstantinou. View-based data integration. In *Encyclopedia of Database Systems, Second Edition*. 2018.

[48] Yasar Khan, Antoine Zimmermann, Alokkumar Jha, Vijay Gadepally, Mathieu d'Aquin, and Ratnesh Sahay. One size does not fit all: Querying web polystores. *IEEE Access*, 7:9598–9617, 2019.

[49] Selma Khouri, Ilyès Boukhari, Ladjel Bellatreche, Eric Sardet, Stéphane Jean, and Mickaël Baron. Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool. *Comput. Ind.*, 63(8):799–812, 2012.

[50] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[51] Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley Publishing, 2004.

[52] Ralph Kimball and Richard Merz. *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*. John Wiley & Sons, Inc., 2000.

[53] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley Publishing, 2nd edition, 2002.

[54] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley Publishing, 3rd edition, 2013.

[55] Dimitrios Kouzis-Loukas. *Learning Scrapy*. Packt Publishing Ltd, 2016.

[56] A. J. Lee, A. Nica, and E. A. Rundensteiner. The eve approach: view synchronization in dynamic distributed environments. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):931–954, 2002.

[57] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246, 2002.

[58] Sebastian Link and Henri Prade. Relational database schema design for uncertain data. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1211–1220. ACM, 2016.

[59] Jun Liu, Mark Roantree, and Zohra Bellahsene. Optimizing XML data with view fragments. In *Database Technologies 2010, Twenty-First Australasian*

Database Conference (ADC 2010), Brisbane, Australia, 18-22 January, 2010, Proceedings, pages 151–159, 2010.

[60] Jun Liu, Mark Roantree, and Zohra Bellahsene. A schemaguide for accelerating the view adaptation process. In *Conceptual Modeling - ER 2010, 29th International Conference on Conceptual Modeling, Vancouver, BC, Canada, November 1-4, 2010. Proceedings*, pages 160–173, 2010.

[61] Hadj Mahboubi and Jérôme Darmont. Enhancing xml data warehouse query performance by fragmentation. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1555–1562, 2009.

[62] Yann Le Mansec, Benjamin Pageaux, Antoine Nordez, Sylvain Dorel, and Marc Jubeau. Mental fatigue alters the speed and the accuracy of the ball in table tennis. *Journal of Sports Sciences*, 36(23):2751–2759, 2018. PMID: 29260619.

[63] Silvano Martello and Paolo Toth. Subset-sum problem. In *Knapsack problems : algorithms and computer implementations*, chapter 4, pages 105–136. Wiley-Interscience, 1990.

[64] Andrew McCarren, Suzanne McCarthy, Conor O. Sullivan, and Mark Roantree. Anomaly detection in agri warehouse construction. In *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017, Geelong, Australia, January 31 - February 3, 2017*, pages 17:1–17:10, 2017.

[65] Suzanne McCarthy, Andrew McCarren, and Mark Roantree. Predicting prices, volume and trade for meat and dairy products: an agri food data model. Technical report, Dublin City University, 2017.

[66] Suzanne McCarthy, Andrew McCarren, and Mark Roantree. An architecture and services for constructing data marts from online data sources. Technical report, Dublin City University, 2018.

[67] Suzanne McCarthy, Andrew McCarren, and Mark Roantree. An automated etl for online datasets. Technical report, Dublin City University, 2018.

[68] Suzanne McCarthy, Andrew McCarren, and Mark Roantree. Combining web and enterprise data for lightweight data mart construction. In *Database and Expert Systems Applications - 29th International Conference, DEXA 2018, Regensburg, Germany, September 3-6, 2018, Proceedings, Part II*, pages 138–146, 2018.

[69] Suzanne McCarthy, Andrew McCarren, and Mark Roantree. A method for automated transformation and validation of online datasets. In *23rd IEEE International Enterprise Distributed Object Computing Conference, EDOC 2019, Paris, France, October 28-31, 2019*, pages 183–189, 2019.

[70] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[71] Todd D. Millstein, Alon Y. Halevy, and Marc Friedman. Query containment for data integration systems. *J. Comput. Syst. Sci.*, 66(1):20–39, 2003.

[72] K. I. Mohammed. Data warehouse design and implementation based on star schema vs. snowflake schema. *International Journal of Academic Research in Business and Social Sciences*, 9:25–38, 2019.

[73] Mukesh Mohania. Avoiding re-computation: View adaptation in data warehouses. In *Proceedings of the 8th International Database Workshop*, pages 151–165, 11 1997.

[74] Jindrich Mynarz, Jakub Klímek, Marek Dudás, Petr Skoda, Christiane Engels, Fathoni A. Musyaffa, and Vojtech Svátek. Reusable transformations of data cube vocabulary datasets from the fiscal domain. In *Proceedings of the 4th International Workshop on Semantic Statistics co-located with 15th International Semantic Web Conference, SemStats@ISWC 2016, Kobe, Japan October 18, 2016*, 2016.

[75] Duane Q. Nykamp. Proper subset definition. `http://mathinsight.org/definition/proper_subset`.

[76] Stats NZ. https://www.stats.govt.nz/, 2019.

[77] United States Department of Agriculture. http://www.ers.usda.gov/data-products/chart-gallery/detail.aspx?chartId=40037, 2019.

[78] Juan Manuel Pérez, Rafael Berlanga Llavori, María José Aramburu, and Torben Bach Pedersen. Integrating data warehouses with web data: A survey. *IEEE Trans. Knowl. Data Eng.*, 20(7):940–955, 2008.

[79] Barbara Pernici and Monica Scannapieco. *Data Quality in Web Information Systems*, pages 48–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[80] Pig333.com. https://www.pig333.com/. Accessed on 2020-03-20.

[81] Charnyote Pluempitiwiriyawej and Joachim Hammer. A classification scheme for semantic and schematic heterogeneities in xml data sources. Technical report, 2000.

[82] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis, Alkis Simitsis, and Nils-Erik Frantzell. Supporting streaming updates in an active data warehouse. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 476–485, 2007.

[83] Quandl. https://www.quandl.com/, 2019.

[84] Mark Roantree and Jun Liu. A heuristic approach to selecting views for materialization. *Softw. Pract. Exp.*, 44(10):1157–1179, 2014.

[85] Oscar Romero, Alkis Simitsis, and Alberto Abelló. GEM: requirement-driven generation of ETL and multidimensional conceptual designs. In *Data Warehousing and Knowledge Discovery - 13th International Conference, DaWaK 2011, Toulouse, France, August 29-September 2,2011. Proceedings*, pages 80–95, 2011.

[86] V Rozand, F Lebon, C Papaxanthis, and R Lepers. Effect of mental fatigue on speed–accuracy trade-off. *Neuroscience*, 297:219–230, 2015.

[87] Elke A Rundensteiner, Andreas Koeller, and Xin Zhang. Maintaining data warehouses over changing information sources. *Communications of the ACM*, 43(6):57–62, 2000.

[88] Giovanni Maria Sacco. Fragmentation: A technique for efficient query processing. *ACM Trans. Database Syst.*, 11(2):113–133, 1986.

[89] Rashed Salem, Omar Boussaïd, and Jérôme Darmont. Active xml-based web data integration. *Information Systems Frontiers*, 15(3):371–398, 2013.

[90] Ricardo Jorge Santos and Jorge Bernardino. Real-time data warehouse loading methodology. In *12th International Database Engineering and Applications Symposium (IDEAS 2008), September 10-12, 2008, Coimbra, Portugal*, pages 49–58, 2008.

[91] Michael Scriney, Suzanne McCarthy, Andrew McCarren, Paolo Cappellari, and Mark Roantree. Automating data mart construction from semi-structured data sources. 2018. To appear in the Computer Journal, Oxford University Press, 2018.

[92] Michael Scriney, Martin F. O'Connor, and Mark Roantree. Integrating online data for smart city data marts. In *Data Analytics - 31st British International Conference on Databases, BICOD 2017, London, UK, July 10-12, 2017, Proceedings*, pages 23–35, 2017.

[93] Dimitrios Skoutas and Alkis Simitsis. Designing etl processes using semantic web technologies. In *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, pages 67–74. ACM, 2006.

[94] Dimitrios Skoutas and Alkis Simitsis. Ontology-based conceptual design of ETL processes for both structured and semi-structured data. *Int. J. Semantic Web Inf. Syst.*, 3(4):1–24, 2007.

[95] Dimitrios Skoutas, Alkis Simitsis, and Timos K. Sellis. Ontology-driven conceptual design of ETL processes using graph transformations. *J. Data Semantics*, 13:120–146, 2009.

[96] Mitchell R Smith, Linus Zeuwts, Matthieu Lenoir, Nathalie Hens, Laura MS De Jong, and Aaron J Coutts. Mental fatigue impairs soccer-specific decision-making skill. *Journal of sports sciences*, 34(14):1297–1304, 2016.

[97] UN Stats. Harmonized commodity description and coding systems (hs). `https://unstats.un.org/unsd/tradekb/Knowledgebase/50018/Harmonized-Commodity-Description-and-Coding-Systems-HS`, 2019.

[98] Tesco. `https://www.tesco.ie/groceries/SpecialOffers/default.aspx`, 2019.

[99] Vasileios Theodorou, Alberto Abelló, Wolfgang Lehner, and Maik Thiele. Quality measures for ETL processes: from goals to implementation. *Concurr. Comput. Pract. Exp.*, 28(15):3969–3993, 2016.

[100] C. Thomsen, T. B. Pedersen, and W. Lehner. Rite: Providing on-demand data for right-time data warehousing. In *2008 IEEE 24th International Conference on Data Engineering*, pages 456–465, 2008.

[101] Frank S. C. Tseng and Chia-Wei Chen. Integrating heterogeneous data warehouses using XML technologies. *J. Inf. Sci.*, 31(3):209–229, 2005.

[102] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. Modeling ETL activities as graphs. In *Design and Management of Data Warehouses 2002, Proceedings of the 4th Intl. Workshop DMDW'2002, Toronto, Canada, May 27, 2002*, pages 52–61, 2002.

[103] Boris Vrdoljak, Marko Banek, and Zoran Skocir. Integrating XML sources into a data warehouse. In *Data Engineering Issues in E-Commerce and Services, Second International Workshop, DEECS 2006, San Francisco, CA, USA, June 26, 2006, Proceedings*, pages 133–142, 2006.

[104] Florian Waas, Robert Wrembel, Tobias Freudenreich, Maik Thiele, Christian Koncilia, and Pedro Furtado. On-demand ELT architecture for right-time BI: extending the vision. *IJDWM*, 9(2):21–38, 2013.

[105] Richard Y. Wang and Diane M. Strong. Beyond accuracy: What data quality means to data consumers. *J. of Management Information Systems*, 12(4):5–33, 1996.

[106] A. Wibowo. Problems and available solutions on the stage of extract, transform, and loading in near real-time data warehousing (a literature study). In *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 345–350, May 2015.

[107] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Springer-Verlag London, UK, 2000.

[108] Nengfu Xie. Research on the inconsistency checking in agricultural knowledge base. In *Computer and Computing Technologies in Agriculture VI - 6th IFIP WG 5.14 International Conference, CCTA 2012, Zhangjiajie, China, October 19-21, 2012, Revised Selected Papers, Part I*, pages 290–296, 2012.

[109] Nengfu Xie, Wensheng Wang, Bingxian Ma, Xuefu Zhang, Wei Sun, and Fenglei Guo. Research on an agricultural knowledge fusion method for big data. In *Data Science Journal*, 2015.

[110] Ying Yang. On-demand query result cleaning. In *Proceedings of the VLDB 2014 PhD Workshop.*, 2014.

[111] Ying Yang, Niccolò Meneghetti, Ronny Fehling, Zhen Hua Liu, and Oliver Kennedy. Lenses: An on-demand approach to ETL. *PVLDB*, 8(12):1578–1589, 2015.

[112] Sana Yousuf and Sanam Shahla Rizvi. A comparative study of etl tools. In *3rd International Conference on Computer and Automation Engineering*, volume 1, pages 251–255, 2011.

[113] Huang Yu, Zhang Xiao-yi, Yuan Zhen, and Jiang Guo-quan. A universal data cleaning framework based on user model. In *Proceeding of the IEE International Colloquium on Computing, Communication, Control and Management (ISECS)*, pages 200 – 202, 09 2009.

[114] Youchan Zhu, Lei An, and Shuangxi Liu. Data updating and query in real-time data warehouse system. In *International Conference on Computer Science and Software Engineering, CSSE 2008, Volume 5: E-learning and Knowledge Management / Socially Informed and Instructinal Design / Learning Systems Platforms and Architectures / Modeling and Representation / Other Applications , December 12-14, 2008, Wuhan, China*, pages 1295–1297, 2008.

[115] ZuivelNL. `https://www.zuivelnl.org/`, 2019.

Appendices

Appendix A

Ruleset

id	attribute_name	source_term	standard_term	conversion
1	any			1
2	price	eur/tonne	EUR/KG	0.001
3	price	euro per ton	EUR/KG	0.001
4	trade_weight	100KG	KG	100
5	price	USD/MT	USD/KG	0.001
6	price	US - $/lb	USD/KG	2.20462
7	trade_value	VALUE_1000EURO	EUR	1000
8	trade_weight	QUANTITY_TON	KG	1000
9	trade_weight	kgm	KG	1
10	trade_weight	kg	KG	1
11	trade_weight	MT	KG	1000
12	trade_value	Dollars	USD	1
13	trade_value	US$	USD	1
14	trade_value	Can$	CAD	1
15	weight	Thousand tonnes	KG	1000000
16	content	Fat content (% of product weight)	Fat content	1
17	content	Protein content (% of product weight)	Protein content	1
18	slaughter	Thousand head (animals)	head	1000

19	price	euro/100kg	EUR/KG	0.01
20	weight	Tonnes	KG	1000
21	price	USD per tonne	USD/KG	0.001
22	price	pound per tonne	BPS/KG	0.001
23	weight	million litres	LITRES	1000000
24	price	$/cwt	USD/KG	50.80235
25	price	EUR/100kg	EUR/KG	0.01
26	weight	thousand tons	KG	1000000
27	price	euro per head	EUR/head	1
28	slaughter	1000 head	head	1000
29	slaughter	1 head	head	1
30	weight	tons	KG	1000
31	price	Yen\kg	YEN/KG	1
32	weight	kg/carcass	KG/carcass	1
33	price	€ euro/100kg	EUR/KG	0.01
34	slaughter	thou. head	head	1000
35	weight	tonnes	KG	1000
36	weight	Kg	KG	1
37	weight	Kg per carcass	KG/carcass	1
38	weight	millions of litres	LITRES	1000000
39	price	euro_per_100kg	EUR/KG	0.01
40	price	EUR_per_kg	EUR/KG	1
41	weight	Litres	LITRES	1
42	count	Percent	percent	1
43	count	Percentage	percent	1
44	price	eur/100kg	EUR/KG	0.01
45	weight	TONNES	KG	1000
46	slaughter	1000 HEAD	head	1000
47	count	PERCENT	percent	1

48	weight	1000 MT CWE	KG	1000000
49	price	EUR_per_100kg	EUR/KG	0.01
50	population	Persons	persons	1
51	currency	GBP	GBP	1
52	count	EA	EACH	1
53	count	EACH	EACH	1
54	currency	DKK	DKK	1
55	count	DOZ	dozen	1
56	count	Number	number	1
57	weight	million liters	LITRES	1000000
58	price	euro per 100kg	EUR/KG	0.01
59	weight	thousand tonnes	KG	1000000
60	price	Yen/kg	YEN/KG	1
61	price	$/mt	USD/KG	0.001
62	currency	NZD	NZD	1
63	price	cent	USD	0.01
64	price	cent	EUR	0.01
65	price	â¬ euro/100kg	EUR/KG	0.01
66	price	cent/kg	EUR/KG	100
67	slaughter	thousand	head	1000
68	price	dollar per ton	USD/KG	0.001
69	price	DKK_per_100kg	DKK/KG	0.01
70	price	USD_per_100kg	USD/KG	0.01
71	price	GBX_per_100kg	GBX/KG	0.01
72	price	HUF_per_100kg	HUF/KG	0.01
73	price	CNY_per_100kg	CNY/KG	0.01
74	price	CLP_per_100kg	CLP/KG	0.01
75	price	PLN_per_100kg	PLN/KG	0.01
76	price	UAH_per_100kg	UAH/KG	0.01

77	price	CAD_per_100kg	CAD/KG	0.01
78	weight	million kg	KG	1000000
79	price	$/kg	USD/KG	1

Appendix B

Import Template Examples

data_address	scrape_date	source	measure	frequency	industry	transformed	loaded	queried	num_cols	num_valid_cols	num_rows	header_start	dim_col_list	fact_col_list	skip_rows
G:\My Drive\Ph	11_09_2018	agricultur	price	weekly	cattle	N	N	N	13	8	200270	0	3,4,5,6,7,8	9	NULL
G:\My Drive\Ph	11_09_2018	agricultur	price	weekly	cattle	N	N	N	13	8	117245	0	3,4,5,6,7,8	9	NULL
G:\My Drive\Ph	11_09_2018	alic	slaughters	monthly	pigs	N	N	N	9	5	327	0	1,4,5	2,3	NULL
G:\My Drive\Ph	11_09_2018	alic	price	monthly	pigs	N	N	N	9	5	1806	0	1,3,4,5	2	NULL
G:\My Drive\Ph	11_09_2018	alic	tradeweig	monthly	pigs	N	N	N	9	5	1890	0	1,3,4,5	2	NULL
G:\My Drive\Ph	11_09_2018	alic	tradeweig	monthly	pigs	N	N	N	9	5	3648	0	1,3,4,5	2	NULL
G:\My Drive\Ph	11_09_2018	ausbureal	slaughters	monthly	cattle	N	N	N	11	6	14076	0	2,1,4,5,6	3	NULL
G:\My Drive\Ph	11_09_2018	ausbureal	slaughters	monthly	cattle	N	N	N	11	6	8740	0	2,1,4,5,6	3	NULL
G:\My Drive\Ph	11_09_2018	bordbia	price	weekly	cereal	N	N	N	10	5	5384	0	1,2,3,5	4	NULL
G:\My Drive\Ph	11_09_2018	bordbia	price	weekly	dairy	N	N	N	10	5	8237	0	2,3,4,6	5	NULL
G:\My Drive\Ph	11_09_2018	bordbia	slaughters	weekly	pigs	N	N	N	10	5	5634	0	1,2,3,5	4	NULL
G:\My Drive\Ph	11_09_2018	bordbia	price	weekly	pigs	N	N	N	10	5	24990	0	1,2,3,5	4	NULL
G:\My Drive\Ph	11_09_2018	bpex	slaughters	annual	pigs	N	N	N	11	5	162	0	2,1,4,5	3	NULL
G:\My Drive\Ph	11_09_2018	bpex	slaughters	weekly	pigs	N	N	N	10	5	3570	0	2,1,4,5	3	NULL
G:\My Drive\Ph	11_09_2018	bpex	consumpt	annual	pigs	N	N	N	10	5	80	0	1,2,3,5	4	NULL
G:\My Drive\Ph	11_09_2018	bpex	price	weekly	pigs	N	N	N	10	4	381	0	1,3,4	2	NULL
G:\My Drive\Ph	11_09_2018	clal	productio	monthly	dairy	N	N	N	10	5	3024	0	2,3,4,5	6	NULL
G:\My Drive\Ph	11_09_2018	clal	price	monthly	dairy	N	N	N	13	5	912	0	4,5,6,7	8	NULL
G:\My Drive\Ph	11_09_2018	clal	productio	monthly	dairy	N	N	N	10	5	480	0	4,5,6,7	8	NULL
G:\My Drive\Ph	11_09_2018	clal	price	weekly	dairy	N	N	N	13	5	199	0	2,3,4,5	6	NULL
G:\My Drive\Ph	11_09_2018	cme	multiple	monthly	multiple	N	N	N	16	11	100	0	2,3,10	4,5,6,7,8,9,11	NULL
G:\My Drive\Ph	11_09_2018	comtrade	tradeweig	monthly	meat	N	N	N	18	10	7656	0	2,5,6,8,9,7,12,	10,11	NULL
G:\My Drive\Ph	11_09_2018	comtrade	tradeweig	monthly	meat	N	N	N	18	10	1694	0	2,5,6,8,9,7,12,	10,11	NULL
G:\My Drive\Ph	11_09_2018	dairyaustr	productio	monthly	dairy	N	N	N	12	5	651	0	4,5,6,8	7	NULL
G:\My Drive\Ph	11_09_2018	dairyaustr	productio	monthly	dairy	N	N	N	12	5	528	0	4,5,6,8	7	NULL
G:\My Drive\Ph	11_09_2018	dairyaustr	sales	monthly	dairy	N	N	N	12	5	336	0	4,5,6,8	7	NULL
G:\My Drive\Ph	11_09_2018	dairyaustr	sales	monthly	dairy	N	N	N	12	5	288	0	4,5,6,8	7	NULL
G:\My Drive\Ph	11_09_2018	dairyco	productio	monthly	dairy	N	N	N	12	5	48	0	4,5,6,8	7	NULL
G:\My Drive\Ph	11_09_2018	dairyco	productio	daily	dairy	N	N	N	10	5	365	0	2,3,4,6	5	NULL
G:\My Drive\Ph	11_09_2018	dairyco	price	monthly	dairy	N	N	N	10	5	990	0	2,3,4,6	5	NULL

Appendix C

DataMap Example

attr_type	rule_id	source_term	standard_term	dimension	dim_attr
D		yearmonth	yearmonth	dim_date_monthly	yearmonth
D		201301	201301	dim_date_monthly	yearmonth
D		201302	201302	dim_date_monthly	yearmonth
D		201111	201111	dim_date_monthly	yearmonth
D		201112	201112	dim_date_monthly	yearmonth
D		201701	201701	dim_date_monthly	yearmonth
D		201702	201702	dim_date_monthly	yearmonth
D		201703	201703	dim_date_monthly	yearmonth
D		201704	201704	dim_date_monthly	yearmonth
D		201705	201705	dim_date_monthly	yearmonth
D		201706	201706	dim_date_monthly	yearmonth
D		201707	201707	dim_date_monthly	yearmonth
D		201708	201708	dim_date_monthly	yearmonth
D		201709	201709	dim_date_monthly	yearmonth
D		201710	201710	dim_date_monthly	yearmonth
D		201711	201711	dim_date_monthly	yearmonth
D		201712	201712	dim_date_monthly	yearmonth
D		201201	201201	dim_date_monthly	yearmonth
D		201202	201202	dim_date_monthly	yearmonth
D		201203	201203	dim_date_monthly	yearmonth
D		201204	201204	dim_date_monthly	yearmonth
D		201205	201205	dim_date_monthly	yearmonth
D		201206	201206	dim_date_monthly	yearmonth
D		201207	201207	dim_date_monthly	yearmonth
D		201208	201208	dim_date_monthly	yearmonth
D		201209	201209	dim_date_monthly	yearmonth
D		201210	201210	dim_date_monthly	yearmonth
D		201211	201211	dim_date_monthly	yearmonth
D		201212	201212	dim_date_monthly	yearmonth
D		area	geo	dim_geo	geo
D		Oceania	OCEANIA	dim_geo	geo
D		USA	UNITED STATES	dim_geo	geo
D		France	FRANCE	dim_geo	geo
D		Germany	GERMANY	dim_geo	geo
D		china	CHINA	dim_geo	geo
D		argentina	ARGENTINA	dim_geo	geo
D		new zealand	NEW ZEALAND	dim_geo	geo
D		type	product	dim_product	product
D		Butter	BUTTER	dim_product	product
D		WMP	WMP	dim_product	product
D		SMP	SMP	dim_product	product
D		Caseins	Caseins	dim_product	product
D		Whey	Whey	dim_product	product
D		milk	milk	dim_product	product
D		unit	unit	dim_unit	unit_desc
D	3	euro per ton	EUR/KG	dim_unit	unit_desc
D	58	euro per 100kg	EUR/KG	dim_unit	unit_desc
D	68	dollar per ton	USD/KG	dim_unit	unit_desc
F		value	price		

Appendix D

Algorithms

Algorithm 4.1 Create CubeMap

```
1: function CreateCubeMap(C, F, CDM)
2:     Initialise CubeMap CM = ⟨cm_id⟩
3:     CM.cm_id ←′ CM′ + C.cube_id
4:     for i ∈ C.headers do
5:         Initialise CubeVector CV = ⟨name, type, min, max, has_nulls, valueset⟩
6:         CV.name ← i
7:         if i ∈ CDM.date_dimensions then
8:             CV.type ← date
9:         else if i ∈ CDM.measures then
10:            CV.type ← numerical
11:            if |F| > 0 then
12:                Initialise CV.Function_Spec
13:                Function_Spec.function_type ← F.function_type
14:                Function_Spec.group ← F.group
15:                Function_Spec.group_order ← F.group_order
16:            end if
17:        else
18:            CV.type ← string
19:        end if
20:        V ← set{i.values}
21:        if null ∈ V then
22:            CV.has_nulls ← True
23:        else
24:            CV.has_nulls ← False
25:        end if
26:        if CV.type ∈ (date, numerical) then
27:            CV.min ← minimum(V)
28:            CV.max ← maximum(V)
29:            CV.valueset ← null
30:        else
31:            CV.min ← null
32:            CV.max ← null
33:            CV.valueset ← V
34:        end if
35:        CM = CM + CV
36:    end for
37: end function
```

Algorithm 4.2 Create QueryMap

```
1: function CreateQueryMap(Q, CDM)
2:     Initialise QueryMap QM = ⟨qm_id⟩
3:     QM.cm_id ←′ qm′ + Q.query_id
4:     for i ∈ Q.RA do
5:         Initialise CubeVector CV = ⟨name, type, min, max, has_nulls, valueset⟩
6:         CV.name ← i
7:         for rv_i ∈ Q.RV do
8:             if rv_i.A ∈ CDM.date_dimensions then
9:                 CV.type ← date
10:            else if rv_i.A ∈ CDM.measures then
11:                if Q.F then
12:                    Initialise FunctionSpec FS = ⟨function_type,
13:                    CubeVector.name, group, group_order⟩
14:                end if
15:                CV.type ← numerical
16:            else
17:                CV.type ← string
18:            end if
19:            if rv_i.valueset =!null then
20:                CV.has_nulls ← False
21:            else
22:                CV.has_nulls ← True
23:            end if
24:            V ← set{rv_i.valueset}
25:            if CV.type ∈ (date, numerical) then
26:                CV.min ← minimum(V)
27:                CV.max ← maximum(V)
28:                CV.valueset ← null
29:            else
30:                CV.min ← null
31:                CV.max ← null
32:                CV.valueset ← V
33:            end if
34:        end for
35:    end for
36:    return QM
37: end function
```

Algorithm 4.3 Fragment Query

```
1: function FragmentQuery(Q)
2:     Initialise QF=[]
3:     for r ∈ ra do
4:         if r.attribute ⊄ rv then
5:             QF = QF + (r.attribute, ∗)
6:         end if
7:     end for
8:     for r ∈ rv do
9:         Initialise QF_i = ⟨attribute, value⟩
10:        QF_i.attribute ← r.attribute
11:        for v ∈ r.value do
12:            QF_i.value ← v
13:            QF = QF + QF_i
14:        end for
15:    end for
16:    return QF
17: end function
```

Algorithm 4.4 Create Cube Matrix

```
1: function CreateMatrix(field_names)
2:     Initialise CubeMatrix
3:     CubeMatrix.field_names =[dimension, dimension_attribute,
4:     dimension_valueset, cube_id, cube_vector, intersect]
5:     return CubeMatrix
6: end function

7: function PopulateMatrix1(CM, D, M)
8:     for d ∈ D do
9:         CM.dimension ← d
10:        From d get attributes
11:        CM.dimension_attribute ← d.attributes
12:        From d get valueset
13:        CM.dimension_valueset ← d.valueset
14:    end for
15:    return CubeMatrix
16: end function

17: function PopulateMatrix2(CM, C)
18:    for c ∈ C do
19:        CM.cube_id ← c.cube_id
20:        From c get CubeVectors
21:        for cv ∈ CubeVectors do
22:            CM.cube_vector ← cv
23:            cube_valueset ← cv.valueset
24:            CM.intersect ← cube_valueset ∩ CM.dimension_valueset
25:        end for
26:    end for
27: end function
```

Algorithm 4.5 Check Containment

```
1: function CheckContainmentCont(CubeMap, valueset)
2:     cube_min ← CubeVector.min
3:     cube_max ← CubeVector.max
4:     if cube_min ≤ min(valueset) and cube_max ≥ max(valueset) then
5:         return True
6:     else
7:         return False
8:     end if
9: end function

10: function CheckContainmentDim(CM.valueset, QM.valueset)
11:     if QM.valueset ⊆ CM.valueset then
12:         return True
13:     else
14:         return False
15:     end if
16: end function

17: function CheckContainmentFunc(CubeMap, F)
18:     C_F ← CubeMap.Function
19:     if C_F.function_type ≠ F.function_type then
20:         return False
21:     else if C_F.cubevector ≠ F.cubevector then
22:         return False
23:     else if C_F.group ≠ F.group then
24:         return False
25:     else if C_F.group_order ≠ F.group_order then
26:         return False
27:     else
28:         return True
29:     end if
30: end function
```

Appendix E

Dynamic Data Cubes Method Demonstration

Example E.0.1. Extraction from Data Lake

S={

(PERIOD,201708),

(DECLARANT_LAB,France),

(PARTNER_LAB,Netherlands),

(FLOW_LAB,EXPORT),

(PRODUCT,2011000),

(PRODUCT_LAB,CARCASES OR HALF-CARCASES OF BOVINE ANIMALS, FRESH OR CHILLED),

(INDICATORS,VALUE_1000EURO),

(INDICATOR_VALUE,828.68),

(PERIOD,201708),

(DECLARANT_LAB,France),

(PARTNER_LAB,Netherlands),

(FLOW_LAB,EXPORT),

(PRODUCT,2011000),

(PRODUCT_LAB,CARCASES OR HALF-CARCASES OF BOVINE ANIMALS,

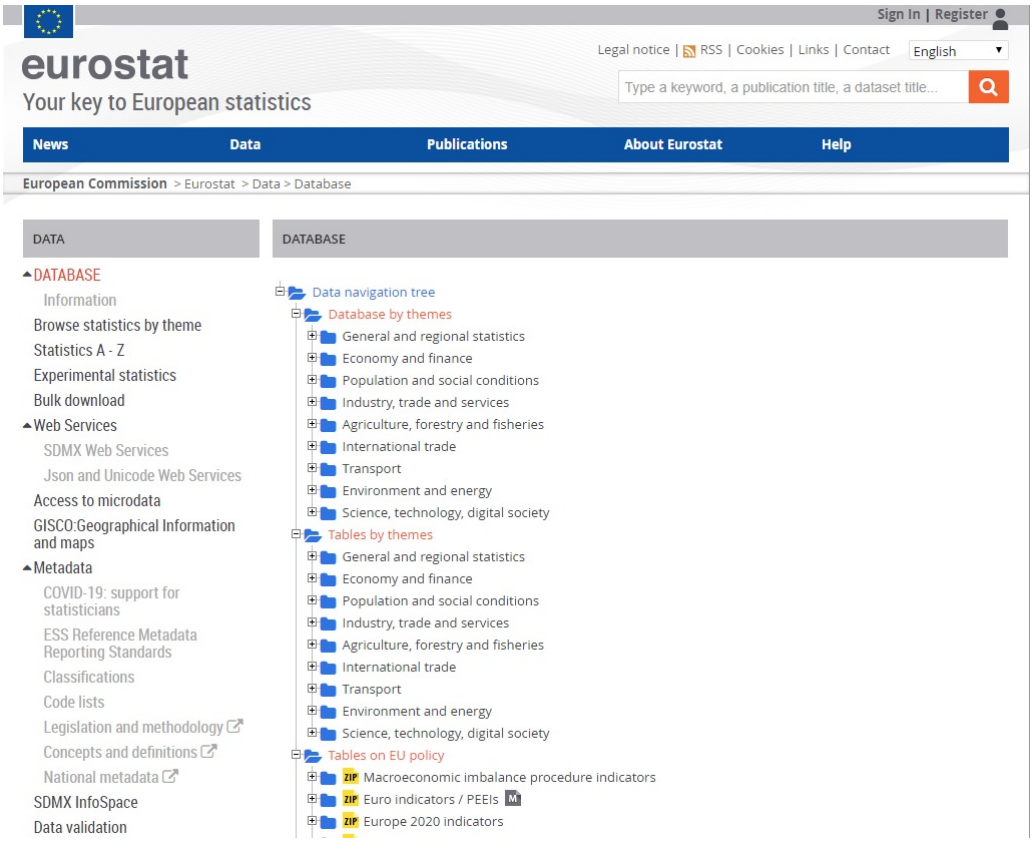

Figure E.1: Eurostat Web Portal

DEC_LAB	PARTNER_LAB	PRODUCT	PRODUCT_LAB	FLOW_LAB	PERIOD	INDICATORS	INDIC_VALUE
France	Netherlands	2011000	CARCASES OR HALF-CARCASES OF BOVINE...	EXPORT	201708	VALUE_1000EURO	828.68
France	Netherlands	2011000	CARCASES OR HALF-CARCASES OF BOVINE...	EXPORT	201708	QUANTITY_TON	198.6
France	Netherlands	2011000	CARCASES OR HALF-CARCASES OF BOVINE...	EXPORT	201709	VALUE_1000EURO	773.63
France	Netherlands	2011000	CARCASES OR HALF-CARCASES OF BOVINE...	EXPORT	201709	QUANTITY_TON	184.1

Figure E.2: Eurostat imported data

FRESH OR CHILLED),
(INDICATORS,QUANTITY_TON),
(INDICATOR_VALUE,198.6), ... }

Example E.0.2. Eurostat data after Transformation
S'={ (yearmonth,201708),
(reporter,FRANCE),
(partner,NETHERLANDS),
(flow,exports),
(product_code,2011000),
(product_desc,CARCASES OR HALF-CARCASES OF BOVINE ANIMALS, FRESH OR CHILLED),
(unit,EURO),
(value,828680),
(yearmonth,201708),
(reporter,FRANCE),
(partner,NETHERLANDS),
(flow,exports),
(product_code,2011000),
(product_desc,CARCASES OR HALF-CARCASES OF BOVINE ANIMALS, FRESH OR CHILLED),
(unit,KG),
(value,198600) }

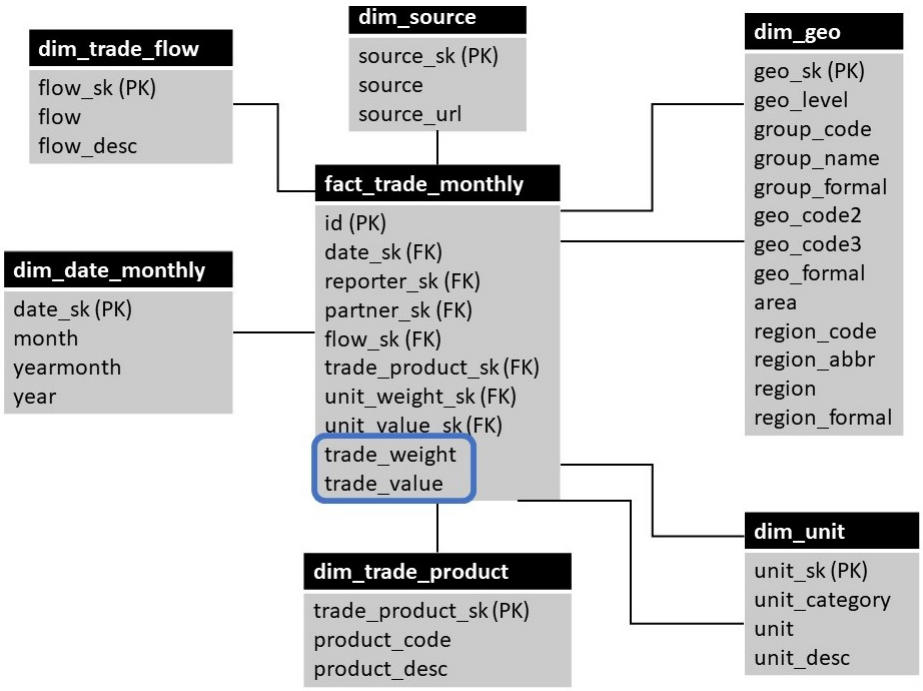

Figure E.3: Agri Trade data mart