AALBORG UNIVERSITET

Student Report

The Department of Computer Science
Selma Lagerlöfs Vej 300
Telephone +45 9940 9940
Fax +45 9940 9798
http://www.cs.aau.dk/

Title:
Modeling Linkable Multidimensional Cubes with Logical Patterns

Theme:
Database Technology Specialization

Project period:
SW10, spring semester 2014

Project group:
sw102f14

Participants:

Alex Bondo Andersen

Kim Ahlstrøm Jakobsen

Supervisors:
Torben Bach Pedersen
Katja Hose

Page count: 81

Appendix count: 7

Finished: 4/6 – 2014

Abstract:

Traditional Decision Support Systems (DSS) rely on local data to answer business questions. This limits the decision maker to a simplified world. Ad-hoc data integration enables the decision maker to answer more business questions because the local data is enriched with situationally appropriate data. Linked data in Resource Description Framework (RDF) format is attractive as situational data because of its growing amount and its interconnectivity between sources. Several systems have been proposed to facilitate ad-hoc data integration and there is a high potential for further research. We explore the potential of storing the local data in RDF – as well as the situational data. We show how business questions can be answered on RDF formatted cubes annotated with the QB4OLAP vocabulary, called Linkable Multidimensional (LMD) cubes. We introduce three logical patterns for LMD cubes, which have different characteristics with regards to load time, storage size, and query evaluation time. Our novel algorithm Semantic Web OLAP Denormalizer (SWOD) is used to convert cubes of one pattern to another. The well-known TPC Benchmark^{TM}H (TPC-H) dataset is converted to RDF and described as LMD cubes. We evaluate our patterns on these cubes, with queries based on the TPC-H business questions.

Preface

This report is structured as a scientific paper with a number appendices, which present minor details and contributions. The appendix presenting Semantic Web (Appendix A) originates from our previous semester project [3], with very few changes. In Appendix G we present the TPC-H query names, descriptions, and business questions exactly as they are in the specification [33].

The source code for our SWOD algorithm is located at a GIT repository[1]. The source code for our load program is located at a GIT repository[2]. The scripts we use to orchestrate the loading and querying of our data are located at a GIT repository[3].

We publish our cubes on the SPARQL endpoint `http://extbi.lab.aau.dk/sparql` in the named graph `http://extbi.lab.aau.dk/resource/ltpch/` with the ontology describing them in the named graph `http://extbi.lab.aau.dk/ontology/ltpch/`.

[1] `https://github.com/abondoa/swod.git`
[2] `https://github.com/kimajakobsen/sesame-repository-test-workspace.git`
[3] `https://github.com/abondoa/lmd-cubes.git`

Resumé – Dansk

I dette projekt udforsker vi muligheden for at bruge RDF som opbevaringsformat for multidimensionelle kuber. Dette giver mulighed for at bruge SPARQL queries til at udføre OLAP operationer. Fordelen ved at bruge SPARQL frem for andre query sprog, er at SPARQL har indbygget funktionalitet til at lave ad-hoc data integration gennem query federationer. Ad-hoc data integration er nyttigt i sammenhænge hvor man ikke på forhånd kan vide hvilke forretnings spørgsmål man kan blive stillet overfor. Traditionelt indsamler man data i et data warehouse (DW) og evaluerer sine forretnings spørgsmål, i form af queries, derpå. Det er både dyrt og tidskrævende at opbygge denne proces (kaldet ETL), der indsamler data og gemmer det i sit DW. Ved at supportere ad-hoc data integration kan man tillade udforskning af nye data kilder der har relevance for et forretnings spørgsmål uden at skulle lave store investeringer. Derudover kan et svar også forventes langt hurtigere ved ad-hoc data integration ved brug af SPARQL end hvis man skulle til at ændre sin ETL proces.

I forbindelse med OLAP bruger man i den relationelle verden nogle forskellige skema teknikker til at modellere sine kuber. Selvom der findes vokabulare der beskriver kuber i RDF format er der ikke tidligere undersøgt og beskrevet hvordan relationelle modellerings tekniker kan bruges på RDF kuber. Vi beskriver hvordan disse teknikker bruges på RDF data, der er beskrevet med et vokabular, beregnet til at beskrive OLAP kuber, QB4OLAP. Vi siger at en kube er i Snowflake-pattern hvis hver level instans i hver dimension er repræsenteret som en ressource. Hvis en kube i stedet har en ressource for hver dimensions instans siger vi at den er i Star-pattern. Hvis der hverken er level eller dimension instanser, men facts er direkte tilknyttet dimensions attributterne, er kuben i Denormalized-pattern.

Vi designer og implementerer en algoritme til at konvertere en kube fra Snowflake-pattern til Star-pattern eller Denormalized-pattern. Denne algoritme kan håndtere kuber med multiple hierarkier og ubalancerede hierarkier i dimensioner. Algoritmen konverterer instans dataen og genererer en onologiudvidelse der, sammen med den eksisterende ontologi, beskriver de konverterede kuber.

For at teste vores patterns konverterer vi det velkendte TPC-H dataset til RDF og annoterer det med QB4OLAP. Vi definerer kuber i dette dataset og modellerer dem som Snowflake-pattern i en række forskellige størrelser. Disse kuber konverteres til Star-pattern og Denormalized-pattern og der køres queries på all tre versioner of kuberne, hvorefter køretiderne sammenlignes. Vi ser at Star-pattern klarer sig generelt bedst, mens Denormalized-pattern ikke ser ud til at have nogle klare fordele i nogle af vores kuber. Pladsforbruget for Star-pattern er marginalt højere end for Snowflake-pattern, men Star-pattern kan både håndtere flere af vores test queries uden at time out og generelt evaluere queries hurtigere.

Modeling Linkable Multidimensional Cubes with Logical Patterns

Alex Bondo Andersen
Aalborg University
Denmark
aban09@student.aau.dk

Kim Ahlstrøm Jakobsen
Aalborg University
Denmark
kjakob09@student.aau.dk

ABSTRACT

Traditional Decision Support Systems (DSS) rely on local data to answer business questions. This limits the decision maker to a simplified world. Ad-hoc data integration enables the decision maker to answer more business questions because the local data is enriched with situationally appropriate data. Linked data in Resource Description Framework (RDF) format is attractive as situational data because of its growing amount and its interconnectivity between sources. Several systems have been proposed to facilitate ad-hoc data integration and there is a high potential for further research. We explore the potential of storing the local data in RDF – as well as the situational data. We show how business questions can be answered on RDF formatted cubes annotated with the QB4OLAP vocabulary, called Linkable Multidimensional (LMD) cubes. We introduce three logical patterns for LMD cubes, which have different characteristics with regards to load time, storage size, and query evaluation time. Our novel algorithm Semantic Web OLAP Denormalizer (SWOD) is used to convert cubes of one pattern to another. The well-known TPC Benchmark^{TM}H (TPC-H) dataset is converted to RDF and described as LMD cubes. We evaluate our patterns on these cubes, with queries based on the TPC-H business questions.

1. INTRODUCTION

When an organization is operating in a volatile and competitive environment, fast decision making is essential. Decision Support Systems (DSS) are used to help the decision makers make these decisions based on data stored in the organizations' Data Warehouse (DW). To make well-informed decisions, the DW must contain data relevant to the decisions. Data from different sources may be integrated with the organization's operational data to construct the DW. Traditionally this integration is performed through an Extract Transform Load (ETL) process. The creation of an ETL process is often slow and expensive, and it is not possible to accommodate for all future decisions because the environment may change or new sources become available. In the latest years an increasing amount of systems [11, 1, 32, 29] have been designed with focus on enabling ad-hoc data integration, which enables data to be integrated at decision time. A common denominator is that they use the Resource Description Framework (RDF) [37] to facilitate ad-hoc data integration. This opens up for a large repository of data, which can be integrated with a local DW ad-hoc when an unforeseen decision arises, instead of having to alter the entire ETL process.

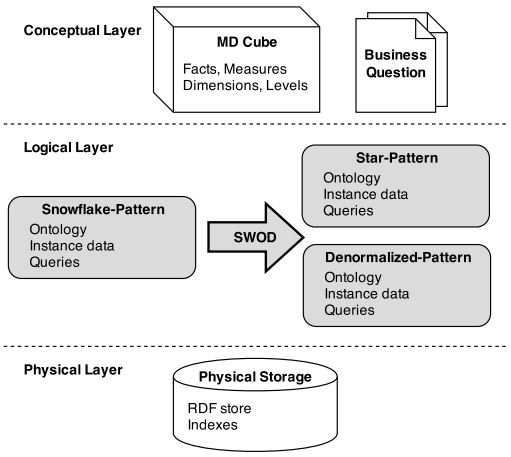


Figure 1: Illustration of our main contributions.

Other systems store Multidimensional (MD) cubes of the local DW in a traditional format (relational or multidimensional databases) and extending the query system to accommodate for ad-hoc data integration. We use RDF databases to store the local MD cubes instead. This allows us to query the local MD cube with SPARQL, which has native support for ad-hoc data integration through query federation. We use the term *Linkable Multidimensional (LMD) cube* for an MD cube in RDF format. To describe LMD cubes we use ontologies which are based on the well-known vocabulary QB [17], extended with QB4OLAP [22]. We present three patterns for modeling the LMD cubes at the logical layer, which prove to have different characteristics with regard to load time, storage size, and query evaluation time. These patterns are based on the snowflake schema, star schema, and fully denormalized schema known from the relational OLAP community. Because there are no strict schemas for

Orders		
Nation	*Customer*	*Order*
France	Andre	130
		143
	Charlotte	112
Germany	Ben	141
	Paul	142

Partsupplier			
Part			
Seashell	Firebrick	Chocolate	Aquamarine
—	—	$150	$350
—	$50	—	—
—	—	$75	$300
$100	—	—	—
—	$50	—	$350

Figure 2: Constructed extract of the Lineitem-cube showing the price of parts on certain orders.

defining resources in RDF at the logical layer, as we know from the relational world, we introduce the concept logical patterns, or simply *patterns*, rather than using schemas. An LMD cube can be logically modeled as either Snowflake-pattern, Star-pattern, or Denormalized-pattern. In the context of the Semantic Web no one has, to the best of our knowledge, previously described how to apply these schema techniques.

We create a novel algorithm, which converts an LMD cube in Snowflake-pattern to an LMD cube in either Star-pattern or Denormalized-pattern. We call this algorithm Semantic Web OLAP Denormalizer (SWOD). The algorithm can handle multiple dimension hierarchies as well as unbalanced hierarchies.

In Figure 1 we see the three database model layers and the terms we use in each layer. At the conceptual layer there is a set of business questions used as part of a decision making process, which the MD cube is designed to answer. At the logical layer we operate with our three patterns, which are used to model an MD cube as an LMD cube. SWOD is used to convert an LMD cube from Snowflake-pattern to either Star-pattern or Denormalized-pattern. Each pattern has its own ontology, instance data, and queries. An ontology contains the metadata of an LMD cube and describes the instance data. Both ontologies and instance data are in RDF format and annotated with QB4OLAP. Queries are used to answer business questions and are written in SPARQL. LMD cubes are stored physically in an RDF store. The marked concepts are our main contributions.

TPC Benchmark^{TM}H (TPC-H) is a decision support benchmark with business questions and a dataset focusing on sales. We convert the TPC-H dataset to RDF and create LMD cubes. We define three LMD cubes: Lineitem-cube, Orders-cube, and Partsupplier-cube. Collectively, these cubes constitute the dataset Linkable TPC-H (LTPC-H). From the business questions, SPARQL queries are constructed. Figure 2 contains a constructed extract from the Lineitem-cube. It shows the orders of two French and two German customers. Their orders contain one or several purchases of parts with the price given in dollars. We use this extract in our examples throughout this paper.

In order to show how ad-hoc data integration is performed we introduce the business question: "Show the total revenue per nation for nations, which have at least a population of thirty million". The population of a nation is not a part of the TPC-H dataset but is available at the DBpedia endpoint[1].

Our novel contributions are: (i) Definitions of three patterns for modeling LMD cubes; (ii) Design and implementation of the algorithm SWOD, which transforms cubes in Snowflake-pattern into Star-pattern and Denormalized-pattern; (iii) Annotation of the TPC-H dataset and queries with QB4OLAP.

The structure of the remaining paper is as follows. Next we introduce preliminaries where we provide necessary background information for RDF, MD cubes, and MD vocabularies in Section 2. We describe related work in Section 3 before we describe how we introduce TPC-H into a Semantic Web context in Section 4. We explain the logical patterns in Section 5. In Section 6 we explain the SWOD algorithm and its auxiliary functions. We describe the evaluation setup in Section 7 and in Section 8 we present the actual evaluation. Last in Section 9 we have the conclusion and future work.

2. PRELIMINARIES

We present a basic understanding of the main topics used in this paper, namely definitions of an RDF graph, a basic graph pattern, a multidimensional cube, and the vocabulary we are using to describe our test data, namely QB4OLAP [22]. For a more intuitive and elaborate description of triples and querying using basic graph patterns see Appendix A.

2.1 RDF Graphs

An RDF graph, or simply a graph, consists of a set of *resources* linked together with labeled directed edges called *predicates* [18]. A resource can represent anything, including physical things. A triple consists of three elements, a *subject*, a predicate, and an *object*. In a triple, the subject is the resource located at the base of the predicate arrow and the object at the tip. In Figure 3 `Ben` is the subject, `lives_in` is the predicate, and `Germany` is the object. A formal definition is given in Definition 1 [34]. A resource, which can be used as a predicate is called a *property*. A resource can be an instance of a *class*. A class is also a resource itself and is used to describe the meaning of its instances and how they can be related to other instances. A triple indicates that there is some relationship between the subject and the object. The nature of the relationship is given by the property used as predicate.

[1] `http://dbpedia.org/sparql`

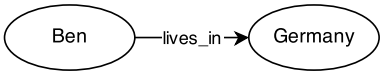


Figure 3: An example triple.

Subjects can be *IRI*s or *blank nodes*. Internationalized Resource Identifiers (IRIs) are used to uniquely identify a resource and a blank node is a resource without a globally available identifier, therefore also called anonymous resources. Objects can be *literals*, IRIs, or blank nodes. Literals are of basic types, such as strings, integers, and dates. Predicates are IRIs.

Definition 1 (Triple, Graph)**.** Given a set of IRIs U, a set of blank nodes B, and set of literals L, a triple t is defined as $t = (s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$. A graph G is a set of triples: $G \subseteq (U \cup B) \times U \times (U \cup B \cup L)$.

In some cases a triple is present in a *context*, which essentially makes it a *quad* instead of a triple. Contexts can be used to group sub-graphs together by adding a context to the triples of a sub-graph. Such sub-graphs are called *named graphs*, where the context represent the name. This allows for a query on a graph to specify the named graph which is of interest and only consider triples from it when answering.

To make IRIs in triples more readable, prefixes for *namespaces* can be defined. These prefixes are used as a shorthand notation for frequently used IRIs. In Figure 4 the table contains the namespaces we use throughout this paper. A prefix is used by appending it with a *local name*, which is then automatically appended to the IRI of the prefix. E.g. `rdfs:Class` translates to `<http://www.w3.org/2000/01/rdf-schema#Class>`.

2.2 Basic Graph Patterns

A term related to the triple is the Basic Triple Pattern (BTP), which may contain variables at any position (subject, predicate, and/or object). Variables in a BTP are prefixed with a "?" to identify them. A set of BTPs is called a Basic Graph Pattern (BGP). BGPs are the basic building blocks of queries over a graph. A formal definition of a BTP and a BGP is given in Definition 2 [34].

Definition 2 (Basic Triple Pattern, Basic Graph Pattern)**.** Given a set of IRIs U, a set of blank nodes B, a set of literals L, and a set of variables V, a BTP btp is defined as $btp = (s, p, o) \in (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$. A BGP bgp is a set of BTPs: $bgp \subseteq (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$. When two or more BTPs share a variable, we say that there is a *join* between these BTPs and the shared variable is the *joining variable*.

When *evaluating* a BGP $\{(s_1, p_1, o_1), (s_2, p_2, o_2), \cdots, (s_n, p_n, o_n)\}$ over a graph G we use the following notation:

$$G\,(s_1\ p_1\ o_1\ .\ s_2\ p_2\ o_2\ .\ \cdots\ .\ s_n\ p_n\ o_n)$$

The result is a set of tuples containing bindings for the variables in the BGP. The bindings are often represented as a table, with the variables as columns and each row representing a single binding. Note that a joining variable has only a single column (not one for each BTP with the variable), thereby creating a join between the BTPs with the joining variable.

In practice we use the query language SPARQL 1.1 [39], which uses BGPs as the basic building block for querying a graph dataset. It has additional functionality as well, such as filtering, aggregation, and query federation. Querying is further described in Appendix A.

2.3 Multidimensional Cubes

An MD cube is often used when performing OLAP operations. We use a definition of MD cubes based on the one given by Jensen et al. [25]. An MD cube is used to capture a number of *facts* in a set of *dimensions*. Intuitively the facts are located inside the cube while the dimensions are located along the axes. This would suggest a fixed number of three dimensions, however, any positive number of dimensions is possible. Every dimension has a number of *hierarchies*, which contain a number of *levels*. The levels define different granularities of the dimension. The finest granularity is called the *lowest level*. The coarsest granularity is called the *top level*. A classic example of a dimension is the date dimension, which might have a hierarchy with the levels date, month, and year (and implicitly a top level called "all dates"). Note that the lowest level has the same name as the dimension itself, which is often the case. Every level may have a number of *attributes*. A month could have an attribute indicating the season and a date could have an attribute indicating whether or not it is a weekend.

A fact is related to an instance of each dimension at a given level – initially the lowest level. Furthermore, a fact, usually, has a number of measures. A measure is a numeric value and an *aggregation formula*. A variant of the measure is the *derived measure*, which consists of several numeric values. E.g. a sale may have revenue and expenses as regular measures, and a profit ($revenue - expenses$) as a derived measure.

A cube can be *rolled up* along a dimension hierarchy to a coarser level. When rolling up, a new set of facts is constructed from the finer grained facts, by aggregating the measures and associating the facts with the coarser dimension levels. A rolled-up cube is similar to an ordinary cube in that it can be further rolled up along the same or other dimension hierarchies, but it can also be *drilled down* along a rolled-up dimension hierarchy, which is the reverse of rolling up. In short, rolling up grants overview, while drilling down grants details.

To cope with the details of a large, fine grained cube it is possible to *slice* it. A slice is a selection on a dimension of a specific subset of the cube, e.g. dates, which are weekends. When slicing a cube the output is a new cube, which can be further sliced (often called *diced*) along the same or other dimensions, and rolled up. Note the difference between rolling up and slicing; when rolling up we combine facts and their measures to get a new cube, when slicing we remove facts to get a new cube.

The snowflake and star schema are the two most commonly used models when structuring MD cubes in a relational DW

Prefix	IRI
ltpch:	http://extbi.lab.aau.dk/ontology/ltpch/
ltpch-inst:	http://extbi.lab.aau.dk/resource/ltpch/
qb:	http://purl.org/linked-data/cube#
dbpprop:	http://dbpedia.org/property/
dbponto:	http://dbpedia.org/ontology/
qb4o:	http://publishing-multidimensional-data.googlecode.com/git/index.html#ref_qbplus_
func:	http://example.org/customfunction/
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
owl:	http://www.w3.org/2002/07/owl#
xsd:	http://www.w3.org/2001/XMLSchema#

Figure 4: RDF prefixes and their IRIs.

with the intention of performing Online Analytical Processing (OLAP) on them. Depending on the data the appropriate schema is chosen. A snowflake schema stores a dimension in several tables where each table contains a level [28, pp. 55-57][25, p. 14]. A completely snowflaked schema is in third normal form [28, p. 55], such that data redundancy is kept at a minimum and also keeping the logical model very similar to the conceptual model. In a star schema the dimensions are denormalized, which results in data redundancy [28, p. 55][2, pp. 9-24][25, pp. 12-13]. When querying a star schema, the number of joins between tables is usually lower, because every dimension level is already captured in single table. This generally leads to better query evaluation time, but at the cost of a larger storage space.

A third schema is the (fully) denormalized schema. Costa et al. [16] explores using a denormalized schema such that all queries are performed only on a single table. This results in an increase in storage use, but makes it possible to calculate query evaluation time very accurately. They also see improvements in query evaluation time as the data storage size increases compared to a snowflake schema. For most cases, however, the denormalized schema is outperformed by the snowflake schema. IBM considered using a denormalized schema for their Blink project [5], but refrained from this because the amount of redundant data simply became larger than the gain from having a single table.

We present three patterns for modeling LMD cubes at a logical level, these are inspired by the three schemas from the relational world. We call these Snowflake-pattern, Star-pattern, and Denormalized-pattern.

2.4 Multidimensional Vocabularies

In a Semantic Web context there are published a number of vocabularies and ontologies in different areas. With regards to MD data, there are two main vocabularies: QB and Open Cube vocabulary [21] (OC), along with a combination of the two called QB4OLAP [22]. QB is a W3C recommendation while OC is suggested by Etcheverry and Vaisman. The underlying model of QB is compatible with model of SDMX [19], which is a standard for modeling statistical data. This is because QB is built with the purpose of publishing statistical data. However, it is usable for other applications of MD data, such as OLAP cubes. OC, on the other hand, was built with OLAP cubes in mind, which means that it is more specialized towards it. OC allows for defining relations between dimension levels at both the ontology and instance level, while QB only allows this at the instance level. In effect, this means that a consumer of an OLAP cube described with QB cannot easily get an overview of the dimension levels, compared to an OLAP cube described with OC. OC also has a notion of aggregate functions, which may be related to different measures, such that one measure can have one (or more) aggregate function associated with it. QB is the de facto standard of these two vocabularies for describing MD data despite its shortcomings when it comes to OLAP specific subjects. The third vocabulary, QB4OLAP [22], has been created by the authors of OC to be an extension of QB, which is more suited for OLAP cubes. It adds definitions for relationships between dimension levels as part of the ontology and aggregate functions for measures. QB4OLAP is, however, extended from an older version of QB namely the draft from April 2012[2]. All three vocabularies described above have a Snowflake-pattern approach to OLAP. We show how QB4OLAP can be applied to MD cubes in Star-pattern and Denormalized-pattern.

We here give a brief description of the QB4OLAP vocabulary. A full description of QB4OLAP can be found in [22]. Figure 5 shows classes and properties, which make up the vocabulary. It is used to annotate the ontology describing an MD cube. The white background indicates that a class or property is taken from the QB vocabulary, while a gray background indicates additions made in QB4OLAP. Finally, ghosted classes and properties are belonging to external vocabularies, e.g. `skos:Concept`, which belongs to the Simple Knowledge Organization System (SKOS) vocabulary[3]. To describe a cube with QB4OLAP a resource of the type `qb:DataStructureDefinition` must be created. This resource is connected to a number of `qb:ComponentSpecification`s, each of which define a level, dimension, measure, or attribute. A `qb:ComponentSpecification` links to a property of the type `qb:ComponentProperty`. `qb:ComponentProperty` has five subclasses: `qb:LevelProperty`, `qb:DimensionProperty`, `qb:MeasureProperty`, `qb:AttributeProperty`, and `qb:CodedProperty`. The last one is not relevant in this paper. Additional properties can be used to further specify a `qb:ComponentSpecification`, depending on

[2] `http://www.w3.org/TR/2012/WD-vocab-data-cube-20120405/`
[3] `http://www.w3.org/2004/02/skos/`

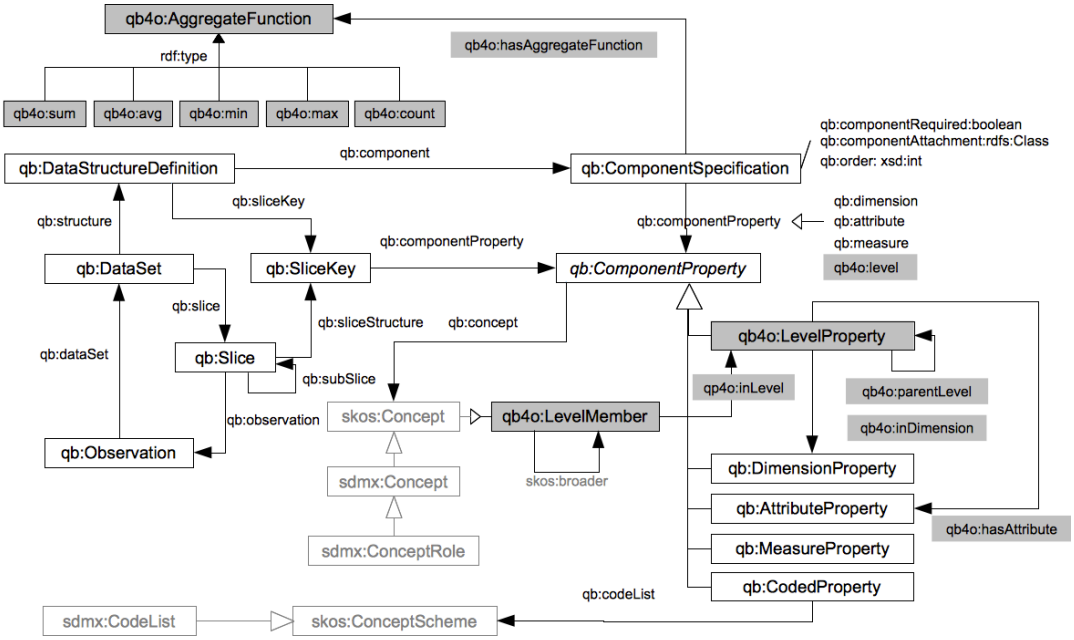


Figure 5: The QB4OLAP vocabulary. From Etcheverry and Vaisman [22].

the subtype of `qb:ComponentProperty` used. E.g. when defining a `qb:MeasureProperty`, the allowed aggregation function(s) can be specified with the property `qb4o:hasAggregateFunction`, which points to an instance of the `qb4o:AggregateFunction` such as `qb4o:sum`. When a complete `qb:DataStructureDefinition` is defined with a number of `qb:ComponentSpecification`s, an instance of type `qb:DataSet` can be defined, by giving it a name (through `rdf:label`) and linking it to the `qb:DataStructureDefinition`. A fact is defined as an instance of `qb:Observation` and points to an instance of `qb:DataSet`. An example of QB4OLAP in use is given in section 4.

3. RELATED WORK

The usage of RDF data to enable ad-hoc data integration has lately started receiving some attention. Abelló et al. [1] propose a high level architecture of a system where stationary data and situational data is combined to make *Fusion Cubes*. The stationary data is in the form of a snowflake schema or a star schema and is augmented by the situational data. The situational data is preferably in RDF format and is used as a mean to add semi-structured and unstructured data which has narrow focus and/or short life span – hence *situational* data. Romero and Abelló [32] present a concrete architecture, which is inspired by Abelló et al. [1]. Both these papers work toward a DSS with ad-hoc data integration based on the RDF standard, while using a "traditional" DW to store the stationary data. We store the stationary data in RDF format as well, thus relying on the ad-hoc query federation allowed through SPARQL to enable ad-hoc data integration.

Prior to SPARQL 1.1 there was no support for aggregation in SPARQL, which meant OLAP queries were not a real possibility. The team at Virtuoso made an extension to SPARQL in 2009 to enable OLAP queries [20]. This extension is called SPARQL-BI. To test the extension, relational TPC-H data was mapped to RDF through Virtuoso. The ontology for this data was generated in an ad-hoc manner, such that every attribute in the relational schema becomes a predicate and every table becomes a class. TPC-H queries were constructed in SPARQL-BI and run on the RDF views.

The queries made in SPARQL-BI were picked up by team from Berlin SPARQL Benchmark [12] (BSBM), who released a benchmark tool using TPC-H data in the form of RDF views as the base data. They also developed a series of support tools, one of these tools are csv2ttl which we utilize. Along with the queries in SPARQL-BI, queries where also rewritten to Virtuoso SPARQL 1.1, which had been released in the meantime. The queries were, however, following Virtuoso SPARQL rather than the official SPARQL standard, which means that they do not run on all types of RDF databases. Also, not all queries are correct with regards to the TPC-H specification, e.g. missing ordering of results. We translate the TPC-H dataset to RDF, describe it with the known vocabulary QB and its extension QB4OLAP, and perform OLAP queries on it using standard SPARQL 1.1.

Kämpgen and Harth [26] model the Star Schema Benchmark [30] (SSB) – a benchmark dataset similar to TPC-H [33] – in two ways: Using the vocabulary QB and an ad-hoc ontology created from the relational schema of SSB. Queries evaluated on the data modeled with QB generally perform worse than on the data modeled from the relational schema. We present the well-known TPC-H dataset modeled as Snowflake-pattern, Star-pattern, and Denormalized-pattern described with the QB4OLAP vocabulary. Furthermore, we create an automatic way of converting an LMD

cube described by QB4OLAP from a Snowflake-pattern into Star-pattern and Denormalized-pattern, and evaluate these.

Kämpgen and Harth [27] propose an open source OLAP engine that enables users to pose MDX and metadata queries on Semantic Web sources. They support data annotated with the QB vocabulary but do not consider QB4OLAP. We use the QB4OLAP vocabulary, which makes it possible to annotate which aggregate functions the different measures support. We use SPARQL as our query language as oppose to MDX, to have native support for ad-hoc data integration by use of the `SERVICE` keyword.

4. TPC-H IN THE SEMANTIC WEB

TPC-H [33] is a decision support benchmark used to compare database system vendors in terms of queries per hour. This benchmark contains a data generator, 22 queries, ACID tests, and a performance matrix. The data generator generates the data in a serialized database format, which can be loaded into a relational database. The schema for the generated relational data is seen in Figure 6. The figure shows the concepts encapsulated in tables and foreign keys are indicated with arrows between the tables. When generating a TPC-H dataset a *scaling factor* is selected, to define the size. The queries are designed to represent complex business questions. The queries contain a number of substitution parameter, which need to be instantiated before running them. This make it possible to run a query several times by changing the instantiations of these substitution parameter and thus avoiding an unfair amount of cache hits. TPC-H tests for ACID properties in the form of transactions, but since we focus on data integration we deem this out of scope. The TPC-H performance matrix is based on results from running both queries and ACID tests in parallel, which means we will not be using it. We use the raw query evaluation time as evaluation measure. To use TPC-H data in a Semantic Web context we convert it into RDF format. We use an existing vocabulary, based on the de facto standard vocabulary for cubes, to describe the data after conversion, namely QB4OLAP [22]. The TPC-H dataset converted to RDF and described with this vocabulary is called LTPC-H.

The Lineitem-cube has two dimensions, Partsupplier and Orders. The Partsupplier dimension has the levels: Partsupplier, Part, Supplier, Nation, and Region. These levels are divided into two hierarchies, one for Part and one for Supplier, where the Nation and Region are in the Supplier hierarchy. The Orders dimension has the levels: Orders, Customer, Nation, and Region. All these levels are in the same hierarchy. The dimension levels and their order are seen in Figure 7. The Nation and Region levels are present in both the Orders and Partsupplier dimension. We use only one set of instances of Nations and Regions and refer to these in both dimensions. We do this to avoid data duplication and to enable queries which link between the dimensions at these levels. The measures are Price, Discount percentage, Tax, and Quantity. Additionally there is a derived measure called Revenue, which is calculated as $price \cdot (1 - discount)$.

In Figure 2 we see a constructed example of the Lineitem-cube, where the Part hierarchy of the Partsupplier dimension is rolled up to Part and the Supplier hierarchy is completely rolled up. The Orders dimension hierarchy is not

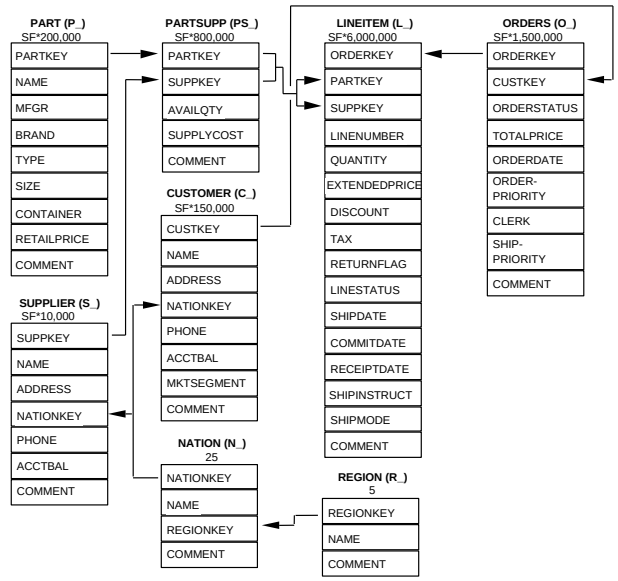


Figure 6: The TPCH relational schema. Figure originates from [33].

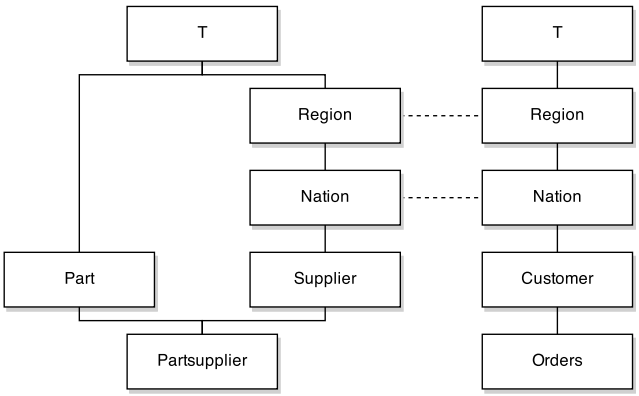
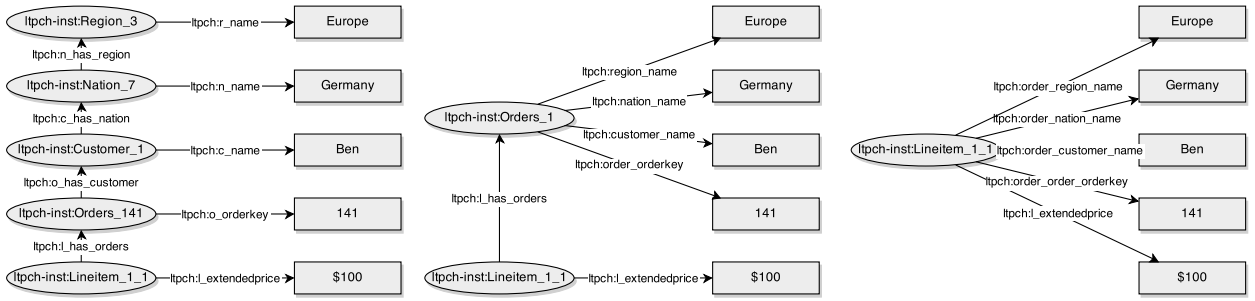


Figure 7: The dimensions Orders and Partsupplier from the Lineitem-cube.

rolled up, instead we have made a slice on the Region level of the Orders dimension which selects the Region Europe. Only the Price measure is shown in this example.

We use the TPC-H data generator DBGEN[4] to generate TBL files, which is a format for serialized database tables. The TBL files contain rows, delimited by a new-line character, which are divide into a number of columns, delimited by a pipe "|". This data is converted into RDF using the BIBM conversion tool[5]. The conversion is based on a schema file which defines how to convert each TBL file into RDF[6]. For each row in the TBL file being converted, a URI is constructed based on information in the schema file. For each

[4] `http://www.tpc.org/tpch/spec/tpch_2_16_1.zip`
[5] `http://sourceforge.net/p/bibm/code/HEAD/tree/trunk/bibm/src/com/csvreader/`
[6] `http://sourceforge.net/p/bibm/code/HEAD/tree/trunk/bibm/tpch/virtuoso/tpch_schema.json`



(a) Snowflake-pattern example. **(b) Star-pattern example.** **(c) Denormalized-pattern example.**

Figure 8: The figures show the fact and Orders dimension describing Ben's purchase in Figure 2 modeled with our three patterns.

column in every row, a triple is constructed with the rows URI as subject, the column value as the object and the predicate defined in the schema file. Then we decorate the RDF TPC-H dataset with the QB4OLAP vocabulary and create three cubes. The cubes have Lineitem, Orders, and Partsupplier as facts and are named accordingly. These cubes and their ontologies comprise LTPC-H. Figure 8a shows an instance of a Lineitem fact with level instances of the Orders dimension relating to the purchase by the Customer Ben in Figure 2.

In Code Snippet 1 the `qb:DataStructureDefinition` and the `qb:DataSet` definitions for the Lineitem-cube are seen. The `qb:ComponentSpecification`s are blank nodes and typed implicitly by being object of a triple with `qb:component` as predicate. The cube has two dimensions and the lowest levels of each dimension are shown by the predicates `ltpch:l_has_order` and `ltpch:l_has_partsupplier` respectively. The Lineitem-cube has four measures, which are `ltpch:l_quantity`, `ltpch:l_extendedprice`, `ltpch:l_discount`, and `ltpch:l_tax`, each has an aggregation type associated with it. Finally a cube, `ltpch:lineitemCube`, is defined with the `qb:structure` predicate pointing to the previously defined `ltpch:lineitemStructure`.

We define cubes in a similar manner where Orders and Partsuppliers are facts. These cubes, the dimensions, and the levels are found in Appendix D. The Partsupplier-cube has two dimensions, Part and Supplier. The Part dimension has only a single level: Part. The Supplier has three levels: Supplier, Nation, and Region. The Orders-cube only has a single dimension, the Customer dimension, which has the levels: Customer, Nation, and Region.

We have made SPARQL queries corresponding to the 22 business questions from TPC-H and split them among the three cubes, based on which resources the queries require. We based these queries on the BIBM queries[7]. However, we have made three kinds of changes: *Standardization changes*, *correction changes*, and *optimization changes*. Standardization changes are applied to conform to the SPARQL standard [39]. Some queries from BIBM do not conform to the TPC-H specification. To these queries we make correction changes to make them conform. Optimization changes are reordering of triple patterns based on knowledge of how the data is stored. All the queries are in Appendix G. Below we present the concrete changes we have made.

Standardization Changes. Standardization changes are applied to all queries because the queries provided by BIBM were in Virtuoso SPARQL. We changed them to follow the official W3C recommendation[8], such that they can be run on e.g. Sesame. These changes are syntactic in nature. We change the binary operator `and` to `&&` and replace the `like`-operator with the `REGEX` function. Further we remove commas between variables and encapsulate aggregation functions with parentheses in the `SELECT` clause.

Correction Changes. We made the necessary corrections such that the queries correspond to the TPC-H specification. All queries, which use the brand attribute for slicing of the Part level, have been modified to do the match case-insensitively, due to a mismatch between case convention in the data and the queries. In query 7 we add `?cust_nation` in the order-by clause. In query 15 and 20 we introduce a substitution parameter in the date selection, to conform to the TPC-H specification.

Optimization Changes. To have a broader foundation for comparing our patterns, we optimize some of the queries, such that they can be evaluated in reasonable time. These changes are called optimization changes and are, like standardization changes, on syntactic changes. We removed excess triple patterns such as `ltpch:customer_ben a ltpch:customer` when the type could be implicitly inferred from other properties used on the resource, `ltpch:customer_ben` in this example. In query 2 the filter on Region name is moved into a sub-query, to make the result set from the sub-query smaller without interfering with the semantics of the query. In query 4 we have introduced a sub-query instead of having a filter-exists, which is slower when using Sesame without object indexes.

[7] `http://sourceforge.net/p/bibm/code/HEAD/tree/trunk/bibm/tpch/sparql/`

[8] `http://www.w3.org/TR/sparql11-query/`

```
ltpch:lineitemStructure a qb:DataStructureDefinition ;
  qb:component [ qb4o:level ltpch:l_has_order ; qb:order 1 ] ;
  qb:component [ qb4o:level ltpch:l_has_partsupplier ; qb:order 2 ] ;
  qb:component [ qb:measure ltpch:l_quantity ; qb:hasAggregateFunction qb4o:sum ] ;
  qb:component [ qb:measure ltpch:l_extendedprice ;
                 qb4o:hasAggregateFunction qb4o:sum ] ;
  qb:component [ qb:measure ltpch:l_discount ; qb4o:hasAggregateFunction qb4o:avg ] ;
  qb:component [ qb:measure ltpch:l_tax ; qb4o:hasAggregateFunction qb4o:avg ] .

ltpch:lineitemCube a qb:DataSet ;
  qb:structure ltpch:lineitemStructure ;
  rdf:label "Lineitem Cube" .
```

Code Snippet 1: Part of ontology defining the data structure for the Lineitem-cube.

5. LOGICAL PATTERNS

Data stored in Snowflake-pattern adheres to the Linked Data (LD) principles made by Tim-Berners Lee (presented in Appendix A.2), since every "thing" is represented by a URI (a specialization of IRIs), which happens to be a Hyper Text Transfer Protocol (HTTP) URL. These URIs can be looked up and data about the thing is shown. This is good for publishing data, because it allows other people to use it and gain an understanding of it quite easily. However, our focus is on the ability to perform OLAP queries on the data, rather than the data being published in a nice format. Other work (Kämpgen and Harth [26]) suggests that having dimension level instances in a denormalized format improve performance (for most queries of the SSB dataset). Therefore, we have developed two additional patterns for modeling LMD cubes, namely Star-pattern and Denormalized-pattern.

LMD cubes in Star-pattern and Denormalized-pattern are not linked data, since a URI may actual represent several things in these patterns. A dimension instance of the Orders dimension modeled as Star-pattern represents both an Order, a Customer, a Nation, and a Region. However, since they are in RDF format, they can still be linked to other sources which are in RDF.

Below we present the details for our three patterns and relate them to each other.

5.1 Snowflake-pattern

A dimension modeled as Snowflake-pattern has a number of level instances. Facts links a to level instance of the lowest level of a Snowflake-pattern dimension. Such a level instance may have a number of attribute values which are reached through attribute properties, defined in the ontology. Furthermore, level instances may refer to a level instance of the parent level, called the parent level instance. In Figure 8a we see the fact representing Ben's purchase, `ltpch-inst:Lineitem_1_1`, linking to a level instance of the Orders level, `ltpch-inst:Orders_141`, which again links to Customer level instance, `ltpch-inst:Customer_1`, etc. Every level instance in this example has a single attribute, which describe each level instance. In practice more attributes are often required. The Snowflake-pattern is similar to the snowflake schema in a relational database. Here we use the term Snowflake-pattern rather than snowflake schema since we are working with graphs rather than tables and therefore find it more appropriate to say that a dimension is of a certain pattern instead of a certain schema. An LMD cube with all dimensions modeled as Snowflake-pattern is said to be in Snowflake-pattern. Modeling an LMD cube as Snowflake-pattern ensures minimum data redundancy of the three patterns and keeps the cube as linked data.

5.2 Star-pattern

An LMD cube in Star-pattern does not have level instances. Instead every dimension has a number of dimension instances. A dimension instance is a resource, which has all attributes of all levels in the given dimension, similar to a dimension modeled as a star schema in a relational database. In Figure 8b we show the Orders dimension and fact relating to Ben's purchase modeled as Star-pattern. In the figure, the level instances `ltpch-inst:Region_3`, `ltpch-inst:Nation_7`, and `ltpch-inst:Customer_1` are merged into the bottom level instance, `ltpch-inst:Orders_141`. The merge involve changing triples relating to all none-bottom level instances such that they relate to the bottom level instance instead – e.g. the triple with the predicate `ltpch:c_name` is changed such that it has `ltpch-inst:Orders_1` as subject, we also change the predicate, which we explain further below. After the merge we say that `ltpch-inst:Orders_1` is a dimension instance, rather than a level instance, since it represents an instance of the Orders dimension rather than just a level. When all dimensions of an LMD cube are in Star-pattern we say the cube is in Star-pattern.

As Figure 8a and Figure 8b show, we are introducing new properties in the Star-pattern. We do this to avoid name collisions of properties between levels when merging level instances. Although this is not a risk in LTPC-H, because properties on every level are already prefixed with a letter unique for that level, it may be a problem for other datasets. If LTPC-H did not have these letter prefixes, the introduction of new properties would ensure that we could distinguish between the name of the Nation and the name of the Customer. These new properties are defined as sub-properties of the properties on which they are based, e.g. `ltpch:customer_name ltpch:subPropertyOf ltpch:c_name`, in the ontology of the LMD cube. We give an algorithm for converting an LMD cube from Snowflake-pattern to Star-pattern in Section 6.

5.3 Denormalized-pattern

In the Denormalized-pattern the dimensions do not have any instances. All level attributes are moved to be attached directly at the facts. This means that there will be a lot of data duplication, but the dimension attributes will be located closer to the facts. In Figure 8c we see Ben's purchase modeled as Denormalized-pattern. We see that there are now only the fact, measure, and attribute values, which are attached directly to the fact. Like the Star-pattern we have to avoid predicate collisions, both between levels but also between dimensions – both the Partsupplier and the Orders dimension have the Nation and Region levels. We use a similar property naming convention as in the Star-pattern, but additionally we prefix the new properties with the name of the dimension. We also define the new properties as sub-properties of the properties which they are based on, e.g. `ltpch:orders_customer_name ltpch:subPropertyOf ltpch:c_name`, in the ontology of the LMD cube. When all dimensions of a cube are in a Denormalized-pattern we say the cube is in a Denormalized-pattern.

5.4 Unbalanced Hierarchies

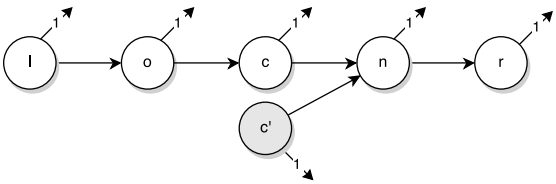
Dimension hierarchies may be unbalanced [25, p. 47], e.g. not every Customer in LTPC-H has placed Orders and in effect is not related to any orders. In Figure 9a we see an abstract example of an unbalanced hierarchy in the LTPC-H dataset in Snowflake-pattern. The circles represent resource instances where "l" is a Lineitem, "o" is an Order, "c" and "c′" are Customers, "n" is a Nation, and "r" is a Region. Each resource instance has an arrow labeled with a number, this represent the number of attribute sets that is connected to the resource. An attribute set is the set of attributes of a level instance. The Customer, c′, does not have any Orders, however this Customer should still be included in the Lineitem-cube. Since c′ cannot be merged into any orders, we create a dimension instance based on c′ instead. This means that every level instance above c′ is merged into c′. This is illustrated in Figure 9b where c′ now has the attribute set of its Nation and Region i.e. three. In the Denormalized-pattern we use the same strategy as can be seen in Figure 9c.

5.5 New Properties

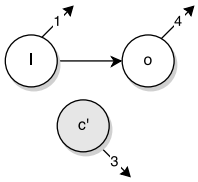
When making new properties and declaring them as sub-properties, in both Star-pattern and Denormalized-pattern, we have to be careful. When a property `prop_b` is a sub-property of `prop_a`, every triple `res_1 prop_b res_2` implies the triple `res_1 prop_a res_2`[9]. This is generally our intention. However, for the property `owl:sameAs` and for properties declared as `owl:InverseFunctionalProperty`, this has unintentional side effects.

When there is an `owl:sameAs` predicate between two resources, it means that these are *exactly* the same. There might be an `owl:sameAs` from a level instance to an external resource. When this level instance is merged into the lowest level to create a dimension instance, we do not wish define the dimension instance as being the exact same resource as the external resource. To avoid this we do not declare new properties based on the `owl:sameAs` property as sub-properties of `owl:sameAs`.

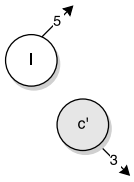
[9] `http://www.w3.org/TR/rdf-schema/#def-subproperty`



(a) Snowflake-pattern.



(b) Star-pattern.



(c) Denormalized-pattern.

Figure 9: The Customer c′ is an unbalanced level instance. The figures show how this instance is represented in the three patterns.

When a property `prop_a` is declared as an instance of `owl:InverseFunctionalProperty`, it means that the object of a triple with `prop_a` as the predicate, uniquely defines the subject resource of the triple[10]. If two presumed different resources point to the same object through `prop_a`, they are actually *the same resource*. Inverse functional properties are similar to primary keys in the relational world. In LTPC-H we have declared a number of properties as `owl:InverseFunctionalProperty`, e.g. `ltpch:c_custkey` on the Customer level. When declaring a new property based on a property of the type `owl:InverseFunctionalProperty`, we do not declare the new property as a sub-property of the old. This ensures that we avoid situations like when two Orders dimension instances in Star-pattern would be considered the same resource, because they have the same `ltpch:customer_custkey` value.

6. SEMANTIC WEB OLAP DENORMALIZER

To convert LMD cubes from Snowflake-pattern to either Star-pattern or Denormalized-pattern we use the SWOD algorithm, which we present in this section along with auxiliary algorithms. We perform the conversion by analyzing the QB4OLAP based ontology describing the cubes. Our approach is to merge the level instances into dimension instances to produce Star-pattern cubes or merge level instances with the facts to produce Denormalized-pattern cubes. SWOD can run in either SWOD-S mode or SWOD-D mode, depending on whether the output pattern is Star-pattern or Denormalized-pattern. We describe the algorithms in general terms with references to LTPC-H instance data and ontology, the latter of which is found in Appendix D. We recommend using the ontology to gain a better understanding of the algorithms.

Semantic Web OLAP Denormalizer (SWOD) is described in Algorithm 1. The inputs to this algorithm are a collection

[10] `http://www.w3.org/TR/owl-ref/#InverseFunctionalProperty-def`

of LMD cubes cubes described by a QB4OLAP based ontology onto. The output is a new collection of LMD cubes, which are in either Star-pattern or Denormalized-pattern, depending on the mode argument, along with the ontology describing the new cubes. The algorithm uses the recursive function `MergeLevel()` to merge level instances in a dimension hierarchy together. This function is defined in Algorithm 2.

To create a Star-pattern the `MergeStar()` function is called. It uses the lowest level of a dimension to associate every fact to a dimension instance, as defined in Algorithm 3. To create a Denormalized-pattern the `MergeDenorm()` is used. It merges the dimension instances into every fact as defined in Algorithm 4. We also use the function `MergeURIs()`, which combines two or more URIs into a new URI. The exact method used to do this is not important as long as the new URI does not risk URI collision with other resources. In practice we use the first URI and concatenate the local name or the remaining URIs, e.g.:

`MergeURIs(domain_a#a,domain_b#b) = domain_a#a_b`

It should be noted that we operate with two kinds of variables in our algorithms, *BGP variables*, which are prefixed with a question mark "?", and regular *algorithm variables*. In some places we use algorithm variables in a BGP. This does not make it a BGP variable, instead it is used as a constant (URI, literal, or blank node) in the BGP, by extracting the value of the algorithm variable at the time the BGP is constructed. Also note that a graph can be produced by projecting exactly three variables from an evaluation of a BGP. In the following we create a graph G' based on the evaluation of a BGP on the graph G:

$$G' = \{(s, p, o) \in G \, (?s \text{ rdf:type } t \ . \ ?s \ ?p \ ?o)\}$$

G' contains all triples with the subject s, which is an instances of the class t, in the graph G. If a BGP does not contain variables, the result of an evaluation is a boolean value, indicating if a set of triples (a BGP without variables is a set of triples) is present in a given graph.

SWOD, see Algorithm 1, loops through all LMD cubes (resources of the type qb:DataSet). In LTPC-H we have three cubes: `ltpch:lineitemCube`, `ltpch:ordersCube`, and `ltpch:partSupplierCube`. For each cube, the lowest level of each dimension is identified at line 3. The `ltpch :lineitemCube` cube has two dimensions namely Orders and Partsupplier. We loop over the dimensions, with their lowest levels, in the lines 4–11. Each lowest level instance is merged together with the instances at each higher level, using the recursive function `MergeLevel()`. Depending on mode we use either `MergeStar()` or `MergeDenorm()` to integrate the dimension instances with the facts. We also generate a specific ontology extension based on mode, by using either `GetStarOnto()` or `GetDenormOnto()`. The lines 12–14 loop over the facts in the current cube and add these to the Star-pattern cube collection cubes′ along with the measures and attributes associated with each fact. Finally we combine the input ontology and extension ontology (line 15) and return it with the converted cubes (line 16).

Algorithm 1: Conversion of data from Snowflake-pattern to Star-pattern or Denormalized-pattern.

Input: LMD cubes cubes described by a QB4OLAP ontology onto. Both cubes and onto are graphs. mode indicates the expected output pattern, Star-pattern or Denormalized-pattern.
Output: LMD cubes cubes′ described by the ontology onto′. Both cubes′ and onto′ are graphs.

```
1  Function SWOD(cubes,onto,mode) is
2      foreach dataSet ∈ onto(?dataSet a qb:DataSet) do
3          lowestLvls = {(lvl, dim) ∈
           onto(dataSet qb:structure ?structure.
           ?structure qb:component ?component.
           ?component qb4o:level ?lvl.
           ?lvl qb:inDimension ?dim)};
4          foreach (lvl, dim) ∈ lowestLvls do
5              dimInsts = MergeLevel(cubes,onto,lvl,dim);
6              if mode = SWOD-S then
7                  cubes′ = cubes′ ∪
                   MergeStar(cubes,dimInsts,lvl,dataSet);
8                  onto′ = onto′ ∪ GetStarOnto(onto,lvl,dim);
9              if mode = SWOD-D then
10                 cubes′ = cubes′ ∪
                   MergeDenorm(cubes,dimInsts,lvl,dataSet);
11                 onto′ = onto′ ∪ GetDenormOnto(onto,lvl,dim);
12         foreach
           (fact, prop, obj) ∈ cubes(?fact qb:dataSet dataSet.
           ?fact ?prop ?obj) do
13             if onto(prop a qb:AttributeProperty) ∨
               onto(prop a qb:MeasureProperty) then
14                 cubes′ = cubes′ ∪ {(fact, prop, obj)};
15     onto′ = onto′ ∪ onto;
16     return cubes′,onto′;
```

6.1 Merging Level Instances

The `MergeLevel()` function, see Algorithm 2, recursively merge level instances together into dimension instances which are used in a Star-pattern cube or further merged with the facts of a Denormalized-pattern cube. The function initially receives a level of a dimension along with the graph containing the instances, cubes, and the ontology describing the dataset, onto. The function uses recursion to merge parent level instances, before merging these with the instances of the current level. When the recursion reaches its depth – a level with no parent levels – it returns the instances of the current level with all their triples, which have attribute properties as predicates. Note that we use `MergeURIs()` to alter the predicates before adding them to the result graph. When the top call returns a set of merged level instances, we call these dimension instances.

First we run through each parent level – recall that the Partsupplier level has two parents, the Part and the Supplier level – in lines 2–8. Here, we recursively merge the parent levels. For each instance at the parent level (lines 4–8), we find children at the current level, if any, in lines 5–6

Algorithm 2: Generation of level instances through recursively merging parent levels.

Input: LMD cubes cubes described by a QB4OLAP ontology onto. Both cubes and onto are collections of triples. lvl is a level in dim, both of which are resources defined in onto.

Output: A collection of triples dimInsts of dimension instances.

```
1  Function MergeLevel(cubes,onto,lvl,dim) is
2      foreach parLvl ∈ onto(lvl qb4o:parentLevel ?parLvl .
       ?parLvl qb:inDimension dim) do
3          parInsts = MergeLevel(cubes,onto,parLvl,dim);
4          foreach parInst ∈ parInsts(?parInst qb4o:inLevel [ ])
           do
5              foreach lvlInst ∈ cubes(?lvlInst parLvl parInst)
               do
6                  dimInsts =
                   dimInsts ∪ {(lvlInst, prop, obj)|(prop, obj) ∈
                   parInsts(parInst ?prop ?obj)};
7              if cubes(?lvlInst parLvl parInst) = ∅ then
8                  dimInsts = dimInsts ∪ {(parInst, prop, obj) ∈
                   parInsts(parInst ?prop ?obj)};
9      foreach prop ∈ onto(lvl qb4o:hasAttribute ?prop) do
10         dimInsts =
           dimInsts ∪ {(lvlInst, MergeURIs(lvl,prop) obj) ∈
           cubes(?lvlInst qb4o:inLevel lvl. ?lvlInst prop ?obj)};
11     dimInsts = dimInsts ∪ {(lvlInst, qb4o:inLevel, lvl)|lvlInst ∈
       cubes(?lvlInst qb4o:inLevel lvl)} ;
12     return dimInsts;
```

and merge them by extracting every triple related to the parent instance and creating a new similar triple with the current level instance as the subject (instead of the parent instance). Note that we use [ ] to indicate an anonymous resource in line 4. To accommodate for unbalanced hierarchies, we add ancestor level instances which do not have any children in the current level in lines 7–8. Then we add the triples relevant to the current level to the result graph cubes' in lines 9–11. Relevant triples are those with predicates declared as attribute properties of the current level (through the use of qb4o:hasAttribute) in the ontology and those which indicate the level (with the predicate qb4o:inLevel). Finally, the merged level instances are returned in line 12.

Algorithm 3: Linking facts to dimension levels.

Input: cubes is a graph containing MD cubes in Snowflake-pattern. dimInsts is a set of triples with merged dimension instances. lvl is the initial level of dimension being merged. dataSet is a dataset in cubes.

Output: dimInsts′ is a graph containing all dimension instances and links between facts of dataSet and the dimension instances.

```
1  Function MergeStar(cubes,dimInsts,lvl,dataSet) is
2      dimInsts′ = dimInsts ∪ {(fact, lvl, lvlInst)|(fact, lvlInst) ∈
       cubes(?fact qb:dataSet dataSet. ?fact lvl ?lvlInst)}
       return dimInsts′
```

The function MergeStar() in Algorithm 3 is used by SWOD() to link the facts from the cube dataSet with the dimension instances returned from MergeLevel(). The entire set of dimension instances (dimInsts) is added to the result graph. For every fact in the given dataset we select every triple with the predicate of the lowest level (lvl) of the dimension and its object. These triples are added to the result graph. The objects of these triples are originally level instances, but are used as dimension instances when converted to Star-pattern. By adding these we ensure that the facts are associated with the correct dimension instance.

Algorithm 4: Merging dimension instances with facts.

Input: cubes is a graph containing MD cubes in Snowflake-pattern. dimInsts is a set of triples with merged dimension instances. lvl is the initial level of the dimension being merged. dataSet is a dataset in cubes.

Output: dimInsts′ is a graph containing all the facts of dataSet merged with the dimension instances to which they related.

```
1  Function MergeDenorm(cubes,dimInsts,lvl,dataSet) is
2      foreach lvlInst ∈ dimInsts(?lvlInst qb4o:inLevel [ ]) do
3          foreach fact ∈ cubes(?fact lvl lvlInst) do
4              dimInsts′ =
               dimInsts′ ∪ {(fact, MergeURIs(dim,prop), obj)|
               (prop, obj) ∈ dimInsts(lvlInst ?prop ?obj)};
5          if cubes(?fact lvl lvlInst) = ∅ then
6              dimInsts′ =
               dimInsts′ ∪ {(lvlInst, MergeURIs(dim,prop), obj)|
               (prop, obj) ∈ dimInsts(lvlInst ?prop ?obj)};
7      return dimInsts′
```

In Algortihm 4 we define the function MergeDenorm(). The lines 2–6 are very similar to the lines 4–8 of Algorithm 2. Instead of having a set of parent level instances and a set of current level instances, we have a set of dimension instances and a set of facts. The dimension instances are merged with the facts by selecting all triples related to the dimension instance and creating new similar triples with the facts as subjects. As in MergeLevel() we accommodate for unbalanced hierarchies by also adding dimension instances, which do not have facts associated with them – again, this happens for customers which are not related to any orders and thereby not related to any lineitems.

6.2 Generating Ontology Extension

Depending on which pattern we wish to generate, we need a certain extension to our ontology. This ontology extension describes the new properties, which are introduced after the pattern transformation.

In Algorithm 5 we define GetStarOnto(), which generates an ontology extension for a Star-pattern cube. We traverse all attribute properties of the input level, lvl, in the lines 2–6 and define a new property by merging the URIs of the attribute property and the level. We make the new properties sub-properties of the attribute properties on which they are based. If the attribute property is owl:sameAs or is an inverse functional property we define the new property

Algorithm 5: Generate ontology extension for Star-pattern cube.

Input: onto is a graph containing the ontology for an MD cubes in Snowflake-pattern. lvl is a level in dim, both of which are resources located in onto.
Output: onto′ is a graph containing an ontology which extends onto to describe an MD cube in Star-pattern.

```
1 Function GetStarOnto(onto,lvl,dim) is
2     foreach prop ∈ onto(lvl qb4o:hasAttribute ?prop) do
3         if onto(prop a owl:InverseFunctionalProperty)∨
          prop = owl:sameAs then
4             onto′ =
              onto′ ∪ {(MergeURIs(lvl,prop), a, rdfProperty),
              (MergeURIs(lvl,prop), rdfs:seeAlso, prop)};
5         else
6             onto′ = onto′ ∪
              {(MergeURIs(lvl,prop), rdfs:subProprtyOf, prop)};
7     foreach parLvl ∈ onto(lvl qb4o:parentLevel ?parLvl.
      ?parLvl qb:inDimension dim) do
8         onto′ = onto′ ∪ GetStarOnto(onto,parLvl,dim)
9     return onto′
```

Algorithm 6: Generate ontology extension for Denormalized-pattern cube.

Input: onto is a graph containing the ontology for an MD cubes in Snowflake-pattern. lvl is a level in dim, both of which are resources located in onto.
Output: onto′ is a graph containing an ontology which extends onto to describe an MD cube in Denormalized-pattern.

```
1 Function GetDenormOnto(onto,lvl,dim) is
2     foreach prop ∈ onto(lvl qb4o:hasAttribute ?prop) do
3         if onto(prop a owl:InverseFunctionalProperty)∨
          prop = owl:sameAs then
4             onto′ =
              onto′∪{(MergeURIs(dim,lvl,prop), a, rdfProperty),
              (MergeURIs(dim,lvl,prop), rdfs:seeAlso, prop)};
5         else
6             onto′ = onto′ ∪
              {(MergeURIs(dim,lvl,prop), rdfs:subProprtyOf, prop)};
7     foreach parLvl ∈ onto(lvl qb4o:parentLevel ?parLvl.
      ?parLvl qb:inDimension dim) do
8         onto′ = onto′ ∪ GetDenormOnto(onto,parLvl,dim)
9     return onto′
```

as a "fresh" property with a similarity to the old (through the rdfs:seeAlso predicate) instead. In the lines 7–8 we loop through all parent levels and recursively call `GetStarOnto()` with the parent levels as inputs. The resulting triples from these are added to the result graph, which is then returned.

An ontology extension of a Denormalized-pattern cube is generated by `GetDenormOnto()`, defined in Algorithm 6. It functions in much the same way as `GetStarOnto()`, with the exception that it also uses the dimension URI to create the new properties. This is because all attribute properties in a Denormalized-pattern are run through `MergeURIs()` twice, once in `MergeLevel()` and once in `MergeDenorm()`. Since the properties used as predicates in the instance data are run through `MergeURIs()` twice rather than once with more arguments, as in `GetDenormOnto()`, we require `MergeURIs()` to obey the following:

$$\texttt{MergeURIs}(a,b,c) = \texttt{MergeURIs}(a,\texttt{MergeURIs}(b,c))$$

7. EXPERIMENTAL SETUP

In order to understand how the Snowflake-pattern, Star-pattern, and Denormalized-pattern perform we have applied them on the LTPC-H. Now we describe the setup of our experiments.

All experiments are run a PowerEdge^{TM}C1100 server running ubuntu server 14.04 with two Intel(R) Xeon(R) CPU L5520 @ 2.27GHz. It has 72 GB DDR3 DIMM RAM and a Western Digital 1 TB, Internal, 5400 RPM hard-disk.

We use Sesames native store as the physical storage of the data. The blank nodes, URIs, and literals are stored in a data dictionary implemented as a hashtable. A b+tree is used to store the quads (subject, predicate, object, context) and also functions as the index. This means that an index is a quadstore and each additional index is an additional copy of the quadstore where only the triple order differs [13, p. 101][15, 14]. We use two indexes for our repositories: SPOC (subject, predicate, object, context) and POSC (predicate, object, subject, context).

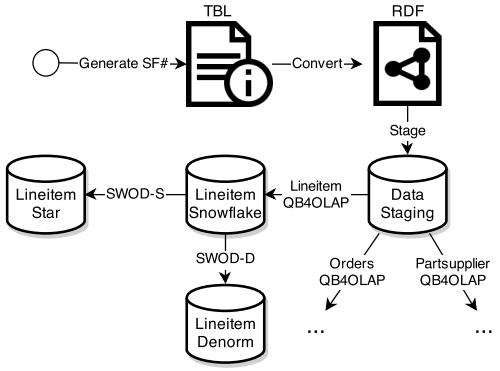


Figure 10: The process of how TPC-H data is transformed to RDF data organized in cubes with Snowflake-pattern, Star-pattern, and Denormalized-pattern.

The process of making the data ready for querying consists of six steps shown in Figure 10. We measure the time of these steps and present them in Section 8. The first step is called **Generate SF#** and is the generation of the TPC-H data according to some scaling factor. We use the TPC-H DBGEN tool to do this. This data is then converted to RDF in the **Convert** step and we correct the `xsd:dateTime` to `xsd:date`. The next step **Stage** is to load the RDF data into the Sesame native store. In the **Lineitem QB4OLAP** step we add QB4OLAP– by defining dimensions, levels and their relations, etc. – and thus generating the Snowflake-pattern

cube. The next two steps are independent of each other. We generate Star-pattern using the **SWOD-S** algorithm and likewise we generate Denormalized-pattern using the **SWOD-D** algorithm. We explain the process in greater detail in Appendix A.6. These last three steps are performed once for each data cube – Lineitem, Orders, Partsupplier.

We construct a new data staging area for each of the scaling factors we test. For each of these scaling factors we store each data cube separately. Finally, every data cube is stored in three different repositories using the three patterns: Snowflake-pattern, Star-pattern, and Denormalized-pattern. We define the size of a repository as its size on the hard disk. We measure the size of these Sesame repositories and the number of triples in each, this is presented in the next section.

We define 22 TPC-H SPARQL queries based on the queries from BIBM. These are defined in three variants such that they fit the Snowflake-pattern, Star-pattern, and Denormalized-pattern. By doing so we are able to compare the query evaluation time of the three patterns, which we do in the next section. In Appendix G all the queries are presented. To show how this in done, recall the query: "Show the total revenue per nation for nations, which have at least a population of thirty million". The strategy is to get the attributes needed to calculate the revenue and find the name of the nation from which the orders originates (the nation of the customer). Then we use data from the Wikipedia endpoint called dbpedia[11] to get the population of the different nations – which we do not have available locally. Last we are able to make a filter such that we only receive nations with a population of more than 30.000.000 and we make a case insensitive comparison of nation names to match nations from dbpedia and our local nations. Code Snippet 2 shows our example query adapted for Snowflake-pattern.

```
SELECT DISTINCT ?n_name1 ?pop
  (sum(?price*(1-?discount)) as ?revenue)
WHERE {
  ?li qb:dataSet ltpch:lineitemCube ;
    ltpch:l_lineextendedprice ?price ;
    ltpch:l_linediscount ?discount ;
    ltpch:l_has_order ?ord .
  ?ord ltpch:o_has_customer ?cust .
  ?cust ltpch:c_has_nation ?nation .
  ?nation ltpch:n_name ?n_name1 .
  SERVICE <http://dbpedia.org/sparql/> {
    ?nation2 a dbponto:Country ;
      dbpprop:commonName ?n_name2 ;
      dbpprop:populationEstimate ?pop .
  }
  FILTER (
    REGEX(?n_name1, str(?n_name2), "i")
    && ?pop > 30000000
  )
}
GROUP BY ?n_name1 ?pop
```

Code Snippet 2: Snowflake-pattern version of our running example query.

[11] http://dbpedia.org/sparql

In Code Snippet 3 the query has been adapted to Star-pattern. Here we see that locally (not in `SERVICE` clause) we are only working with two resources at subject position: A lineitem fact and an orders dimension instance. In the Snowflake-pattern query we have three level instances instead of a single dimension instance. This makes the Star-pattern query slightly shorter, and arguably less complicated because less triple patterns are required to express the same conceptual query.

```
SELECT DISTINCT ?n_name1 ?pop
  (sum(?price*(1-?discount)) as ?revenue)
WHERE {
  ?li qb:dataSet ltpch:lineitemCube ;
    ltpch:l_lineextendedprice ?price ;
    ltpch:l_linediscount ?discount ;
    ltpch:l_has_order ?ord .
  ?ord ltpch:nation_name ?n_name1 .
  SERVICE <http://dbpedia.org/sparql/> {
    ?wiki_nation a dbponto:Country ;
      dbpprop:commonName ?n_name2 ;
      dbpprop:populationEstimate ?pop .
  }
  FILTER (
    REGEX(?n_name1, str(?n_name2), "i")
    && ?pop > 30000000
  )
}
GROUP BY ?n_name1 ?pop
```

Code Snippet 3: Star-pattern version of our running example query.

Code Snippet 4 contains the Denormalized-pattern query. Locally there are neither level instances nor dimension instances, only facts, at subject position. This allows us to reach dimension attribute values directly from the fact and arguably make the query simpler.

```
SELECT DISTINCT ?n_name1 ?pop
  (sum(?price*(1-?discount)) as ?revenue)
WHERE {
  ?li qb:dataSet ltpch:lineitemCube ;
    ltpch:l_lineextendedprice ?price ;
    ltpch:l_linediscount ?discount ;
    ltpch:order_nation_name ?n_name1 .
  SERVICE <http://dbpedia.org/sparql/> {
    ?wiki_nation a dbponto:Country ;
      dbpprop:commonName ?n_name2 ;
      dbpprop:populationEstimate ?pop .
  }
  FILTER (
    REGEX(?n_name1, str(?n_name2), "i")
    && ?pop > 30000000
  )
}
GROUP BY ?n_name1 ?pop
```

Code Snippet 4: Denormalized-pattern version of our running example query.

Ten of the TPC-H queries use functions which are not a part of the Sesame implementation, these functions are `dateadd`, `equals`, and `or`. We have implemented these as SPARQL extension functions, please see Appendix C for implementation details.

As mentioned, all the TPC-H queries have variables which need to be instantiated, BIBM does this based on query description files, which are included in Appendix G. This instantiation is based on a seed. The order of the queries are randomized based on this seed, we call a set of instantiated queries a querymix. For each scaling factor we run six different querymixes, each querymix is run on the three patterns. To calculate the average query evaluation time of a query, we remove the fasted and the slowest query evaluation time and calculate the average of the remaining.

8. EVALUATION

In this section we evaluation LTPC-H. We evaluate the three LMD cubes where we vary the patterns and the scaling factor such that we have 27 LMD cubes. Figure 11 illustrate these cubes. Each cube is evaluated on three measures, these are *load time*, *storage size*, and *query evaluation time*. For the sake of brevity we only present results for the Lineitem-cube here and refer to the results of the other cubes in Appendix F. We refer to the previous section for setup details.

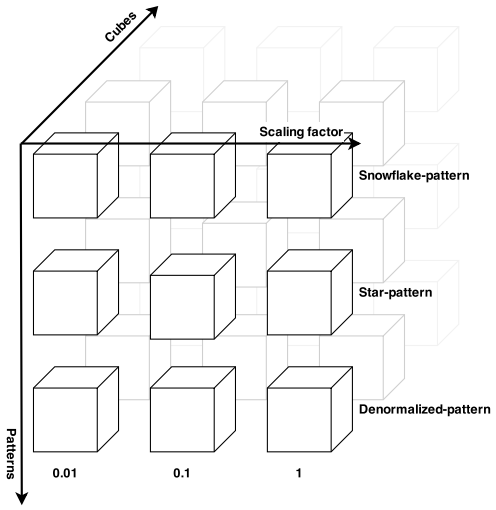


Figure 11: The x axis show the different scaling factor, the z axis show the patterns, and the z axis shows the LMD cubes.

Step	0.01	0.1	1
Generate	9	12	36
Convert	8	58	570
Stage	49	410	24,187
QB4OLAP	66	552	20,619
SWOD-S	58	573	26,198
SWOD-D	133	1,419	

Figure 12: Seconds spent for performing the steps of our process.

Figure 12 shows the load times in seconds for how long it takes to generate the Lineitem-cube. It shows the time it takes to: Generate TPC-H data, convert it to RDF, load it into a staging area, construct the LMD Snowflake-pattern cube with QB4OLAP, construct the LMD Star-pattern cube, and construct the LMD Denormalized-pattern cube. We measure the load times for scaling factors 0.01, 0.1, and 1.

There are no results for the Denormalized-pattern for scaling factor 1 because the process took an unreasonable amount of time and we had to stop it. The load times of Generate and Convert shows the performance of the DBGEN tool and the BIBM conversion tool. Stage is the time it takes for Sesame native store to load the data from the converted RDF file (in turtle format) into its repository. Since the data amount increase with a factor of ten between scaling factor 0.01, 0.1, and 1, we expect load times to do approximately the same. However at scaling factor 1 this does not hold. The reason for this lies in the implementation of Sesames native store. Despite having sufficient memory available, Sesame uses and maintains the indices on the hard drive. Sesame scans these indices before inserting new triples in order to detect and avoid duplicates and find the place to put the new triple, thus requiring and increasingly high amount of disk reads as the repository grows. Recall that the index contain a complete instance of the data thus making the index expensive to read in terms of disk operations. These reasons cause the load of Lineitem-cube with scaling factor 1 as Denormalized-pattern to run for an unreasonably long time. In QB4OLAP and SWOD-S we also see that the load time greatly increase at scaling factor 1.

Pattern	0.01	0.1	1
Snowflake	132	1,032	45,412
Star	190	1,605	71,610
Denormalized	265	2,451	

Figure 13: Seconds spent from generation to finished LMD cube.

The steps performed when loading the LMD cubes are sequential (with SWOD-S and SWOD-D as exceptions). In Figure 13 we see the time required to reach a functional LMD, i.e. the accumulated times of Figure 12. It takes 138% longer to generate Denormalized-pattern at scaling factor 0.1 than Snowflake-pattern and it takes 56% longer time to generate Star-pattern.

Pattern	0.01	0.1	1
Snowflake	1,521,825	15,192,665	151,834,525
Star	1,771,500	17,691,440	176,824,356
Denormalized	4,159,183	41,511,007	

Figure 14: The number of triples in Lineitem-cube using different patterns.

The storage size is measured in two ways: The number of triples in the repository and the physical size of the Sesame native store repository on the disk. In Figure 14 we see the number of instance triples in the Lineitem-cube repositories with different scaling factors and with different patterns. We can see that the amount of triples increase when we convert a LMD cube from Snowflake-pattern to Star-pattern or Denormalized-pattern.

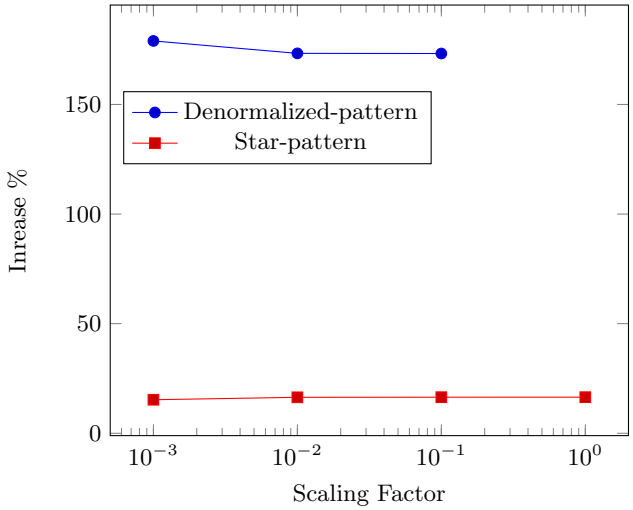


Figure 15: The growth of triples compared to Snowflake-pattern in percent.

Pattern	0.01	0.1	1
Snowflake	122	1,211	12,049
Star	139	1,380	13,760
Denormalized	288	2,866	

Figure 16: The disk size of repositories containing the Lineitem-cube using different patterns. The sizes are in mega bytes.

In Figure 15 we can see how many percent the Star-pattern and Denormalized-pattern cubes are larger than Snowflake-pattern. Note that we have included scaling factor 0.001 in this figure. When the scaling factor is increased, the growth percentage of Star-pattern increase from 15.3% to 16.5% while the growth percentage of Denormalized-pattern decrease from 179% to 173.2%. While these results are promising for the Denormalized-pattern cube, it still contains more than twice as many triples as either of the otherpatterns at a scaling factor of 0.1. The number of triples in the ontologies are 403, 454, and 463 in Snowflake-pattern, Star-pattern, and Denormalized-pattern respectively. The Star-pattern and Denormalized-pattern ontologies have more triples than Snowflake-pattern because they have to avoid URI collisions and therefore create new properties as described in Section 6. Because Denormalized-pattern has to avoid collisions between dimensions as well as level, it contains more new properties than Star-pattern. In Figure 16 we see the size of the Lineitem-cube repositories at different scaling factors and with different patterns. The storage size is approximately increased ten fold, when the scaling factor is increased ten fold. However, the ratio between two repositories of the same pattern with a factor 10 between scaling factors is a little less than ten. This is in part due to the overhead required to store namespaces, which for all scaling factors is the same size, and in part because of the storage of fixed sized levels such as Nation and Region.

There are 17 queries for the Lineitem-cube. These can be seen, along with the ones for the Partsupplier-cube and

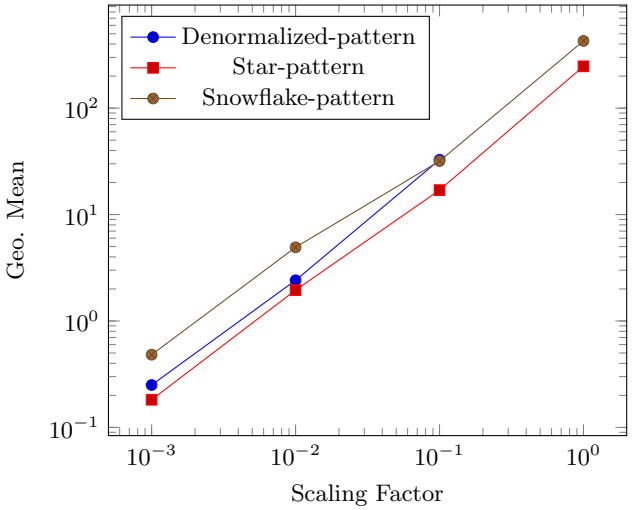


Figure 17: Geometric mean of queries run on Lineitem-cube with double logarithmic axes.

Orders-cube, in Appendix G. Figure 17 shows the geometric mean for the query evaluation times on different scaling factors for the different patterns. Note that we have included results for cubes with the scaling factor 0.001. Figure 18 shows the individual query evaluation times of the queries run on different scaling factors and patterns. In Appendix F bar charts of the results are show. Queries which are not able to finish within 1000 seconds are stopped and we write $>$ 1000 to mark this. We calculate the minimum average and the minimum geometrical mean, we say that these are minimum because the queries which timeout are included as the query evaluation time 1000 in the calculation. The first column in the figure is the names of the queries, seconds is scaling factor 0.01, third is scaling factor 0.1, and last is scaling factor 1. Recall we did not load Denormalized-pattern scaling factor 1 because it timed out, we are therefore not able to present any results for it. Further we experienced hardware problem during the evaluation of scaling factor 1, therefore the results only represent a single run on warm cache instead the average of the middle four runs. Figure 19 compares the Lineitem-cube queries in different aspects. These aspects are the number of substitution parameters in the query, the number of Basic Triple Pattern (BTP) in the query, how many `FILTER` clauses, maximum level depth, distinct levels spanned, number of subqueries, and on which patterns can the query be answer on SF 1 without exceeding 1000 seconds. Due to the fact that in scaling factor 0.1, Denormalized-pattern is slower on all query except queries 7 and 12, this leads us to conclude that it not a attractive pattern to use on the LTPC-H dataset. In general we see that queries with a high number of distinct levels, such as queries 7 and 8, are slow in Snowflake-pattern and fast in Star-pattern. The opposite is also true, when there is a low number of distinct levels, the Snowflake-pattern performs well as seen in query 1, 3 and 6. Queries have to perform subject-object joins every time a level is used in Snowflake-pattern where Star-pattern only does this at the lowest dimension level where after it performs the efficient subject-subject joins. Subject-subject joins are faster because the cubes are stored in indices with the subject as

the primary index attribute. This contributes to the fact that Star-pattern is faster when there is a high number of distinct levels. Query 21 exceed 1000 seconds on all patterns. The query contains a `FILTER NOT EXISTS` which may be the reason why Sesame is not able to answer the query in less the 1000 seconds. Other queries also exceed 1000 seconds in scaling factor 1, this is often caused by a large result set. Snowflake-pattern timeout on eight queries and Star-pattern timeout on six where four of them are shared. In terms of average and geometrical mean, the Star-pattern cubes have the lowest query evaluation times.

As presented in the introduction we evaluate the business question "Show the total revenue per nation for nations, which have at least a population of thirty million". Recall that we use RDF as underlying format because we want to benefit from the native ad-hoc data integration. The query serves as a proof of concept, to show that it is indeed possible to query internal and external data in an MD context. The queries are explained in Section 7. We run the queries on the three patterns on scaling factor 0.1. The query evaluation times are 123.71s, 121.59s, and 97.18s for Snowflake-pattern, Star-pattern, and Denormalized-pattern respectively. This simple query have similar results for Snowflake-pattern and Star-pattern but here Denormalized-pattern the lowest query evaluation time.

Based on the three measures load time, storage size, and query evaluation time we have gained a higher understand of when to use the three patterns. We see that Denormalized-pattern has more than twice as many triples in the repository as the other patterns and show very little potential in terms of query evaluation time. The load time is also the much larger Denormalized-pattern than the other patterns. Snowflake-pattern and Star-pattern both show good results and different types of queries. Star-pattern excel at queries that span several distinct levels and Snowflake-pattern at queries that span few distinct levels. For some of these queries Star-pattern outperforms Snowflake-pattern by orders of magnitude. Despite that Star-pattern has an overhead in form of more triples and a higher load time, it is faster on average and is therefore a suitable alternative to Snowflake-pattern.

9. CONCLUSIONS AND FUTURE WORK

Motivated by the increasing amount of research in the use of Semantic Web technologies for ad-hoc data integration in Decision Support Systems (DSS), we explore the potential of using RDF to store both local and situational data. RDF is an attractive format because the query language SPARQL, which operates on RDF data, has native support for ad-hoc data integration through federated queries. We define the term Linkable Multidimensional (LMD) cubes as Multidimensional (MD) cubes in RDF format. We use the de facto standard MD vocabulary (QB extended with QB4OLAP) for describing the LMD cubes, on the instance and ontological level. We present three logical patterns for modeling LMD cubes, namely Snowflake-pattern, Star-pattern, and Denormalized-pattern. These patterns are inspired by techniques known from the relational OLAP world.

We present the novel algorithm Semantic Web OLAP Denormalizer (SWOD), which can transform an LMD cube in Snowflake-pattern into Star-pattern and Denormalized-pattern. It accounts for unbalanced hierarchies and name collisions of properties belonging to different levels and dimensions. This allows us to only model the data once and convert it to the pattern(s), which is/are best suited for a given use case.

We convert the TPC Benchmark^{TM}H (TPC-H) datasets of different sizes to RDF and call it Linkable TPC-H (LTPC-H). We annotate the datasets with QB4OLAP, as a novel contribution, to create LMD cubes modeled with Snowflake-pattern. We run these through SWOD to create LMD cubes in Star-pattern and Denormalized-pattern. The largest LMD cube has 177 million triples. Our evaluation of the cubes include load time, repository size, and query evaluation time. We measure the query evaluation time by converting the 22 TPC-H queries to SPARQL and evaluating them on our different LMD cubes. The LMD cubes are available on our SPARQL endpoint `http://www.extbi.lab.aau/sparql`.

In our evaluation we found that generally Star-pattern and Snowflake-pattern both perform well on load times, repository size, and query evaluation time, while Denormalized-pattern is outperformed by the other patterns. For the LTPC-H dataset the Star-pattern is about 15% larger than Snowflake-pattern, where Denormalized-pattern is more than 170% larger. The time it takes to create a Star-pattern LMD cube is 59% longer than the Snowflake-pattern, while the Denormalized-pattern is 138% slower than the Snowflake-pattern. On average, the query evaluation time of the Snowflake-pattern cubes evaluate our queries at least 70% slower than the Star-pattern cubes. Furthermore, the Star-pattern cubes are able to handle more queries than Snowflake-pattern cubes without timing out. On average the Denormalized-pattern cubes perform comparable with the Snowflake-pattern cubes. While none of the patterns prove to be strictly better than the others, we recommend Star-pattern over the other two patterns for storing the LTPC-H cubes.

We evaluate the business question "Show the total revenue per nation for nations, which have at least a population of thirty million" and find that the lowest query evaluation time is when using Denormalized-pattern. Because we are able to evaluate the query we show that LMD cubes is a viable option for answering business questions with ad-hoc data integration.

Interesting research directions are developing algorithms for automatically converting queries of one pattern to another along with testing performance of the patterns on different datasets. Work in these directions will further improve the versatility of LMD cubes and help generate general heuristics for the use of the patterns.

10. ACKNOWLEDGMENT

We would like to thank our supervisors Torben B. Pedersen and Katja Hose for fruitful guidance within the areas of BI and semantic web. We would also like to thank Anders T. Olesen for providing the test server and providing maintenance of it.

Query	Snow.	Star.	Denorm.	Snow.	Star.	Denorm.	Snow.	Star.
1	5.28	5.42	5.31	54.67	56.98	59.11	545.50	> 1000
3	0.85	1.54	1.56	8.82	17.23	33.11	> 1000	> 1000
4	0.29	0.26	0.64	2.87	2.26	33.03	272.92	24.95
5	1.00	0.92	0.88	9.49	11.26	9.97	275.38	101.78
6	0.87	0.89	0.88	26.00	27.81	29.73	257.69	> 1000
7	> 1000	4.62	1.62	> 1000	55.11	41.60	> 1000	569.36
8	> 1000	0.09	1.17	> 1000	0.76	15.71	> 1000	107.25
9	3.11	2.33	5.32	32.57	25.60	59.69	> 1000	> 1000
10	3.64	0.45	0.76	38.64	4.19	7.26	685.27	46.10
12	3.98	2.91	2.83	49.96	49.39	41.61	606.95	422.95
14	2.38	2.36	0.53	28.44	28.31	33.41	> 1000	> 1000
15	2.34	2.35	2.42	21.80	22.48	25.04	294.15	265.86
17	1.25	1.38	4.11	11.70	14.37	435.51	628.97	396.98
18	1.33	1.31	1.39	12.48	12.48	14.04	> 1000	243.09
19	3.92	4.66	4.65	38.03	49.75	58.04	> 1000	587.65
20	1.60	1.53	2.53	15.24	15.50	27.83	287.65	260.77
21	> 1000	> 1000	> 1000	> 1000	> 1000	> 1000	> 1000	> 1000
>Average	168.44	57.39	57.59	186.15	77.42	106.93	658.58	501.48
>Geo. Mean	4.92	1.95	2.41	31.83	16.97	32.74	428.16	247.20

(a) scaling factor 0.01 **(b) scaling factor 0.1** **(c) scaling factor 1**

Figure 18: Query Evaluation Times of queries on the different patterns with scaling factor 0.01, 0.1, and 1 respectively.

Query	Substitution Parameters	BTPs	Filters	Level Depth	Distinct Levels	Sub-queries	Completion
1	1	8	1	0	0	0	Sn
3	2	10	3	2	2	0	
4	1	6	3	1	1	0	Sn,St
5	2	13	4	4	7	0	Sn,St
6	3	5	5	0	0	0	Sn
7	2	12	6	3	6	1	St
8	2	16	4	4	8	2	St
9	1	13	1	3	5	1	
10	1	15	3	3	3	0	Sn,St
12	3	7	5	1	1	0	Sn,St
14	1	7	2	2	2	1	
15	1	17	5	2	2	3	Sn,St
17	2	11	3	2	2	1	Sn,St
18	1	12	2	2	2	1	St
19	6	11	20	2	2	0	St
20	3	12	5	3	4	2	Sn,St
21	1	18	6	3	5	2	

Figure 19: The queries of the Lineitem-cube compared on different parameters.

References

[1] 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.

[2] Chris Adamson and Mike Venerable. *Data Warehouse Design Solutions*. John Wiley & Sons, Inc., New York, NY, USA, 1998. ISBN 0-471-25195-X.

[3] Alex Bondo Andersen and Kim Ahlstrøm Jakobsen. Linking open data: Through a danish agricultural & business-based case study, Dec 2013. Student report.

[4] FBI Productions ApS. Funny business inc. URL `http://www.fbi.dk/`. Retrieved: December 9th 2013.

[5] Ronald Barber, Peter Bendel, Marco Czech, Oliver Draese, Frederick Ho, Namik Hrle, Stratos Idreos, Min-Soo Kim, Oliver Koeth, Jae-Gil Lee, Tianchao Tim Li, Guy M. Lohman, Konstantinos Morfonios, René Möller, Keshava Murthy, Ippokratis Pandis, Lin Qiao, Vijayshankar Raman, Richard Sidle, Knut Stolze, and Sandor Szabo. Business analytics in (a) blink. *IEEE Data Eng. Bull.*, 35(1):9–14, 2012. URL `http://dblp.uni-trier.de/db/journals/debu/debu35.html#BarberBCDHHIKKLLLMMMPQRSSS12`.

[6] David Beckett. Rdf/xml syntax specification (revised), February 2004. URL `http://www.w3.org/TR/n-triples/`. Retrieved: December 22th 2013.

[7] David Beckett. Rdf 1.1 n-triples, November 2013. URL `http://www.w3.org/TR/n-triples/`. Retrieved: December 22th 2013.

[8] David Beckett and Tim Berners-Lee. Turtle - terse rdf triple language, March 2011. URL http://www.w3.org/TeamSubmission/turtle/. Retrieved: December 22th 2013.

[9] Tim Berners-Lee. Linked data – design issues, July 2006. URL http://www.w3.org/DesignIssues/LinkedData.html. Retrieved: October 30th 2013.

[10] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax, March 2011. URL http://www.w3.org/TeamSubmission/n3/. Retrieved: December 22th 2013.

[11] Henrike Berthold, Philipp Rösch, Stefan Zöller, Felix Wortmann, Alessio Carenini, Stuart Campbell, Pascal Bisson, and Frank Strohmaier. An architecture for ad-hoc and collaborative business intelligence. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 13:1–13:6, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-990-9. doi: 10.1145/1754239.1754254. URL http://doi.acm.org/10.1145/1754239.1754254.

[12] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal On Semantic Web and Information Systems*, 2009.

[13] J. Broekstra. *Storage, Querying and Inferencing for Semantic Web Languages*. PhD thesis, Vrije Universiteit, 7 2005.

[14] J. Broekstra. Sesame native store technical description, May 2014. URL http://answers.semanticweb.com/questions/21881/why-is-sesame-limited-to-lets-say-150m-triples. Retrieved: May 13th 2014.

[15] J. Broekstra. Sesame native store technical description, May 2014. URL https://groups.google.com/forum/#!topic/sesame-users/SwPksydmIbE. Retrieved: May 13th 2014.

[16] João Pedro Costa, José Cecílio, Pedro Martins, and Pedro Furtado. One: A predictable and scalable dw model. In Alfredo Cuzzocrea and Umeshwar Dayal, editors, *DaWaK*, volume 6862 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2011. ISBN 978-3-642-23543-6. URL http://dblp.uni-trier.de/db/conf/dawak/dawak2011.html#CostaCMF11.

[17] Richard Cyganiak and Dave Reynolds. The rdf data cube vocabulary, 2014. URL http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/. Retrieved: March 5th 2014.

[18] Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax, February 2014. URL hhttp://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. Retrieved: May 20th 2014.

[19] Statistical Data and Metadata eXchange. Statistical data and metadata exchange. URL http://sdmx.org/. Retrieved: March 5th 2014.

[20] Orri Erling and Ivan Mikhailov. *Business Intelligence Extensions for SPARQL*. OpenLink Software, 2009. URL http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSArticleBISPARQL2.

[21] Lorena Etcheverry and Alejandro A. Vaisman. Enhancing olap analysis with web cubes. In *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*, ESWC'12, pages 469–483, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-30283-1. doi: 10.1007/978-3-642-30284-8_38. URL http://dx.doi.org/10.1007/978-3-642-30284-8_38.

[22] Lorena Etcheverry and Alejandro A. Vaisman. Qb4olap: A vocabulary for olap cubes on the semantic web. In Juan Sequeda, Andreas Harth, and Olaf Hartig, editors, *COLD*, volume 905 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. URL http://dblp.uni-trier.de/db/conf/semweb/cold2012.html#EtcheverryV12.

[23] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011. ISBN 9781608454303. URL http://linkeddatabook.com/.

[24] http://dbpedia.org/. About: Federal bureau of investigation. URL http://dbpedia.org/page/Federal_Bureau_of_Investigation. Retrieved: December 9th 2013.

[25] Christian S. Jensen, Torben Bach Pedersen, and Christian Thomsen. *Multidimensional Databases and Data Warehousing*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010. URL http://dx.doi.org/10.2200/S00299ED1V01Y201009DTM009.

[26] Benedikt Kämpgen and Andreas Harth. No size fits all – running the star schema benchmark with sparql and rdf aggregate views. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 290–304. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38287-1. doi: 10.1007/978-3-642-38288-8_20. URL http://dx.doi.org/10.1007/978-3-642-38288-8_20.

[27] Benedikt Kämpgen and Andreas Harth. Olap4ld - a framework for building analysis applications over governmental statistics. Accepted demo of the 11th ESWC 2014, 2014.

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

[29] Jose-Norberto Mazón, Jose Jacobo Zubcoff, Irene Garrigós, Roberto Espinosa, and Rolando Rodríguez. Open business intelligence: On the importance of data quality awareness in user-friendly data mining. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 144–147, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1143-4. doi: 10.1145/2320765.2320812. URL http://doi.acm.org/10.1145/2320765.2320812.

[30] Pat O'Neil, Betty O'Neil, and Xuedong Chen. Star schema benchmark. June 2009. URL http://www.cs.umb.edu/~poneil/StarSchemaB.PDF.

[31] Woody Pidcock and Michael Uschold. What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a metamodel?, 2010. URL http://infogrid.org/trac/wiki/Reference/PidcockArticle. Retrieved: December 22th 2013.

[32] Oscar Romero and Alberto Abelló. Open access semantic aware business intelligence. In *eBISS*, pages 121–149, 2013.

[33] *TPC BENCHMARK^{TM}H (Decision Support) Standard Specification*. Transaction Processing Performance Council (TPC), Building 572B Ruger St. (surface) P.O. Box 29920 (mail) San Francisco, CA 94129-0920, revision 2.16.0 edition, 2013.

[34] Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. Comparing data summaries for processing live queries over linked data. *World Wide Web*, 14(5-6):495–544, 2011.

[35] W3C. Properties, . URL http://www.w3.org/TR/rdf-schema/#ch_property. Retrieved: December 22th 2013.

[36] W3C. Linked data, . URL http://www.w3.org/standards/semanticweb/data. Retrieved: December 21th 2013.

[37] w3c. Resource description framework (rdf):concepts and abstract syntax, Febuary 2004. URL http://www.w3.org/TR/rdf-concepts/. Retrieved: December 13th 2013.

[38] W3C. Sparqlendpoints, January 2013. URL http://www.w3.org/wiki/SparqlEndpoints. Retrieved: December 21th 2013.

[39] W3C. Sparql 1.1 query language, March 2013. URL http://www.w3.org/TR/sparql11-query/. Retrieved: December 21th 2013.

A. BUSINESS INTELLIGENCE IN THE SEMANTIC WEB

This section is originates from [3, pp. 3-6]. Small modifications have been made and the paragraph about blank nodes is new.

A.1 Semantic Web

Here we describe the basics of the Semantic Web as a background for the article. This is not meant to be a full description of every aspect of the Semantic Web, for this we refer to our sources in our bibliography.

The Semantic Web – also occasionally called Web 3.0 or a component thereof – is sometimes described as an evolution of World Wide Web (WWW). The Semantic Web differentiates itself from WWW in two regards.

For one, Semantic Web links *resources*, where WWW links *documents*. The difference is that resources can *represent* physical things, where a document can only *describe* a physical thing. For instance: There are several documents describing FBI, but if you want to represent some data about FBI, you cannot uniquely identify FBI as part of your data. You could write human readable text which, from the context, would make it clear that it is the Federal Bureau of Investigation [24] you are talking about (not the Danish comedian organization Funny Business Incorporated [4]). In the Semantic Web you are able to uniquely define a resource, such as the Federal Bureau of Investigation, through Uniform Resource Identifiers (URIs). A URI can be used as part of an RDF graph, which we define further down, to add information about the resource which the URI represents.

The second difference is that the links/references between documents in WWW are through the anchor element in HTML. Such a link is not named – it is not clear what relationship(s) exists between the referring and the referred document. The links between resources in the Semantic Web *are* named. A link in the Semantic Web is called a *predicate*. A predicate is a URI, which is also present in the Semantic Web, hence the semantics of the given predicate (and relationship between resources) can be extracted.

A.2 Linked Data

Data in the Semantic Web can be in the form of Linked Data (LD). For LD Tim Berners-Lee has set up four principles which the data must live up to (quoted from Berners-Lee [9]):

1. "Use URIs as names for things"
2. "Use HTTP URIs so that people can look up those names."
3. "When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)"
4. "Include links to other URIs. so that they can discover more things."

This means that a URI should not only represent a resource, it should also locate the resource or at least information about the resource, such as links to other resources (through their URIs).

To store LD the format called RDF is the standard, defined by W3C [36]. A set of data stored in RDF is called an *RDF graph*, or simply a graph for shorthand notation, and consist of *triples*. Triple consists of three elements – hence the name – a *subject*, a *predicate*, and an *object*. The subject and predicate of a triple are always URIs. The object can either be a URI or a *literal* (string, number, data etc.). A triple indicates that there is some relationship between the subject and the object. The predicate defines the nature of this relationship. Two resources can have several different relationships. This can be modeled by having a triple for each relationship, each with a different predicate to indicate each relationship. A set of triples is, as mentioned, called a graph, because each URI and literal from subjects and objects can be viewed as nodes in a graph and every predicate will be a directed named edge from the subject to the object.

A.3 Resource Description Framework Formats

RDF does not in itself define a serialization format. There exists a number of serialization formats for RDF; Turtle [8], N-Triples [7], Notation 3 [10], and RDF/XML [6]. The Turtle syntax, which we use in this project, serializes RDF data by separating triples with a dot (.). The three parts of each triple are separated by a whitespace. URIs are written between less than (<) and greater than (>) symbols (used as angled brackets). Literals are written between quotes (") and can optionally be appended by two "hats" (^^) and a data type (string, integer, date, etc.).

There are shorthand notations for having several triples with the same subject, or subject and predicate, these are illustrated in Code Snippet 5. Semi-colon (;) is used to indicate that the following triple uses the same subject as the previous. Comma (,) is used when both subject and predicate is reused from previous triple. In total Code Snippet 5 has four triples, three of which have the same subject, namely `<resourceA>`, two these have the same predicate, `<predicate2>`. As shown, the types are prefixed with `xsd:`.

```
<resourceA> <predicate1> <resourceB> ;
            <predicate2> "String litteral"^^xsd:string ,
                         "String, without explicit data type" .

<resourceB> <predicate3> "123"^^xsd:integer .
```

Code Snippet 5: Example of RDF data in Turtle format.

Notation 3 and N-Triple are similar to the Turtle format. Notation 3 is more expressive than Turtle, since every legal Notation 3 document is not necessarily legal RDF data – Notation 3 allows blank nodes as predicates, which the RDF standard does not. The Turtle and Notation 3 formats offer a shorthand version of `rdf:type`, namely `a`, which is a predicate used to denote the type of a resource. N-Triple is very similar to Turtle with the exception that there are no shorthand notation for continuous use of a single subject, or subject and predicate, thus making it a bit more bloated to read and cumbersome to write by hand. RDF/XML is very different from the other formats, mainly because it is based on Extensible Markup Language (XML). We will not present further about RDF/XML, since we do not use it in this project.

A.4 Identifying & Locating Resources

Additionally to URIs uniquely defining resources, they also should be human readable. This means that from seeing the URI a person should have a good chance of knowing what it refers to, assuming he is knows the domain of the resource in question. When choosing a URI for a resource it should also be possible to look up the resource, which is why most URIs in the Semantic Web are defined as web addresses, which can be resolved with Hyper Text Transfer Protocol (HTTP), as mentioned as part of the four principles for LD in Section A.2. In this way the URI does not only serve as a unique identifier for a resource, but also as the means of reaching the resource. When looking up a URI which represents a real world object, e.g. a URI representing a person, this object cannot actually be transmitted over HTTP. In this case the server responding to the request should redirect to a page which describes the real world object. This can be achieved by the server sending HTTP response code 303 along with the address to a page describing the object, which tells the client to look at the alternative page [23, sec. 2.3.1] – browsers will automatically redirect the user when encountering a 303 response code. An alternative way to handle this is to use *hash URIs*. A hash URI is a HTTP address with the use of the hash (#) symbol. The part after the hash symbol, the *fragment identifier*, is removed when making the HTTP request, thus preventing a client to trying to access the real world object through HTTP [23, sec. 2.3.2].

RDF also allows for *blank nodes* to be used as subjects and objects. A blank node is a resource without a URI or a literal, it is an anonymous node. Blank nodes start with an underscore followed by a colon and the name of the variable `_:Order`. Blank nodes can also be written as a nested list, an example of this is seen in Code Snippet 6. The person Ben lives in a

```
ltpch:customer_ben ltpch:c_name "Ben" ;
                   ltpch:c_has_nation [ ltpch:n_has_region ltpch:region_europe ;
                                        ltpch:comment "A warm place" ] .
```

Code Snippet 6: Caption

country which is in the region europe, but the nation is a blank node and thus unknown to us. Blank nodes are often used to protect information or create complex attributes e.g. an address that consist of a street, streetnumber and house.

A.5 Ontology

Since predicates in a triple are also URIs it is possible to use them as subjects (or objects). This is useful to define the nature of the relationship which the predicate represent. When defining this nature we say that we define a *property* [35]. Overall, there are two classes of properties, those that are to be used as predicates with resources (URIs) as objects in triples and those which are to be used with literals as objects. These two broad classes of are called *object properties* and *data type properties*, respectively. If we look at Code Snippet 5, we see examples of these in the two first lines. `<predicate1>` is an object property, while `<predicate2>` is a data type property. The definition of these properties in Turtle format is shown in Code Snippet 7. Note that we use the shorthand notation `a` for defining the types, as described in Section A.3.

```
<predicate1> a owl:ObjectProperty .

<predicate2> a owl:DatatypeProperty .
```

Code Snippet 7: RDF in Turtle format describing two predicates of different types.

Aside from defining properties to be used in an RDF graph, ontologies can also define *classes* of resources. This is useful to group together resources of the same type and to help make more specific properties. E.g. we might want to have a class called `Company` which every company resource is supposed to belong to. This is shown in Code Snippet 8 in Turtle format. This ontology also defines the data type property `bus:name`, which is used to define names of companies.

```
bus:Company a owl:Class .

bus:name a            owl:DatatypeProperty ;
         rdfs:domain  bus:Company ;
         rdfs:range   xsd:string .
```

Code Snippet 8: RDF in Turtle format describing the type of a company. A "name" is defined as a DatatypeProperty type, with the domain of a company, and the range is a string.

An RDF ontology can be used to formalize different levels of understanding. Often terms such as vocabulary, taxonomy, and thesaurus are used to describe a collection of explicit listed concepts [31]. We use the term vocabulary when we describe the multidimensional metadata such as QB4OLAP. We are using the term ontology to describe the metadata of a dataset e.g. the TPCH dataset.

A.6 Querying Linked Open Data

Querying the Semantic Web is often done through a SPARQL Endpoint. Several such endpoints are hosted by different organizations [38]. The query language is called SPARQL [39] and uses syntax and keywords similar to those of SQL. An example query in SPARQL is shown in Code Snippet 9, where we retrieve every field in our dataset (called a *named graph*) `agri:FieldGraph`, with an area under 10 (hectare). The two first lines in our `WHERE`-clause are called *triple patterns*, and are similar to triples in Turtle format, with the difference that variables are allowed. Strictly speaking several triple patterns can be located on the same line so long they are separated by a dot (.). Variables are recognized by the prefixed question mark (?). The `FILTER`-keyword allows making restrictions such as limiting our result set to fields in a particular area range.

```
SELECT ?field ?area
FROM agri:FieldGraph
WHERE {
  ?field a       agri:Field .
  ?field ex:area ?area .
  FILTER(?area < 10)
}
```

Code Snippet 9: Example SPARQL query. All resources `?field` are selected together with their area from our graph `agri:FieldGraph`. Each resource must be a field and have an area under 10.

B. PROCESS

The process of converting TPC-H to cubes with different patterns consists of six steps. In Figure 10 this process is illustrated. We now explain each of the six steps.

Generate SF#. First we use the TPC-H DBGEN to generate TBL files, which contains the data in a compact format similar to comma separated data. The TPC-H DBGEN generates data according to the specified scaling factor, in Code Snippet 10 an example of TBL data is shown. Recall that we generate data with scaling factor 0.01, 0.1, and 1, a scaling factor of 1 correspond to 1 GB of TBL data.

```
1 1|Customer#000000001|IVhzIApeRb ot,c,E|15|25-989-741-2988|711.56|BUILDING|to the even,
      regular platelets. regular, ironic epitaphs nag e|
2 2|Customer#000000002|XSTf4,NCwDVaWNe6tEgvwfmRchLXak|13|23-768-687-3665|121.65|AUTOMOBILE|l
      accounts. blithely ironic theodolites integrate boldly: caref|
```

Code Snippet 10: Data from a TBL file with data regarding customers.

Conversion. In the conversion step we transform the TBL data files to RDF. Here we use the BIBM tool csv2ttl[12] to create to turtle RDF. Code Snippet 11 shows an example of how we invoke the tool. The csv2ttl tool requires a json file wich

[12] `http://sourceforge.net/p/bibm/code/HEAD/tree/trunk/bibm/csv2ttl.sh`

```
/bibm/csv2ttl.sh -schema /bibm/tpch/virtuoso/rdfh_schema.json -ext tbl /nation.tbl
```

Code Snippet 11: How we call the csv2ttl tool.

describes the RDF classes and how they relate to the input file. It is also required to specify format of the input.

Because of an error in the converter all dates are marked as `xsd:dateTime`, this is however incorrect because they do not contain a timestamp. We replace the type with the correct `xsd:date`. We you SED to this, in Code Snippet 12 the command is seen.

```
sed -i s/xsd:dateTime/xsd:date/
```

Code Snippet 12: Command to replace dateTime with date.

Stage. In order to load the data into the Sesame native store repository we utilize the sesame openrdf API. We create a new repository and load the turtle files. We call this repository for the stageing area because it contain without any multidimensional annotations.

The next three steps are perform for each cube, in our example it is the Lineitem, Orders, and Partsupplier.

QB4OLAP. This step consist of a manual phase where the data is analyzed, we create construct queries, and execute these. Based on which kind of queries that need to be answered we design cubes. Measures and dimensions are identified and defined according to QB4OLAP. The *cube definitions* for TPC-H are located in Appendix D.

We write a series of construct queries that create one repository for each cube described in the cube definition. In Appendix E we have included the construct queries for the Snowflake-pattern Lineitem-cube.

Last we execute these construct queries and the repositories are created. We measure the execution time of the construct queries.

SWOD-S. In this step we use SWOD to generate construct queries for Star-pattern as described in Section 6, but we do not count the generation time as part of the load time. The load time is how many seconds it takes to run the construct queries.

SWOD-D. Similar to SWOD-S we generate construct queries but for the Denormalized-pattern, this is also described in Section 6. We the load time in the same way.

C. EXTENSION FUNCTIONS

To make our queries more comprehensible, we have created some *extension functions* on our Sesame SPARQL server. These are: `dateadd`, `equals`, and `or`. These extension functions are located at the namespace `http://example.org/customfunction/`.

`dateadd` is used to add a duration to a date. We are using simple durations, to allow the function to be simple and efficient. A simple duration only has one unit, such as "21 days" or "3 months". The duration "1 year and 6 months" is not a simple duration. The function is used when a query requires results in a particular date interval, which is defined by a date and a simple duration. The unit of the duration is given as a string, the value is given as an integer, and the date, which is added a duration to, is an XML Schema date[13]. The extension function is implemented in Java, since Sesame is implemented in Java and is easily extended through it. We implement it as an interface to the built-in XML data types (date and duration) and rely on these to handle the underlying computation of adding a duration to a date.

The extension function `equals` is an equality test function, which returns 0 or 1 instead of true or false. It is used when aggregating measures of both an entire set and a selected subset. This is the case when e.g. calculating the market share percentage of a certain nation in query 8. The alternative to using `equals`, would be to have two separate sub-queries, one

[13] `http://www.w3.org/TR/xmlschema-2/#date`

aggregating the entire set and one for aggregating the subset, then finally calculating the market share. By using `equals`, we can have a single sub-query which calculate both the aggregate value of the entire set and selected subset by multiplying the aggregate value in the latter case with `equals`, with the selection condition as parameter.

In one query (query 12) the selection is a disjunctive equality, which we enable by the `or` extension function. We use results from the calls to `equals` as inputs, if either of the calls to `equals` returns 1, `or` will also return 1 otherwise it returns 0 (as would be expected). The results of `or` are summed together to get the count on the selection.

D. CUBE DEFINITIONS

```
#####################################################################
# Dimensions
#####################################################################
ltpch:ordersDim a qb:DimensionProperty .
ltpch:partSupplierDim a qb:DimensionProperty .

ltpch:l_has_order a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:ordersDim ;
  qb4o:parentLevel ltpch:o_has_customer ;
  qb4o:hasAttribute ltpch:o_orderkey ,
                    ltpch:o_custkey ,
                    ltpch:o_orderstatus ,
                    ltpch:o_totalprice ,
                    ltpch:o_orderdate ,
                    ltpch:o_orderpriority ,
                    ltpch:o_clerk ,
                    ltpch:o_shippriorit ,
                    ltpch:o_comment .

ltpch:o_has_customer a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:ordersDim ;
  qb4o:parentLevel ltpch:c_has_nation ;
  qb4o:hasAttribute ltpch:c_custkey ,
                    ltpch:c_name ,
                    ltpch:c_address ,
                    ltpch:c_nationkey ,
                    ltpch:c_phone ,
                    ltpch:c_acctbal ,
                    ltpch:c_mktsegment ,
                    ltpch:c_comment .

ltpch:c_has_nation a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:ordersDim ;
  qb4o:parentLevel ltpch:n_has_region ;
  qb4o:hasAttribute ltpch:n_nationkey ,
                    ltpch:n_name ,
                    ltpch:n_regionkey ,
                    ltpch:n_comment .

ltpch:s_has_nation a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:partSupplierDim ;
  qb4o:parentLevel ltpch:n_has_region ;
  qb4o:hasAttribute ltpch:n_nationkey ,
                    ltpch:n_name ,
                    ltpch:n_regionkey ,
                    ltpch:n_comment .

ltpch:n_has_region a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:ordersDim ,
                   ltpch:partSupplierDim ;
  qb4o:hasAttribute ltpch:r_regionkey ,
                    ltpch:r_name ,
                    ltpch:r_comment .

ltpch:l_has_partsupplier a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:partSupplierDim ;
  qb4o:parentLevel ltpch:ps_has_part ,
                   ltpch:ps_has_supplier ;
  qb4o:hasAttribute ltpch:ps_partkey ,
                    ltpch:ps_suppkey ,
```

```
                    ltpch:ps_availqty ,
                    ltpch:ps_supplycost ,
                    ltpch:ps_comment .

ltpch:ps_has_part a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:partSupplierDim ;
  qb4o:hasAttribute ltpch:p_partkey ,
                    ltpch:p_name ,
                    ltpch:p_mfgr ,
                    ltpch:p_brand ,
                    ltpch:p_type ,
                    ltpch:p_size ,
                    ltpch:p_container ,
                    ltpch:p_retailprice ,
                    ltpch:p_comment .

ltpch:ps_has_supplier a qb4o:LevelProperty ;
  qb4o:inDimension ltpch:partSupplierDim ;
  qb4o:parentLevel ltpch:s_has_nation ;
  qb4o:hasAttribute ltpch:s_suppkey ,
                    ltpch:s_name ,
                    ltpch:s_address ,
                    ltpch:s_nationkey ,
                    ltpch:s_phone ,
                    ltpch:s_acctbal ,
                    ltpch:s_comment .

#####################################################################
# Line Item Data Cube
#####################################################################
ltpch:lineitemStructure a qb:DataStructureDefinition ;
  qb:component [ qb4o:level ltpch:l_has_order ; qb:order 1 ] ;
  qb:component [ qb4o:level ltpch:l_has_partsupplier ; qb:order 2 ] ;
  qb:component [ qb:measure ltpch:l_quantity ; qb:hasAggregateFunction qb4o:sum ] ;
  qb:component [ qb:measure ltpch:l_extendedprice ;
                 qb4o:hasAggregateFunction qb4o:sum ] ;
  qb:component [ qb:measure ltpch:l_discount ; qb4o:hasAggregateFunction qb4o:avg ] ;
  qb:component [ qb:measure ltpch:l_tax ; qb4o:hasAggregateFunction qb4o:avg ] .

ltpch:lineitemCube a qb:DataSet ;
  qb:structure ltpch:lineitemStructure ;
  rdf:label "Lineitem Cube" .

#####################################################################
# Part Supplier Data Cube
#####################################################################
ltpch:partSupplierStructure a qb:DataStructureDefinition ;
  qb:component [ qb4o:level ltpch:ps_has_part ; qb:order 1 ] ;
  qb:component [ qb4o:level ltpch:ps_has_supplier ; qb:order 2 ] ;
  qb:component [ qb:measure ltpch:ps_availqty ;
                 qb4o:hasAggregateFunction qb4o:sum ] ;
  qb:component [ qb:measure ltpch:sp_supplycost ;
                 qb4o:hasAggregateFunction qb4o:sum ] .

ltpch:partSupplierCube a qb:DataSet ;
  qb:structure ltpch:partSupplierStructure ;
  rdf:label "Part Supplier Cube" .

#####################################################################
# Orders Data Cube
#####################################################################
ltpch:ordersStructure a qb:DataStructureDefinition ;
  qb:component [ qb4o:level ltpch:o_has_customer ; qb:order 1 ] ;
  qb:component [ qb:measure ltpch:o_totalprice ;
                 qb4o:hasAggregateFunction qb4o:sum ] .

ltpch:ordersCube a qb:DataSet ;
  qb:structure ltpch:ordersStructure ;
  rdf:label "Orders Cube" .
```

Code Snippet 13: Part of ontology defining the data structure for the Lineitem-cube.

E. CONSTRUCT QUERIES FOR SNOWFLAKE LINEITEM CUBE

In this section we show the construct queries for the Lineitem-cube. We made these queries manually, we also produced queries for the Orders and Partsupplier-cubes.

```
construct
{
    ?li a qb:Observation .
    ?li ?prop ?obj .
    ?li qb:dataSet ltpch:lineitemCube .
    ?li ltpch:l_has_partsupplier ?partsupp .
}
where
{
    ?li a ltpch:lineitem .
    ?li ?prop ?obj .
    ?li ltpch:l_has_supplier ?supp .
    ?li ltpch:l_has_part ?part .
    BIND(URI(CONCAT("http://lod2.eu/schemas/ltpch-inst#partsupp",SUBSTR(STR(?part),38),
        SUBSTR(STR(?supp),42))) as ?partsupp) .
}
```

Code Snippet 14: Construct query for Snowflake-pattern that use the lineitem class

```
construct
{
    ?order a qb4o:LevelMember .
    ?order ?prop ?obj .
    ?order qb4o:inLevel ltpch:l_has_order .
    ?order skos:broader ?cu .
}
where
{
    ?order a ltpch:orders .
    ?order ?prop ?obj .
    ?order ltpch:o_has_customer ?cu .
}
```

Code Snippet 15: Construct query for Snowflake-pattern that use the orders class

```
construct
{
    ?cu a qb4o:LevelMember .
    ?cu ?prop ?obj .
    ?cu qb4o:inLevel ltpch:o_has_customer .
    ?cu skos:broader ?na .
}
where
{
    ?cu a ltpch:customer .
    ?cu ?prop ?obj .
    ?cu ltpch:c_has_nation ?na .
}
```

Code Snippet 16: Construct query for Snowflake-pattern that use the customer class

```
construct
{
    ?nation a qb4o:LevelMember .
    ?nation ?prop ?obj .
    ?nation qb4o:inLevel ltpch:c_has_nation ,
                         ltpch:s_has_nation .
    ?nation skos:broader ?re .
}
where
```

```
{
    ?nation a ltpch:nation .
    ?nation ?prop ?obj .
    ?nation ltpch:n_has_region ?re .
}
```

Code Snippet 17: Construct query for Snowflake-pattern that use the nation class

```
construct
{
    ?region a qb4o:LevelMember .
    ?region ?prop ?obj .
    ?region qb4o:inLevel ltpch:n_has_region .
}
where
{
    ?region a ltpch:region .
    ?region ?prop ?obj .
}
```

Code Snippet 18: Construct query for Snowflake-pattern that use the region class

```
construct
{
    ?partsupp a qb4o:LevelMember .
    ?partsupp ?prop ?obj .
    ?partsupp qb4o:inLevel ltpch:l_has_partsupplier .
    ?partsupp skos:broader ?part .
    ?partsupp skos:broader ?supplier .
}
where
{
    ?partsupp a ltpch:partsupp .
    ?partsupp ?prop ?obj .
    ?partsupp ltpch:ps_has_part ?part .
    ?partsupp ltpch:ps_has_supplier ?supplier .
}
```

Code Snippet 19: Construct query for Snowflake-pattern that use the partsupplier class

```
construct
{
    ?part a qb4o:LevelMember .
    ?part ?prop ?obj .
    ?part qb4o:inLevel ltpch:ps_has_part .
}
where
{
    ?part a ltpch:part .
    ?part ?prop ?obj .
}
```

Code Snippet 20: Construct query for Snowflake-pattern that use the part class

```
construct
{
    ?supp a qb4o:LevelMember .
    ?supp ?prop ?obj .
    ?supp qb4o:inLevel ltpch:ps_has_supplier .
    ?supp skos:broader ?na .
}
where
{
    ?supp a ltpch:supplier .
    ?supp ?prop ?obj .
    ?supp ltpch:s_has_nation ?na .
}
```

Code Snippet 21: Construct query for Snowflake-pattern that use the supplier class

F. ADDITIONAL EVALUATION RESULTS

This section contains additional results and graphs.

Pattern	0.001	0.01	0.1
snowflake	2	19	193
star	2	19	189
denormalized	3	33	329

Figure 20: The disk size of repositories containing the Orders-cube using different patterns. The sizes are in mega bytes.

Pattern	0.001	0.01	0.1
snowflake	1	9	91
star	1	8	80
denormalized	2	19	187

Figure 21: The disk size of repositories containing the Partsupplier-cube using different patterns. The sizes are in mega bytes.

Pattern	0.001	0.01	0.1
snowflake	20,025	198,225	1,980,225
star	20,100	201,000	2,010,000
denormalized	39,700	397,000	3,970,000

Figure 22: The triples of repositories containing the Orders-cube using different patterns.

Pattern	0.001	0.01	0.1
snowflake	8,635	89,325	891,225
star	8,126	85,300	853,000
denormalized	22,096	248,000	2,480,000

Figure 23: The triples of repositories containing the Partsupplier-cube using different patterns.

Step	0.001	0.01	0.1
Generate	9	9	12
Convert	3	8	58
Stage	13	49	410
QB4OLAP	5.58	12.41	74.24
SWOD-S	5.47	16.89	140.05
SWOD-D	3.76	7.65	34.83

Figure 24: Seconds spent for performing the steps of our process in the Orders-cube.

Step	0.001	0.01	0.1
Generate	9	9	12
Convert	3	8	58
Stage	13	49	410
QB4OLAP	4.29	11.13	73.65
SWOD-S	5.64	10.07	38.2
SWOD-D	4.34	11.26	74.83

Figure 25: Seconds spent for performing the steps of our process in the Partsupplier-cube.

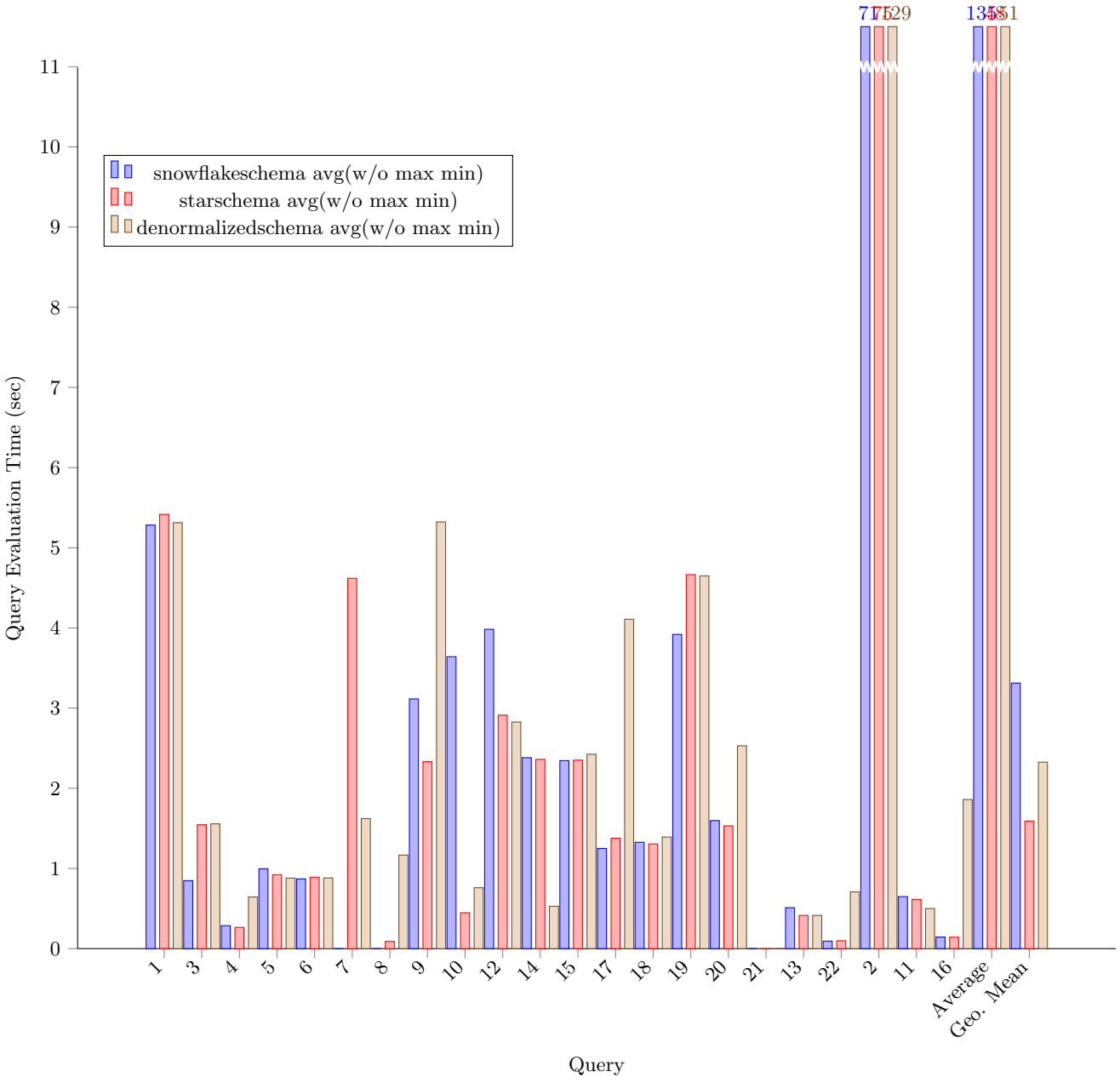


Figure 26: Query Evaluation Times of queries on our different patterns on scaling factor 0.01.

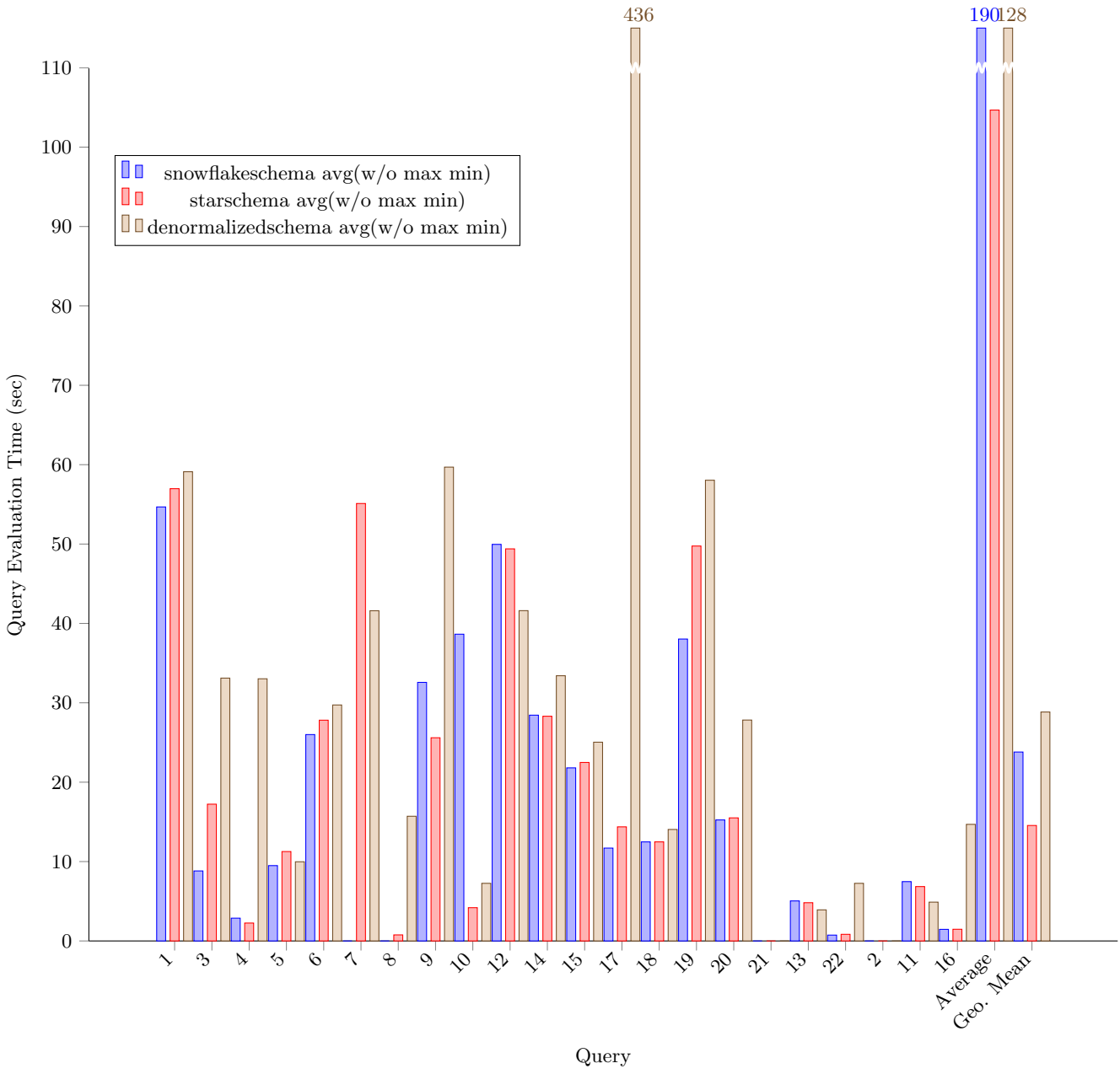


Figure 27: Query Evaluation Times of queries on our different patterns on scaling factor 0.1.

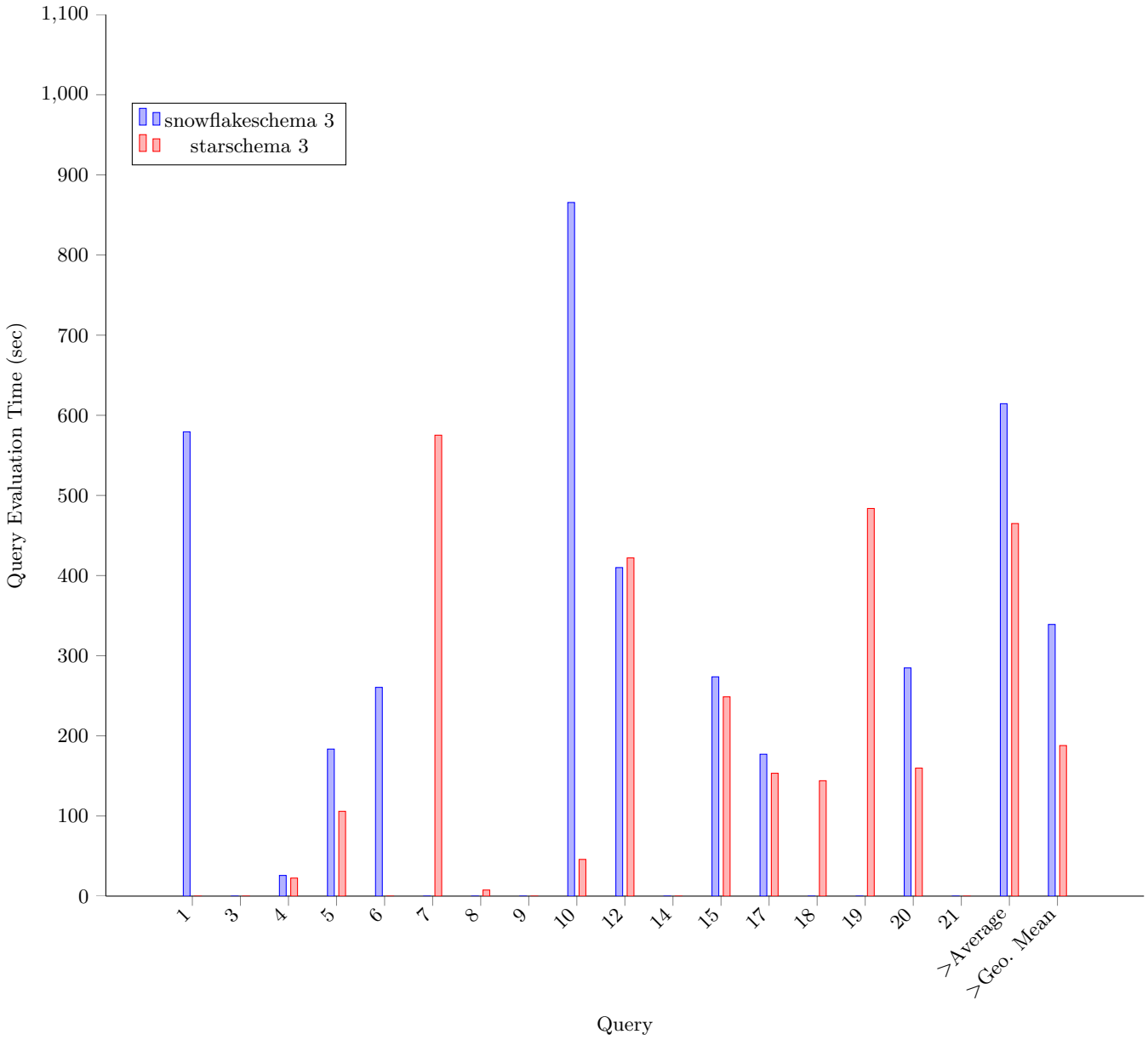


Figure 28: Query Evaluation Times of queries on our different patterns on scaling factor 1.

G. TPCH

This section contains the 22 queries that we run on the TPC-H dataset. Each query consists of a title, description, and business question, which are copied from the TPC-H documentation[14]. Each query is written three times such that there is a query for each pattern. Relevant information from the query description file is also included, these originates from BIBM [15]. This describe the substitution parameter that are inserted into the queries. Because we are not concerned with the TPC-H ACID tests, we remove the update elements from the query description of query 15. This section is meant as a reference work.

G.1 Pricing Summary Report Query (Q1)

This query reports the amount of business that was billed, shipped, and returned.

> The Pricing Summary Report Query provides a summary pricing report for all lineitems shipped as of a given date. The date is within 60 - 120 days of the greatest ship date contained in the database. The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of lineitems in each group is included.

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix bif: <http://example.org/customfunction/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?l_returnflag
  ?l_linestatus
  (sum(?l_linequantity) as ?sum_qty)
  (sum(?l_lineextendedprice) as ?sum_base_price)
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)) as ?sum_disc_price)
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)*(1 + ?l_linetax)) as ?sum_charge)
  (avg(?l_linequantity) as ?avg_qty)
  (avg(?l_lineextendedprice) as ?avg_price)
  (avg(?l_linediscount) as ?avg_disc)
  (count(1) as ?count_order)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_returnflag ?l_returnflag ;
       ltpch:l_linestatus ?l_linestatus ;
       ltpch:l_linequantity ?l_linequantity ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linetax ?l_linetax ;
       ltpch:l_shipdate ?l_shipdate ;
       ltpch:l_linediscount ?l_linediscount .
    filter (?l_shipdate <= bif:dateadd ("day", -%DELTA%, "1998-12-01"^^xsd:date))
}
group by
  ?l_returnflag
  ?l_linestatus
order by
  ?l_returnflag
  ?l_linestatus
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?l_returnflag
  ?l_linestatus
  (sum(?l_linequantity) as ?sum_qty)
  (sum(?l_lineextendedprice) as ?sum_base_price)
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)) as ?sum_disc_price)
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)*(1 + ?l_linetax)) as ?sum_charge)
  (avg(?l_linequantity) as ?avg_qty)
  (avg(?l_lineextendedprice) as ?avg_price)
```

[14] http://www.tpc.org/tpch/spec/tpch2.16.0.pdf
[15] http://sourceforge.net/p/bibm/code/HEAD/tree/trunk/bibm/tpch/querydescriptions/

```
  (avg(?l_linediscount) as ?avg_disc)
  (count(1) as ?count_order)
where {
    ?l a ltpch:lineitem ;
       ltpch:l_returnflag ?l_returnflag ;
       ltpch:l_linestatus ?l_linestatus ;
       ltpch:l_linequantity ?l_linequantity ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linetax ?l_linetax ;
       ltpch:l_shipdate ?l_shipdate ;
       ltpch:l_linediscount ?l_linediscount .
    filter (?l_shipdate <= bif:dateadd ("day", -%DELTA%, "1998-12-01"^^xsd:date))
}
group by
  ?l_returnflag
  ?l_linestatus
order by
  ?l_returnflag
  ?l_linestatus
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?l_returnflag
  ?l_linestatus
  (sum(?l_linequantity) as ?sum_qty)
  (sum(?l_lineextendedprice) as ?sum_base_price)
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)) as ?sum_disc_price)
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)*(1 + ?l_linetax)) as ?sum_charge)
  (avg(?l_linequantity) as ?avg_qty)
  (avg(?l_lineextendedprice) as ?avg_price)
  (avg(?l_linediscount) as ?avg_disc)
  (count(1) as ?count_order)
where {
    ?l qb:dataSet ltpch:lineitemCube  ;
       ltpch:l_returnflag ?l_returnflag ;
       ltpch:l_linestatus ?l_linestatus ;
       ltpch:l_linequantity ?l_linequantity ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linetax ?l_linetax ;
       ltpch:l_shipdate ?l_shipdate ;
       ltpch:l_linediscount ?l_linediscount .
    filter (?l_shipdate <= bif:dateadd ("day", -%DELTA%, "1998-12-01"^^xsd:date))
}
group by
  ?l_returnflag
  ?l_linestatus
order by
  ?l_returnflag
  ?l_linestatus
```

```
params:[
  {name:DELTA, class:Random, range:[60,120], default:90}
 ]
```

Code Snippet 22: Query ones variable

G.2 Minimum Cost Supplier Query (Q2)

This query finds which supplier should be selected to place an order for a given part in a given region.

> The Minimum Cost Supplier Query finds, in a given region, for each part of a certain type and size, the supplier who can supply it at minimum cost. If several suppliers in that region offer the desired part type and size at the same (minimum) cost, the query lists the parts from suppliers with the 100 highest account balances. For each

supplier, the query lists the supplier's account balance, name and nation; the part's number and manufacturer; the supplier's address, phone number and comment information.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?s_acctbal
  ?s_name
  ?nation_name
  ?p_partkey
  ?p_mfgr
  ?s_address
  ?s_phone
  ?s_comment
where
{
  ?ps qb:dataSet ltpch:partSupplierCube ;
      ltpch:ps_has_supplier ?supp;
      ltpch:ps_has_part ?part ;
      ltpch:ps_supplycost ?minsc .
  ?supp ltpch:s_acctbal ?s_acctbal ;
        ltpch:s_name ?s_name ;
        ltpch:s_has_nation ?s_nation ;
        ltpch:s_address ?s_address ;
        ltpch:s_phone ?s_phone ;
        ltpch:s_comment ?s_comment .
  ?s_nation ltpch:n_name ?nation_name ;
            ltpch:n_has_region ?s_region .
  ?part ltpch:p_partkey ?p_partkey ;
        ltpch:p_mfgr ?p_mfgr ;
        ltpch:p_size ?p_size ;
        ltpch:p_type ?p_type .
  filter (REGEX(?p_type, "%TYPE%$" ) && ?p_size = %SIZE%) .
  {
    select
      ?part
      (?m_region as ?s_region)
      (min(?s_cost) as ?minsc)
    where
    {
        ?ps a ltpch:partsupp ;
            ltpch:ps_has_part ?part ;
            ltpch:ps_has_supplier ?minsupp ;
            ltpch:ps_supplycost ?s_cost .
        ?minsupp ltpch:s_has_nation ?m_nation .
        ?m_nation ltpch:n_has_region ?m_region .
        ?m_region ltpch:r_name ?m_region_name .
        filter (?m_region_name = "%REGION%") .
    }
    group by
      ?part
      ?m_region
  }
}
order by
  desc (?s_acctbal)
  ?nation_name
  ?s_name
  ?p_partkey
limit 100
```

```
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>

select
  ?s_acctbal
  ?s_name
```

```
  ?nation_name
  ?p_partkey
  ?p_mfgr
  ?s_address
  ?s_phone
  ?s_comment
where {
  ?ps qb:dataSet ltpch:partSupplierCube ;
      ltpch:ps_has_supplier ?supp;
      ltpch:ps_has_part ?part ;
      ltpch:ps_supplycost ?minsc .
  ?supp ltpch:supplier_acctbal ?s_acctbal ;
          ltpch:supplier_name ?s_name ;
        ltpch:supplier_address ?s_address ;
        ltpch:supplier_phone ?s_phone ;
        ltpch:supplier_comment ?s_comment ;
        ltpch:nation_name ?nation_name ;
        ltpch:region_name ?r_name .
  ?part ltpch:part_partkey ?p_partkey ;
        ltpch:part_mfgr ?p_mfgr ;
        ltpch:part_size ?p_size ;
        ltpch:part_type ?p_type .
  filter (REGEX(?p_type, "%TYPE%$") && ?p_size = %SIZE%) .
  {
    select
      ?part
      ?r_name
      (min(?s_cost) as ?minsc)
    where
    {
      ?ps2 qb:dataSet ltpch:partSupplierCube ;
          ltpch:ps_has_part ?part;
          ltpch:ps_has_supplier ?ms;
          ltpch:ps_supplycost ?s_cost .
      ?ms ltpch:region_name ?r_name .
      FILTER(?r_name = "%REGION%")
    }
    group by
      ?part
      ?r_name
  } .
}
order by
  desc (?s_acctbal)
  ?nation_name
  ?s_name
  ?p_partkey
limit 100
```

```
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>

select
  ?s_acctbal
  ?s_name
  ?nation_name
  ?p_partkey
  ?p_mfgr
  ?s_address
  ?s_phone
  ?s_comment
where
{
  ?ps qb:dataSet ltpch:partSupplierCube ;
      ltpch:ps_supplycost ?minsc ;
      ltpch:supplier_supplier_acctbal ?s_acctbal ;
      ltpch:supplier_supplier_name ?s_name ;
      ltpch:supplier_supplier_address ?s_address ;
      ltpch:supplier_supplier_phone ?s_phone ;
      ltpch:supplier_supplier_comment ?s_comment ;
```

```
      ltpch:supplier_nation_name ?nation_name ;
      ltpch:supplier_region_name ?r_name ;
      ltpch:part_part_partkey ?p_partkey ;
      ltpch:part_part_mfgr ?p_mfgr ;
      ltpch:part_part_size ?p_size ;
      ltpch:part_part_type ?p_type .
  filter (REGEX(?p_type, "%TYPE%$") && ?p_size = %SIZE%) .
  {
    select
      ?p_partkey
      ?r_name
      (min(?s_cost) as ?minsc)
    where
    {
      ?ps2 qb:dataSet ltpch:partSupplierCube ;
          ltpch:part_part_partkey ?p_partkey ;
          ltpch:ps_supplycost ?s_cost ;
          ltpch:supplier_region_name ?r_name .
      FILTER(?r_name = "%REGION%") .
    }
    group by
      ?p_partkey
      ?r_name
  } .
}
order by
  desc (?s_acctbal)
  ?nation_name
  ?s_name
  ?p_partkey
limit 100
```

```
params:[
  {name:SIZE, class:Random, range:[1,50], default:15},
  {name:TYPE, class:Type, range:[3], default:'BRASS'},
  {name:REGION, class:Region, default:'EUROPE'}
 ],
```

Code Snippet 23: Query ones variable

G.3 Shipping Priority Query (Q3)

This query retrieves the 10 unshipped orders with the highest value.

> The Shipping Priority Query retrieves the shipping priority and potential revenue, defined as the sum of l_extendedprice * (1-l_discount), of the orders having the largest revenue among those that had not been shipped as of a given date. Orders are listed in decreasing order of revenue. If more than 10 unshipped orders exist, only the 10 orders with the largest revenue are listed.

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?o_orderkey
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)) as ?revenue)
  ?o_orderdate
  ?o_shippriority
where  {
  ?li qb:dataSet ltpch:lineitemCube ;
      ltpch:l_lineextendedprice ?l_lineextendedprice ;
    ltpch:l_linediscount ?l_linediscount ;
    ltpch:l_has_order ?ord ;
    ltpch:l_shipdate ?l_shipdate .
  ?ord ltpch:o_orderdate ?o_orderdate ;
    ltpch:o_shippriority ?o_shippriority ;
    ltpch:o_orderkey ?o_orderkey ;
    ltpch:o_has_customer ?cust .
  ?cust ltpch:c_mktsegment ?c_mktsegment .
```

```
  filter ((?o_orderdate < "%DATE%"^^xsd:date) &&
    (?l_shipdate > "%DATE%"^^xsd:date) &&
    (?c_mktsegment = "%SEGMENT%") )
}
group by
  ?o_orderkey
  ?o_orderdate
  ?o_shippriority
order by
  desc (sum (?l_lineextendedprice * (1 - ?l_linediscount)))
  ?o_orderdate
limit 10
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?o_orderkey
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)) as ?revenue)
  ?o_orderdate
  ?o_shippriority
where  {
  ?li qb:dataSet ltpch:lineitemCube ;
      ltpch:l_lineextendedprice ?l_lineextendedprice ;
    ltpch:l_linediscount ?l_linediscount ;
    ltpch:l_has_order ?ord ;
    ltpch:l_shipdate ?l_shipdate .
  ?ord ltpch:order_orderdate ?o_orderdate ;
    ltpch:order_shippriority ?o_shippriority ;
    ltpch:order_orderkey ?o_orderkey ;
    ltpch:customer_mktsegment ?c_mktsegment .
  filter ((?o_orderdate < "%DATE%"^^xsd:date) &&
    (?l_shipdate > "%DATE%"^^xsd:date) &&
    (?c_mktsegment = "%SEGMENT%") )
}
group by
  ?o_orderkey
  ?o_orderdate
  ?o_shippriority
order by
  desc (sum (?l_lineextendedprice * (1 - ?l_linediscount)))
  ?o_orderdate
limit 10
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?o_orderkey
  (sum(?l_lineextendedprice*(1 - ?l_linediscount)) as ?revenue)
  ?o_orderdate
  ?o_shippriority
where  {
  ?li qb:dataSet ltpch:lineitemCube ;
      ltpch:l_lineextendedprice ?l_lineextendedprice ;
    ltpch:l_linediscount ?l_linediscount ;
    ltpch:l_shipdate ?l_shipdate ;
    ltpch:order_order_orderdate ?o_orderdate ;
    ltpch:order_order_shippriority ?o_shippriority ;
    ltpch:order_order_orderkey ?o_orderkey ;
    ltpch:order_customer_mktsegment ?c_mktsegment .
  filter ((?o_orderdate < "%DATE%"^^xsd:date) &&
    (?l_shipdate > "%DATE%"^^xsd:date) &&
    (?c_mktsegment = "%SEGMENT%") )
}
```

```
group by
  ?o_orderkey
  ?o_orderdate
  ?o_shippriority
order by
  desc (sum (?l_lineextendedprice * (1 - ?l_linediscount)))
  ?o_orderdate
limit 10
```

```
params:[
  {name:SEGMENT,
   class:OneOf,
   range:[AUTOMOBILE, BUILDING, FURNITURE, MACHINERY, HOUSEHOLD],
   default: BUILDING
  },
```

Code Snippet 24: Query ones variable

G.4 Order Priority Checking Query (Q4)

This query determines how well the order priority system is working and gives an assessment of customer satisfac- tion.

> The Order Priority Checking Query counts the number of orders ordered in a given quarter of a given year in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority sorted in ascending priority order.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?o_orderpriority
  (count(*) as ?order_count)
where
{
  {
    select distinct
      ?o_orderpriority
      ?ord
    where
    {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_has_order ?ord ;
          ltpch:l_commitdate ?l_commitdate ;
          ltpch:l_receiptdate ?l_receiptdate .
      ?ord ltpch:o_orderpriority ?o_orderpriority ;
           ltpch:o_orderdate ?o_orderdate .
      filter (
        (?l_commitdate < ?l_receiptdate) &&
        (?o_orderdate >= "%MONTH%-01"^^xsd:date) &&
        (?o_orderdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date))
      )
    }
  }
}
group by
  ?o_orderpriority
order by
  ?o_orderpriority
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
```

```
  ?o_orderpriority
  (count(1) as ?order_count)
where
{
  {
    select distinct
      ?o_orderpriority
      ?ord
    where
    {
      ?li ltpch:l_has_order ?ord ;
          ltpch:l_commitdate ?l_commitdate ;
          ltpch:l_receiptdate ?l_receiptdate .
      ?ord ltpch:order_orderpriority ?o_orderpriority ;
           ltpch:order_orderdate ?o_orderdate .
      filter (
        (?l_commitdate < ?l_receiptdate) &&
        (?o_orderdate >= "%MONTH%-01"^^xsd:date) &&
        (?o_orderdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date))
      )
    }
  }
}
group by
  ?o_orderpriority
order by
  ?o_orderpriority
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?o_orderpriority
  (count(1) as ?order_count)
where
{
  {
    select distinct
      ?o_orderpriority
      ?orderkey
    where
    {
      ?li ltpch:order_order_orderpriority ?o_orderpriority ;
           ltpch:order_order_orderdate ?o_orderdate ;
           ltpch:order_order_orderkey ?orderkey ;
           ltpch:l_commitdate ?l_commitdate ;
           ltpch:l_receiptdate ?l_receiptdate .
      filter (
        (?l_commitdate < ?l_receiptdate) &&
        (?o_orderdate >= "%MONTH%-01"^^xsd:date) &&
        (?o_orderdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date))
      )
    }
  }
}
group by
  ?o_orderpriority
order by
  ?o_orderpriority
```

```
params:[
  {name:MONTH,
   class:Month,
   range:[1993-01, 1997-10],
   default: 1993-07
   }
 ],
```

Code Snippet 25: Query ones variable

G.5 Local Supplier Volume Query (Q5)

This query lists the revenue volume done through local suppliers.

> The Local Supplier Volume Query lists for each nation in a region the revenue volume that resulted from lineitem transactions in which the customer ordering parts and the supplier filling them were both within that nation. The query is run in order to determine whether to institute local distribution centers in a given region. The query considers only parts ordered in a given year. The query displays the nations and revenue volume in descending order by revenue. Revenue volume for all qualifying lineitems in a particular nation is defined as sum(l_extendedprice * (1 - l_discount)).

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?nation
  (sum(?l_lineextendedprice * (1 - ?l_linediscount)) as ?revenue)
where  {
   ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_has_order ?ord ;
       ltpch:l_has_partsupplier ?ps ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linediscount ?l_linediscount .
    ?ord ltpch:o_has_customer ?cust ;
         ltpch:o_orderdate ?o_orderdate .
    ?ps ltpch:ps_has_supplier ?supp .
      ?supp ltpch:s_has_nation ?s_nation .
    ?s_nation ltpch:n_has_region ?s_region ;
              ltpch:n_name ?nation .
    ?s_region ltpch:r_name ?r_name .
    ?cust ltpch:c_has_nation ?c_nation.
    filter ((?c_nation = ?s_nation) &&
      (?o_orderdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?o_orderdate < bif:dateadd ("year", 1,"%YEAR%-01-01" ^^xsd:date)) &&
      (?r_name = "%REGION%") )
  }
group by
  ?nation
order by
  desc (sum(?l_lineextendedprice * (1 - ?l_linediscount)))
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?nation
  (sum(?l_lineextendedprice * (1 - ?l_linediscount)) as ?revenue)
where  {
   ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_has_order ?ord ;
       ltpch:l_has_partsupplier ?ps ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linediscount ?l_linediscount .
    ?ord ltpch:order_orderdate ?o_orderdate ;
```

```
      ltpch:nation_name ?c_nation_name.
    ?ps ltpch:nation_name ?nation ;
      ltpch:region_name ?r_name .
    filter ((?c_nation_name = ?nation) &&
      (?o_orderdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?o_orderdate < bif:dateadd ("year", 1,"%YEAR%-01-01" ^^xsd:date)) &&
      (?r_name = "%REGION%") )
  }
group by
  ?nation
order by
  desc (sum(?l_lineextendedprice * (1 - ?l_linediscount)))
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?nation
  (sum(?l_lineextendedprice * (1 - ?l_linediscount)) as ?revenue)
where  {
   ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linediscount ?l_linediscount ;
       ltpch:order_order_orderdate ?o_orderdate ;
       ltpch:order_nation_name ?c_nation_name ;
       ltpch:partsupplier_nation_name ?nation ;
       ltpch:partsupplier_region_name ?r_name .
    filter ((?c_nation_name = ?nation) &&
      (?o_orderdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?o_orderdate < bif:dateadd ("year", 1,"%YEAR%-01-01" ^^xsd:date)) &&
      (?r_name = "%REGION%") )
  }
group by
  ?nation
order by
  desc (sum(?l_lineextendedprice * (1 - ?l_linediscount)))
```

```
1  params:[
2   {name:REGION,
3    class:Region,
4    default:'ASIA'
5   },
6   {name:YEAR,
7    class:Random,
8    range:[1993, 1997],
9    default:1994
10  }
11 ],
```

Code Snippet 26: Query ones variable

G.6 Forecasting Revenue Change Query (Q6)

This query quantifies the amount of revenue increase that would have resulted from eliminating certain company- wide discounts in a given percentage range in a given year. Asking this type of "what if" query can be used to look for ways to increase revenues.

> The Forecasting Revenue Change Query considers all the lineitems shipped in a given year with discounts between DISCOUNT-0.01 and DISCOUNT+0.01. The query lists the amount by which the total revenue would have increased if these discounts had been eliminated for lineitems with l_quantity less than quantity. Note that the potential revenue increase is equal to the sum of [l_extendedprice * l_discount] for all lineitems with discounts and quantities in the qualifying range.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  (sum(?l_lineextendedprice * ?l_linediscount) as ?revenue)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linediscount ?l_linediscount ;
       ltpch:l_linequantity ?l_linequantity ;
       ltpch:l_shipdate ?l_shipdate .
    filter ( (?l_shipdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?l_shipdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date)) &&
      (?l_linediscount >= %DISCOUNT% - 0.01) &&
      (?l_linediscount <= %DISCOUNT% + 0.01) &&
      (?l_linequantity < %QUANTITY%) )
}
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  (sum(?l_lineextendedprice * ?l_linediscount) as ?revenue)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linediscount ?l_linediscount ;
       ltpch:l_linequantity ?l_linequantity ;
       ltpch:l_shipdate ?l_shipdate .
    filter ( (?l_shipdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?l_shipdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date)) &&
      (?l_linediscount >= %DISCOUNT% - 0.01) &&
      (?l_linediscount <= %DISCOUNT% + 0.01) &&
      (?l_linequantity < %QUANTITY%) )
}
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  (sum(?l_lineextendedprice * ?l_linediscount) as ?revenue)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_lineextendedprice ?l_lineextendedprice ;
       ltpch:l_linediscount ?l_linediscount ;
       ltpch:l_linequantity ?l_linequantity ;
       ltpch:l_shipdate ?l_shipdate .
    filter ( (?l_shipdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?l_shipdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date)) &&
      (?l_linediscount >= %DISCOUNT% - 0.01) &&
      (?l_linediscount <= %DISCOUNT% + 0.01) &&
      (?l_linequantity < %QUANTITY%) )
}
```

```
params:[
 {name:YEAR,
  class:Random,
  range:[1993, 1997],
  default: 1994
 },
 {name:DISCOUNT,
  class:Random,
  range:[0.02, 0.09],
  default:0.06
 },
 {name:QUANTITY,
  class:Random,
  range:[24, 25],
  default: 24
 }
],
```

G.7 Volume Shipping Query (Q7)

This query determines the value of goods shipped between certain nations to help in the re-negotiation of shipping contracts.

> The Volume Shipping Query finds, for two given nations, the gross discounted revenues derived from lineitems in which parts were shipped from a supplier in either nation to a customer in the other nation during 1995 and 1996. The query lists the supplier nation, the customer nation, the year, and the revenue from shipments that took place in that year. The query orders the answer by Supplier nation, Customer nation, and year (all ascending).

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?supp_nation
  ?cust_nation
  ?li_year
  (sum (?volume) as ?revenue)
where {
  {
    select
      ?supp_nation
      ?cust_nation
      ((YEAR (?l_shipdate)) as ?li_year)
      ((?l_lineextendedprice * (1 - ?l_linediscount)) as ?volume)
    where {
      ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_has_order ?ord ;
        ltpch:l_has_partsupplier ?ps ;
        ltpch:l_shipdate ?l_shipdate ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount .
      ?ord ltpch:o_has_customer ?cust .
      ?cust ltpch:c_has_nation ?custn .
      ?custn ltpch:n_name ?cust_nation .
      ?ps ltpch:ps_has_supplier ?supp .
      ?supp ltpch:s_has_nation ?suppn .
      ?suppn ltpch:n_name ?supp_nation .
      filter ((
        (?cust_nation = "%NATION1%" && ?supp_nation = "%NATION2%") ||
        (?cust_nation = "%NATION2%" && ?supp_nation = "%NATION1%") ) &&
        (?l_shipdate >= "1995-01-01"^^xsd:date) &&
        (?l_shipdate <= "1996-12-31"^^xsd:date) )
      }
   }
}
group by
  ?supp_nation
  ?cust_nation
  ?li_year
order by
```

```
  ?supp_nation
  ?cust_nation
  ?li_year
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select ?supp_nation
?cust_nation
?li_year
(sum (?value) as ?revenue)
where {
  {
    select
      ?supp_nation
      ?cust_nation
      ((YEAR (?l_shipdate)) as ?li_year)
      ((?l_lineextendedprice * (1 - ?l_linediscount)) as ?value)
    where {
      ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_has_order ?ord ;
        ltpch:l_has_partsupplier ?ps ;
        ltpch:l_shipdate ?l_shipdate ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount .
      ?ord ltpch:nation_name ?cust_nation .
      ?ps ltpch:nation_name ?supp_nation .
      filter ((
        (?cust_nation = "%NATION1%" && ?supp_nation = "%NATION2%") ||
        (?cust_nation = "%NATION2%" && ?supp_nation = "%NATION1%") ) &&
        (?l_shipdate >= "1995-01-01"^^xsd:date) &&
        (?l_shipdate <= "1996-12-31"^^xsd:date) )
      }
   }
}
group by
  ?supp_nation
  ?cust_nation
  ?li_year
order by
  ?supp_nation
  ?cust_nation
  ?li_year
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select ?supp_nation
?cust_nation
?li_year
(sum (?value) as ?revenue)
where {
  {
    select
      ?supp_nation
      ?cust_nation
      ((YEAR (?l_shipdate)) as ?li_year)
      ((?l_lineextendedprice * (1 - ?l_linediscount)) as ?value)
    where {
      ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_shipdate ?l_shipdate ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount ;
        ltpch:order_nation_name ?cust_nation ;
        ltpch:partsupplier_nation_name ?supp_nation .
```

```
      filter ((
        (?cust_nation = "%NATION1%" && ?supp_nation = "%NATION2%") ||
        (?cust_nation = "%NATION2%" && ?supp_nation = "%NATION1%") ) &&
        (?l_shipdate >= "1995-01-01"^^xsd:date) &&
        (?l_shipdate <= "1996-12-31"^^xsd:date) )
      }
    }
}
group by
  ?supp_nation
  ?cust_nation
  ?li_year
order by
  ?supp_nation
  ?cust_nation
  ?li_year
```

```
params:[
 {name:NATION1,
  class:Nation,
  default:'FRANCE'
 },
 {name:NATION2,
  class:Nation2,
  default:'GERMANY'
 }
],
```

G.8 National Market Share Query (Q8)

This query determines how the market share of a given nation within a given region has changed over two years for a given part type.

> The market share for a given nation within a given region is defined as the fraction of the revenue, the sum of [l_extendedprice * (1-l_discount)], from the products of a specified type in that region that was supplied by suppliers from the given nation. The query determines this for the years 1995 and 1996 presented in this order.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?o_year
  ((?sum1 / ?sum2) as ?mkt_share)
where {
  { select
    ?o_year
    (sum (?volume * bif:equals (?nation, "%NATION%")) as ?sum1)
    (sum (?volume) as ?sum2)
    where {
      { select
          ((YEAR (?o_orderdate)) as ?o_year)
          ((?l_lineextendedprice * (1 - ?l_linediscount)) as ?volume)
          ?nation
        where {
          ?li qb:dataSet ltpch:lineitemCube ;
              ltpch:l_has_partsupplier ?ps ;
              ltpch:l_has_order ?ord ;
              ltpch:l_has_partsupplier ?ps ;
              ltpch:l_lineextendedprice ?l_lineextendedprice ;
              ltpch:l_linediscount ?l_linediscount .
          ?ps ltpch:ps_has_supplier ?s_supplier .
          ?s_supplier ltpch:s_has_nation ?n2 .
          ?n2 ltpch:n_name ?nation .
          ?ps ltpch:ps_has_part ?part .
          ?part ltpch:p_type ?type .
          ?ord ltpch:o_orderdate ?o_orderdate ;
            ltpch:o_has_customer ?c_customer .
```

```
            ?c_customer ltpch:c_has_nation ?n_nation .
            ?n_nation ltpch:n_has_region ?r_region .
            ?r_region ltpch:r_name ?region.
            filter ((?o_orderdate >= "1995-01-01"^^xsd:date) &&
              (?o_orderdate <= "1996-12-31"^^xsd:date &&
               ?region = "%REGION%" &&
               ?type = "%TYPE%")
            )
        }
      }
    }
    group by
      ?o_year
  }
}
order by
  ?o_year
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?o_year
  ((?sum1 / ?sum2) as ?mkt_share)
where
{
  {
    select
      ?o_year
      (sum (?volume * bif:equals (?nation, "%NATION%")) as ?sum1)
      (sum (?volume) as ?sum2)
    where
    {
      {
        select
          ((YEAR (?o_orderdate)) as ?o_year)
          ((?l_lineextendedprice * (1 - ?l_linediscount)) as ?volume)
          ?nation
        where
        {
          ?li qb:dataSet ltpch:lineitemCube ;
              ltpch:l_has_partsupplier ?ps ;
              ltpch:l_has_order ?ord ;
              ltpch:l_lineextendedprice ?l_lineextendedprice ;
              ltpch:l_linediscount ?l_linediscount .
          ?ps ltpch:nation_name ?nation ;
              ltpch:part_type ?p_type .
          ?ord ltpch:order_orderdate ?o_orderdate ;
               ltpch:region_name ?r_name .
          filter ((?o_orderdate >= "1995-01-01"^^xsd:date) &&
            (?o_orderdate <= "1996-12-31"^^xsd:date) &&
            (?r_name = "%REGION%") &&
            (?p_type = "%TYPE%")
          )
        }
      }
    }
    group by
      ?o_year
  }
}
order by
  ?o_year
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
```

```
select
  ?o_year
  ((?sum1 / ?sum2) as ?mkt_share)
where
{
  {
    select
      ?o_year
      (sum (?volume * bif:equals (?nation, "%NATION%")) as ?sum1)
      (sum (?volume) as ?sum2)
    where
    {
      {
        select
          ((YEAR (?o_orderdate)) as ?o_year)
          ((?l_lineextendedprice * (1 - ?l_linediscount)) as ?volume)
          ?nation
        where
        {
          ?li qb:dataSet ltpch:lineitemCube ;
              ltpch:l_lineextendedprice ?l_lineextendedprice ;
              ltpch:l_linediscount ?l_linediscount ;
              ltpch:partsupplier_nation_name ?nation ;
              ltpch:partsupplier_part_type ?p_type ;
              ltpch:order_order_orderdate ?o_orderdate ;
              ltpch:order_region_name ?r_name .
          filter ((?o_orderdate >= "1995-01-01"^^xsd:date) &&
            (?o_orderdate <= "1996-12-31"^^xsd:date) &&
            (?r_name = "%REGION%") &&
            (?p_type = "%TYPE%")
          )
        }
      }
    }
    group by
      ?o_year
  }
}
order by
  ?o_year
```

```
params:[
 {name:NATION,
  class:Nation,
  default:'BRAZIL'
 },
 {name:REGION,
  class:RegionForNation,
  default:'AMERICA'
 },
 {name:TYPE,
  class:Type,
  range:[1,2,3],
  default:'ECONOMY ANODIZED STEEL'
 }
],
```

G.9 Product Type Profit Measure Query (Q9)

This query determines how much profit is made on a given line of parts, broken out by supplier nation and year.

> The Product Type Profit Measure Query finds, for each nation and each year, the profit for all parts ordered in that year that contain a specified substring in their names and that were filled by a supplier in that nation. The profit is defined as the sum of [(l_extendedprice*(1-l_discount)) - (ps_supplycost * l_quantity)] for all lineitems describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first).

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?nation
  ?o_year
  (sum(?amount) as ?sum_profit)
where {
  { select
      ?nation
      ((YEAR (?o_orderdate)) as ?o_year)
      ((?l_lineextendedprice * (1 - ?l_linediscount) - ?ps_supplycost * ?l_linequantity)
          as ?amount)
    where {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_has_order ?ord ;
          ltpch:l_has_partsupplier ?ps ;
          ltpch:l_linequantity ?l_linequantity ;
          ltpch:l_lineextendedprice ?l_lineextendedprice ;
          ltpch:l_linediscount ?l_linediscount .
      ?ps ltpch:ps_has_part ?part ;
          ltpch:ps_has_supplier ?supp .
      ?supp ltpch:s_has_nation ?s_nation .
      ?s_nation ltpch:n_name ?nation .
      ?ord ltpch:o_orderdate ?o_orderdate .
      ?ps ltpch:ps_supplycost ?ps_supplycost .
      ?part ltpch:p_name ?p_name .
      filter (REGEX (?p_name, "%COLOR%"))
    }
  }
}
group by
  ?nation
  ?o_year
order by
  ?nation
  desc (?o_year)
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?nation
  ?o_year
  (sum(?amount) as ?sum_profit)
where {
  { select
      ?nation
      ((YEAR (?o_orderdate)) as ?o_year)
      ((?l_lineextendedprice * (1 - ?l_linediscount)
       - ?ps_supplycost * ?l_linequantity) as ?amount)
    where {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_has_order ?ord ;
          ltpch:l_has_partsupplier ?ps ;
          ltpch:l_linequantity ?l_linequantity ;
          ltpch:l_lineextendedprice ?l_lineextendedprice ;
          ltpch:l_linediscount ?l_linediscount .
      ?ps ltpch:nation_name ?nation .
      ?ord ltpch:order_orderdate ?o_orderdate .
      ?ps ltpch:partsupplier_supplycost ?ps_supplycost .
      ?ps ltpch:part_name ?p_name .
      filter (REGEX (?p_name,"%COLOR%"))
    }
  }
}
group by
  ?nation
```

```
  ?o_year
order by
  ?nation
  desc (?o_year)
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?nation
  ?o_year
  (sum(?amount) as ?sum_profit)
where {
  { select
      ?nation
      ((YEAR (?o_orderdate)) as ?o_year)
      ((?l_lineextendedprice * (1 - ?l_linediscount)
       - ?ps_supplycost * ?l_linequantity) as ?amount)
    where {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_linequantity ?l_linequantity ;
          ltpch:l_lineextendedprice ?l_lineextendedprice ;
          ltpch:l_linediscount ?l_linediscount ;
          ltpch:partsupplier_nation_name ?nation ;
          ltpch:order_order_orderdate ?o_orderdate ;
          ltpch:partsupplier_partsupplier_supplycost ?ps_supplycost ;
          ltpch:partsupplier_part_name ?p_name .
      filter (REGEX (?p_name,"%COLOR%"))
    }
  }
}
group by
  ?nation
  ?o_year
order by
  ?nation
  desc (?o_year)
```

```
params:[
 {name:COLOR,
  class:Color,
  default: green
 }
],
```

G.10 Returned Item Reporting Query (Q10)

The query identifies customers who might be having problems with the parts that are shipped to them.

> The Returned Item Reporting Query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter. The query lists the customer's name, address, nation, phone number, account balance, comment information and revenue lost. The customers are listed in descending order of lost revenue. Revenue lost is defined as sum(l_extendedprice*(1-l_discount)) for all qualifying lineitems.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?c_custkey
  ?c_companyName
  ((sum(?l_lineextendedprice * (1 - ?l_linediscount))) as ?revenue)
```

```
  ?c_acctbal
  ?nation
  ?c_address
  ?c_phone
  ?c_comment
where  {
  ?li qb:dataSet ltpch:lineitemCube ;
      ltpch:l_returnflag ?l_returnflag ;
      ltpch:l_has_order ?ord ;
      ltpch:l_lineextendedprice ?l_lineextendedprice ;
      ltpch:l_linediscount ?l_linediscount .
  ?ord ltpch:o_has_customer ?cust ;
       ltpch:o_orderdate ?o_orderdate .
  ?cust ltpch:c_address ?c_address ;
      ltpch:c_phone ?c_phone ;
      ltpch:c_comment ?c_comment ;
      ltpch:c_acctbal ?c_acctbal ;
      ltpch:c_custkey ?c_custkey ;
      ltpch:c_has_nation ?c_nation ;
      ltpch:c_name ?c_companyName .
   ?c_nation ltpch:n_name ?nation .
   filter ((?o_orderdate >= "%MONTH%-01"^^xsd:date) &&
      (?o_orderdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date)&&
      (?l_returnflag = "R"))
   )
}
group by
  ?c_custkey
  ?c_companyName
  ?c_acctbal
  ?nation
  ?c_address
  ?c_phone
  ?c_comment
order by
  desc (sum(?l_lineextendedprice * (1 - ?l_linediscount)))
limit 20
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?c_custkey
  ?c_companyName
  ((sum(?l_lineextendedprice * (1 - ?l_linediscount))) as ?revenue)
  ?c_acctbal
  ?nation
  ?c_address
  ?c_phone
  ?c_comment
where  {
  ?li ltpch:l_returnflag ?l_returnflag ;
      ltpch:l_has_order ?ord ;
      ltpch:l_lineextendedprice ?l_lineextendedprice ;
      ltpch:l_linediscount ?l_linediscount .
  ?ord ltpch:order_orderdate ?o_orderdate ;
      ltpch:customer_address ?c_address ;
      ltpch:customer_phone ?c_phone ;
      ltpch:customer_comment ?c_comment ;
      ltpch:customer_acctbal ?c_acctbal ;
      ltpch:customer_custkey ?c_custkey ;
      ltpch:customer_name ?c_companyName ;
      ltpch:nation_name ?nation .
   filter ((?o_orderdate >= "%MONTH%-01"^^xsd:date) &&
      (?o_orderdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) &&
      (?l_returnflag = "R"))
   ) .
}
group by
  ?c_custkey
```

```
  ?c_companyName
  ?c_acctbal
  ?nation
  ?c_address
  ?c_phone
  ?c_comment
order by
  desc (sum(?l_lineextendedprice * (1 - ?l_linediscount)))
limit 20
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?c_custkey
  ?c_companyName
  ((sum(?l_lineextendedprice * (1 - ?l_linediscount))) as ?revenue)
  ?c_acctbal
  ?nation
  ?c_address
  ?c_phone
  ?c_comment
where  {
  ?li ltpch:l_returnflag ?l_returnflag ;
      ltpch:l_lineextendedprice ?l_lineextendedprice ;
      ltpch:l_linediscount ?l_linediscount ;
      ltpch:order_order_orderdate ?o_orderdate ;
      ltpch:order_customer_address ?c_address ;
      ltpch:order_customer_phone ?c_phone ;
      ltpch:order_customer_comment ?c_comment ;
      ltpch:order_customer_acctbal ?c_acctbal ;
      ltpch:order_customer_custkey ?c_custkey ;
      ltpch:order_customer_name ?c_companyName ;
      ltpch:order_nation_name ?nation .
   filter ((?o_orderdate >= "%MONTH%-01"^^xsd:date) &&
      (?o_orderdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) &&
      (?l_returnflag = "R"))
   ) .
}
group by
  ?c_custkey
  ?c_companyName
  ?c_acctbal
  ?nation
  ?c_address
  ?c_phone
  ?c_comment
order by
  desc (sum(?l_lineextendedprice * (1 - ?l_linediscount)))
limit 20
```

```
params:[
  {name:MONTH,
   class:Month,
   range:[1993-02, 1995-1],
   default: 1993-10
  }
 ],
```

G.11 Important Stock Identification Query (Q11

This query finds the most important subset of suppliers' stock in a given nation.

> The Important Stock Identification Query finds, from scanning the available stock of suppliers in a given nation, all the parts that represent a significant percentage of the total value of all available parts. The query displays the part number and the value of those parts in descending order of value.

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?bigpspart
  ?bigpsvalue
where {
      { select
          ?bigpspart
          (sum(?b_supplycost * ?b_availqty) as ?bigpsvalue)
        where
          {
            ?bigps qb:dataSet ltpch:partSupplierCube ;
                   ltpch:ps_has_part ?bigpspart ;
                   ltpch:ps_supplycost ?b_supplycost ;
                   ltpch:ps_availqty ?b_availqty ;
                   ltpch:ps_has_supplier ?b_supplier .
            ?b_supplier ltpch:s_has_nation ?b_nation .
            ?b_nation ltpch:n_name "%NATION%" .
          }
          group by
            ?bigpspart
      } .
      {
      select
          ((sum(?t_supplycost * ?t_availqty) * %FRACTION%) as ?threshold)
        where
          {
            ?thr_ps qb:dataSet ltpch:partSupplierCube ;
                    ltpch:ps_has_part ?t_part ;
                    ltpch:ps_supplycost ?t_supplycost ;
                    ltpch:ps_availqty ?t_availqty ;
                    ltpch:ps_has_supplier ?t_supplier .
            ?t_supplier ltpch:s_has_nation ?t_nation .
            ?t_nation ltpch:n_name "%NATION%" .
          }
      }
    filter (?bigpsvalue > ?threshold)
  }
order by
  desc (?bigpsvalue)
```

```
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>

select
  ?bigpspart
  ?bigpsvalue
where
{
      {
        select
          ?bigpspart
          (sum(?b_supplycost * ?b_availqty) as ?bigpsvalue)
        where
        {
          ?bigps qb:dataSet ltpch:partSupplierCube ;
                 ltpch:ps_has_part ?bigpspart ;
                 ltpch:ps_supplycost ?b_supplycost ;
                 ltpch:ps_availqty ?b_availqty ;
                 ltpch:ps_has_supplier ?b_supplier .
          ?b_supplier ltpch:nation_name ?n_name .
          FILTER(?n_name = "%NATION%") .
        }
        group by
          ?bigpspart
      } .
      {
        select
```

```
        ((sum(?t_supplycost * ?t_availqty) * %FRACTION%) as ?threshold)
      where
      {
        ?thr_ps qb:dataSet ltpch:partSupplierCube ;
                ltpch:ps_has_part ?t_part ;
                ltpch:ps_supplycost ?t_supplycost ;
                ltpch:ps_availqty ?t_availqty ;
                ltpch:ps_has_supplier ?t_supplier .
        ?t_supplier ltpch:nation_name ?n_name .
        FILTER(?n_name = "%NATION%") .
      }
    }
  filter (?bigpsvalue > ?threshold)
}
order by
  desc (?bigpsvalue)
```

```
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>

select
  ?bigpspart
  ?bigpsvalue
where
{
      {
        select
          ?bigpspart
          (sum(?b_supplycost * ?b_availqty) as ?bigpsvalue)
        where
        {
          ?bigps qb:dataSet ltpch:partSupplierCube ;
                 ltpch:part_part_partkey ?bigpspart ;
                 ltpch:ps_supplycost ?b_supplycost ;
                 ltpch:ps_availqty ?b_availqty ;
                 ltpch:supplier_nation_name ?n_name .
          FILTER(?n_name = "%NATION%") .
        }
        group by
          ?bigpspart
      } .
      {
        select
          ((sum(?t_supplycost * ?t_availqty) * %FRACTION%) as ?threshold)
        where
        {
          ?thr_ps qb:dataSet ltpch:partSupplierCube ;
                  ltpch:part_part_partkey ?t_part ;
                  ltpch:ps_supplycost ?t_supplycost ;
                  ltpch:ps_availqty ?t_availqty ;
                  ltpch:supplier_nation_name ?n_name .
          FILTER(?n_name = "%NATION%") .
        }
      }
    filter (?bigpsvalue > ?threshold)
  }
order by
  desc (?bigpsvalue)
```

```
params:[
  {name:NATION,
   class:Nation,
   default:'GERMANY'
  },
  {name:FRACTION,
   class:Fraction,
   default: 0.0001
  }
 ],
```

G.12 Shipping Modes and Order Priority Query (Q12)

This query determines whether selecting less expensive modes of shipping is negatively affecting the critical-prior ity orders by causing more parts to be received by customers after the committed date.

> The Shipping Modes and Order Priority Query counts, by ship mode, for lineitems actually received by customers in a given year, the number of lineitems belonging to orders for which the l_receiptdate exceeds the l_commitdate for two different specified ship modes. Only lineitems that were actually shipped before the l_commitdate are considered. The late lineitems are partitioned into two groups, those with priority URGENT or HIGH, and those with a priority other than URGENT or HIGH.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?l_shipmode
  (sum (
    bif:or (
      bif:equals (?o_orderpriority, "1-URGENT"),
      bif:equals (?o_orderpriority, "2-HIGH") ) ) as ?high_line_count)
  (sum (1 -
    bif:or (
      bif:equals (?o_orderpriority, "1-URGENT"),
      bif:equals (?o_orderpriority, "2-HIGH") ) ) as ?low_line_count)
where  {

    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_has_order ?ord ;
       ltpch:l_commitdate ?l_commitdate ;
       ltpch:l_receiptdate ?l_receiptdate ;
       ltpch:l_shipmode ?l_shipmode ;
       ltpch:l_shipdate ?l_shipdate .
    ?ord ltpch:o_orderpriority ?o_orderpriority .
    filter (?l_shipmode in ("%SHIPMODE1%", "%SHIPMODE2%") &&
      (?l_commitdate < ?l_receiptdate) &&
      (?l_shipdate < ?l_commitdate) &&
      (?l_receiptdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?l_receiptdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date)) )
  }
group by
  ?l_shipmode
order by
  ?l_shipmode
```

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?l_shipmode
  (sum (
      bif:or (
        bif:equals (?o_orderpriority, "1-URGENT"),
        bif:equals (?o_orderpriority, "2-HIGH") ) ) as ?high_line_count)
```

```
  (sum (1 -
      bif:or (
        bif:equals (?o_orderpriority, "1-URGENT"),
        bif:equals (?o_orderpriority, "2-HIGH") ) ) as ?low_line_count)
where  {
    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_has_order ?ord ;
       ltpch:l_commitdate ?l_commitdate ;
       ltpch:l_receiptdate ?l_receiptdate ;
       ltpch:l_shipmode ?l_shipmode ;
       ltpch:l_shipdate ?l_shipdate .
    ?ord ltpch:order_orderpriority ?o_orderpriority .
    filter (?l_shipmode in ("%SHIPMODE1%", "%SHIPMODE2%") &&
      (?l_commitdate < ?l_receiptdate) &&
      (?l_shipdate < ?l_commitdate) &&
      (?l_receiptdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?l_receiptdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date)) )
  }
group by
  ?l_shipmode
order by
  ?l_shipmode
```

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?l_shipmode
  (sum (
      bif:or (
        bif:equals (?o_orderpriority, "1-URGENT"),
        bif:equals (?o_orderpriority, "2-HIGH") ) ) as ?high_line_count)
  (sum (1 -
      bif:or (
        bif:equals (?o_orderpriority, "1-URGENT"),
        bif:equals (?o_orderpriority, "2-HIGH") ) ) as ?low_line_count)
where  {
    ?li qb:dataSet ltpch:lineitemCube ;
       ltpch:l_commitdate ?l_commitdate ;
       ltpch:l_receiptdate ?l_receiptdate ;
       ltpch:l_shipmode ?l_shipmode ;
       ltpch:l_shipdate ?l_shipdate ;
       ltpch:order_order_orderpriority ?o_orderpriority .
    filter (?l_shipmode in ("%SHIPMODE1%", "%SHIPMODE2%") &&
      (?l_commitdate < ?l_receiptdate) &&
      (?l_shipdate < ?l_commitdate) &&
      (?l_receiptdate >= "%YEAR%-01-01"^^xsd:date) &&
      (?l_receiptdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date)) )
  }
group by
  ?l_shipmode
order by
  ?l_shipmode
```

```
params:[
  {name:SHIPMODE1,
   class:Shipmode,
   default:'MAIL'
  },
  {name:SHIPMODE2,
   class:Shipmode2,
   default:'SHIP'
  },
  {name:YEAR,
   class:Random,
   range:[1993, 1997],
   default: 1994
  }
 ],
```

G.13 Customer Distribution Query (Q13)

This query seeks relationships between customers and the size of their orders.

> This query determines the distribution of customers by the number of orders they have made, including customers who have no record of orders, past or present. It counts and reports how many customers have no orders, how many have 1, 2, 3, etc. A check is made to ensure that the orders counted do not fall into one of several special categories of orders. Special categories are identified in the order comment column by looking for a particular pattern.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?c_count
  (count(1) as ?custdist)
where {
    { select
        ?c_custkey
        (count (?ord) as ?c_count)
      where
        {
          ?cust ltpch:c_custkey ?c_custkey .
          optional {
            ?ord qb:dataSet ltpch:ordersCube ;
                 ltpch:o_has_customer ?cust ;
                 ltpch:o_comment ?o_comment .
             filter (!( REGEX (?o_comment , "%WORD1%.*%WORD2%" ) ) ) .
        }
        }
      group by
        ?c_custkey
    }
  }
group by
  ?c_count
order by
  desc (count(1))
  desc (?c_count)
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?c_count
  (count(1) as ?custdist)
where
{
```

```
    {
      select
        ?c_custkey
        (count (?ord) as ?c_count)
      where
      {
        ?cust ltpch:customer_custkey ?c_custkey .
        optional {
           ?ord ltpch:o_has_customer ?cust ;
                ltpch:o_comment ?o_comment .
            filter (!(REGEX(?o_comment, "%WORD1%.*%WORD2%")) ) .
        }
      }
      group by ?c_custkey
    }
  }
group by
  ?c_count
order by
  desc (count(1))
  desc (?c_count)
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?c_count
  (count(1) as ?custdist)
where
{
    {
      select
        ?c_custkey
        (count (?o_comment) as ?c_count)
      where
      {
        ?order ltpch:customer_customer_custkey ?c_custkey .
        optional {
            ?order ltpch:o_comment ?o_comment .
            filter (!(REGEX(?o_comment, "%WORD1%.*%WORD2%")) ) .
        }
      }
      group by ?c_custkey
    }
  }
group by
  ?c_count
order by
  desc (count(1))
  desc (?c_count)
```

```
params:[
  {name:WORD1,
   class:OneOf,
   range:[special,pending,unusual,express],
   default:special
  },
  {name:WORD2,
   class:OneOf,
   range:[packages, requests, accounts, deposits],
   default:requests
  }
 ],
```

G.14 Promotion Effect Query (Q14)

This query monitors the market response to a promotion such as TV advertisements or a special campaign.

> The Promotion Effect Query determines what percentage of the revenue in a given year and month was derived from promotional parts. The query considers only parts actually shipped in that month and gives the percentage. Revenue is defined as (l_extendedprice * (1-l_discount)).

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ((100 * ?sum1 / ?sum2 ) as ?promo_revenue)
where
{
  select
    (sum (
          bif:equals(SUBSTR(?p_type, 1, 5), "PROMO") *
          ?l_lineextendedprice * (1 - ?l_linediscount) ) as ?sum1)
    (sum (?l_lineextendedprice * (1 - ?l_linediscount)) as ?sum2)
  where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount ;
        ltpch:l_shipdate ?l_shipdate ;
        ltpch:l_has_partsupplier ?ps .
    ?ps ltpch:ps_has_part ?part .
    ?part ltpch:p_type ?p_type .
    filter ((?l_shipdate >= "%MONTH%-01"^^xsd:date) &&
      (?l_shipdate < bif:dateadd("month", 1, "%MONTH%-01"^^xsd:date)) )
  }
}
```

```
prefix bif:  <http://example.org/customfunction/>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>


select
  ((100 * ?sum1 / ?sum2 ) as ?promo_revenue)
where
{
  select
    (sum (
          bif:equals(SUBSTR(?p_type, 1, 5), "PROMO") *
          ?l_lineextendedprice * (1 - ?l_linediscount) ) as ?sum1)
    (sum (?l_lineextendedprice * (1 - ?l_linediscount)) as ?sum2)
  where {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_lineextendedprice ?l_lineextendedprice ;
          ltpch:l_linediscount ?l_linediscount ;
          ltpch:l_shipdate ?l_shipdate ;
          ltpch:l_has_partsupplier ?part .
      ?part ltpch:part_type ?p_type .
      filter ((?l_shipdate >= "%MONTH%-01"^^xsd:date) &&
        (?l_shipdate < bif:dateadd("month", 1, "%MONTH%-01"^^xsd:date)) )
  }
}
```

```
prefix bif:  <http://example.org/customfunction/>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>


select
  ((100 * ?sum1 / ?sum2 ) as ?promo_revenue)
```

```
where
{
  select
    (sum (
          bif:equals(SUBSTR(?p_type, 1, 5), "PROMO") *
          ?l_lineextendedprice * (1 - ?l_linediscount) ) as ?sum1)
    (sum (?l_lineextendedprice * (1 - ?l_linediscount)) as ?sum2)
  where {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_lineextendedprice ?l_lineextendedprice ;
          ltpch:l_linediscount ?l_linediscount ;
          ltpch:l_shipdate ?l_shipdate ;
          ltpch:partsupplier_part_type ?p_type .
      filter ((?l_shipdate >= "%MONTH%-01"^^xsd:date) &&
        (?l_shipdate < bif:dateadd("month", 1, "%MONTH%-01"^^xsd:date)) )
  }
}
```

```
params:[
  {name:MONTH,
   class:Month,
   range:[1993-01, 1997-12],
   default: 1995-09
  }
 ],
```

G.15 Top Supplier Query (Q15)

This query determines the top supplier so it can be rewarded, given more business, or identified for special recognition.

> The Top Supplier Query finds the supplier who contributed the most to the overall revenue for parts shipped during a given quarter of a given year. In case of a tie, the query lists all suppliers whose contribution was equal to the maximum, presented in supplier number order.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?s_suppkey
  ?s_name
  ?s_address
  ?s_phone
  ?total_revenue
where  {
    ?supplier a ltpch:supplier ;
        ltpch:s_suppkey ?s_suppkey ;
        ltpch:s_name ?s_name ;
        ltpch:s_address ?s_address ;
        ltpch:s_phone ?s_phone .
    { select
        ?supplier
        (sum(?l_extendedprice * (1 - ?l_discount)) as ?total_revenue)
       where {
            ?li1 qb:dataSet ltpch:lineitemCube ;
                 ltpch:l_shipdate ?l_shipdate ;
                 ltpch:l_lineextendedprice ?l_extendedprice ;
                 ltpch:l_linediscount ?l_discount ;
                 ltpch:l_has_partsupplier ?ps1 .
            ?ps1 ltpch:ps_has_supplier ?supplier .
            filter (
                ?l_shipdate >= "%MONTH%-01"^^xsd:date &&
                ?l_shipdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) )
       }
      group by
        ?supplier
      }
      { select (max (?l2_total_revenue) as ?maxtotal)
```

```
        where {
            { select
                  ?supplier2
                  (sum(?l2_extendedprice * (1 - ?l2_discount)) as ?l2_total_revenue)
               where {
                    ?li2 qb:dataSet ltpch:lineitemCube ;
                      ltpch:l_shipdate ?l2_shipdate ;
                      ltpch:l_lineextendedprice ?l2_extendedprice ;
                      ltpch:l_linediscount ?l2_discount ;
                       ltpch:l_has_partsupplier ?ps2 .
                  ?ps2 ltpch:ps_has_supplier ?supplier2 .
                    filter (
                        ?l2_shipdate >= "%MONTH%-01"^^xsd:date &&
                        ?l2_shipdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date)
                            )
               }
               group by
                ?supplier2
            }
        }
    }
    filter (?total_revenue = ?maxtotal)
}
order by
  ?supplier
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select distinct
  ?s_suppkey
  ?s_name
  ?s_address
  ?s_phone
  ?total_revenue
where
{
  ?partsupp ltpch:supplier_suppkey ?s_suppkey ;
            ltpch:supplier_name ?s_name ;
            ltpch:supplier_address ?s_address ;
            ltpch:supplier_phone ?s_phone .
  {
    select
        ?s_suppkey
        ((sum(?l_extendedprice * (1 - ?l_discount))) as ?total_revenue)
    where
    {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_shipdate ?l_shipdate ;
          ltpch:l_lineextendedprice ?l_extendedprice ;
          ltpch:l_linediscount ?l_discount ;
          ltpch:l_has_partsupplier ?ps .
      ?ps ltpch:supplier_suppkey ?s_suppkey .
      filter (
          ?l_shipdate >= "%MONTH%-01"^^xsd:date &&
          ?l_shipdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) )
    }
    group by
      ?s_suppkey
  } .
  {
    select
      (max (?l2_total_revenue) as ?maxtotal)
    where
    {
      {
        select
          ((sum(?l2_extendedprice * (1 - ?l2_discount))) as ?l2_total_revenue)
        where
        {
```

```
          ?li2 qb:dataSet ltpch:lineitemCube ;
              ltpch:l_shipdate ?l2_shipdate ;
              ltpch:l_lineextendedprice ?l2_extendedprice ;
              ltpch:l_linediscount ?l2_discount ;
              ltpch:l_has_partsupplier ?ps2 .
          ?ps2 ltpch:supplier_suppkey ?s_suppkey2 .
          filter (
              ?l2_shipdate >= "%MONTH%-01"^^xsd:date &&
              ?l2_shipdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) )
        }
        group by
          ?s_suppkey2
      }
    }
  }
  filter (?total_revenue = ?maxtotal)
}
order by
  ?s_suppkey
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select distinct
  ?s_suppkey
  ?s_name
  ?s_address
  ?s_phone
  ?total_revenue
where
{
  ?li ltpch:partsupplier_supplier_suppkey ?s_suppkey ;
    ltpch:partsupplier_supplier_name ?s_name ;
    ltpch:partsupplier_supplier_address ?s_address ;
    ltpch:partsupplier_supplier_phone ?s_phone .
  {
    select
        ?s_suppkey
        ((sum(?l_extendedprice * (1 - ?l_discount))) as ?total_revenue)
    where
    {
      ?li qb:dataSet ltpch:lineitemCube ;
          ltpch:l_shipdate ?l_shipdate ;
          ltpch:l_lineextendedprice ?l_extendedprice ;
          ltpch:l_linediscount ?l_discount ;
          ltpch:partsupplier_supplier_suppkey ?s_suppkey .
      filter (
          ?l_shipdate >= "%MONTH%-01"^^xsd:date &&
          ?l_shipdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) )
    }
    group by
      ?s_suppkey
  } .
  {
    select
      (max (?l2_total_revenue) as ?maxtotal)
    where
    {
      {
        select
          ((sum(?l2_extendedprice * (1 - ?l2_discount))) as ?l2_total_revenue)
        where
        {
          ?li2 qb:dataSet ltpch:lineitemCube ;
              ltpch:l_shipdate ?l2_shipdate ;
              ltpch:l_lineextendedprice ?l2_extendedprice ;
              ltpch:l_linediscount ?l2_discount ;
              ltpch:partsupplier_supplier_suppkey ?s_suppkey2 .
          filter (
              ?l2_shipdate >= "%MONTH%-01"^^xsd:date &&
```

```
                ?l2_shipdate < bif:dateadd ("month", 3, "%MONTH%-01"^^xsd:date) )
          }
          group by
            ?s_suppkey2
        }
      }
    }
    filter (?total_revenue = ?maxtotal)
}
order by
  ?s_suppkey
```

```
params:[
  {name:MONTH,
   class:Month,
   range:[1993-01, 1997-10],
   default:1996-01
  }
 ],
```

G.16 Parts/Supplier Relationship Query (Q16)

This query finds out how many suppliers can supply parts with given attributes. It might be used, for example, to determine whether there is a sufficient number of suppliers for heavily ordered parts.

> The Parts/Supplier Relationship Query counts the number of suppliers who can supply parts that satisfy a particular customer's requirements. The customer is interested in parts of eight different sizes as long as they are not of a given type, not of a given brand, and not from a supplier who has had complaints registered at the Better Business Bureau. Results must be presented in descending count and ascending brand, type, and size.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?p_brand
  ?p_type
  ?p_size
  ((count(distinct ?supp)) as ?supplier_cnt)
where {
    ?ps qb:dataSet ltpch:partSupplierCube ;
        ltpch:ps_has_part ?part ;
        ltpch:ps_has_supplier ?supp .
    ?part ltpch:p_brand ?p_brand ;
        ltpch:p_type ?p_type ;
        ltpch:p_size ?p_size .
    filter (
      (?p_brand != "%BRAND%") &&
      !( REGEX (?p_type,"^%TYPE%")) &&
      (?p_size in (%SIZE1%, %SIZE2%, %SIZE3%, %SIZE4%, %SIZE5%, %SIZE6%, %SIZE7%, %SIZE8
          %))
    ) .
    filter NOT EXISTS {
       ?supp ltpch:s_comment ?badcomment .
       filter ( REGEX (?badcomment, "Customer.*Complaints") ) .
    }
  }
group by
  ?p_brand
  ?p_type
  ?p_size
order by
  desc ((count(distinct ?supp)))
  ?p_brand
  ?p_type
  ?p_size
```

```
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>

select
  ?p_brand
  ?p_type
  ?p_size
  ((count(distinct ?supp)) as ?supplier_cnt)
where {
    ?ps qb:dataSet ltpch:partSupplierCube ;
        ltpch:ps_has_part ?part ;
        ltpch:ps_has_supplier ?supp .
    ?part ltpch:part_brand ?p_brand ;
          ltpch:part_type ?p_type ;
          ltpch:part_size ?p_size .
    FILTER (
      (?p_brand != "%BRAND%") &&
      !(REGEX(?p_type, "^%TYPE%")) &&
      (?p_size in (%SIZE1%, %SIZE2%, %SIZE3%, %SIZE4%, %SIZE5%, %SIZE6%, %SIZE7%, %SIZE8
          %))
    )
    FILTER NOT EXISTS {
       ?supp ltpch:supplier_comment ?badcomment .
       FILTER (REGEX(?badcomment, "Customer.*Complaints") )
    }
  }
group by
  ?p_brand
  ?p_type
  ?p_size
order by
  desc ((count(distinct ?supp)))
  ?p_brand
  ?p_type
  ?p_size
```

```
prefix bif: <http://example.org/customfunction/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>

select
  ?p_brand
  ?p_type
  ?p_size
  ((count(distinct ?supp)) as ?supplier_cnt)
where {
    ?ps qb:dataSet ltpch:partSupplierCube ;
        ltpch:supplier_supplier_suppkey ?supp ;
        ltpch:part_part_brand ?p_brand ;
        ltpch:part_part_type ?p_type ;
        ltpch:part_part_size ?p_size .
    FILTER (
      (!(REGEX(?p_brand , "%BRAND%","i")) &&
      !(REGEX(?p_type, "^%TYPE%")) &&
      (?p_size in (%SIZE1%, %SIZE2%, %SIZE3%, %SIZE4%, %SIZE5%, %SIZE6%, %SIZE7%, %SIZE8
          %))
    )
    FILTER NOT EXISTS {
       ?ps2 ltpch:supplier_supplier_suppkey ?supp ;
            ltpch:supplier_supplier_comment ?badcomment .
       FILTER (REGEX(?badcomment, "Customer.*Complaints") )
    }
  }
group by
  ?p_brand
  ?p_type
  ?p_size
```

```
order by
  desc ((count(distinct ?supp)))
  ?p_brand
  ?p_type
  ?p_size
```

```
params:[
 {name:BRAND,
  class:Brand,
  default:'Brand#45'
 },
 {name:TYPE,
  class:Type,
  range:[1, 2],
  default:'MEDIUM POLISHED'
 },
 {name:SIZE1, class:Random, range:[1, 50],
  default:49},
 {name:SIZE2, class:Random, range:[1, 50],
  default:14},
 {name:SIZE3, class:Random, range:[1, 50],
  default:23},
 {name:SIZE4, class:Random, range:[1, 50],
  default:45},
 {name:SIZE5, class:Random, range:[1, 50],
  default:19},
 {name:SIZE6, class:Random, range:[1, 50],
  default:3},
 {name:SIZE7, class:Random, range:[1, 50],
  default:36},
 {name:SIZE8, class:Random, range:[1, 50],
  default:9}
],
```

G.17 Small-Quantity-Order Revenue Query (Q17)

This query determines how much average yearly revenue would be lost if orders were no longer filled for small quantities of certain parts. This may reduce overhead expenses by concentrating sales on larger shipments.

> The Small-Quantity-Order Revenue Query considers parts of a given brand and with a given container type and determines the average lineitem quantity of such parts ordered for all orders (past and pending) in the 7-year data- base. What would be the average yearly gross (undiscounted) loss in revenue if orders for these parts with a quantity of less than 20% of this average were no longer taken?

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ((sum(?l_lineextendedprice) / 7.0) as ?avg_yearly)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_has_partsupplier ?ps .
    ?ps ltpch:ps_has_part ?part .
    ?part ltpch:p_brand ?p_brand ;
          ltpch:p_container ?p_container .
          {
            select
              ?part
              ((0.2 * avg(?l2_linequantity)) as ?threshold)
            where {
              ?li2  a ltpch:lineitem ;
                    ltpch:l_linequantity ?l2_linequantity ;
                    ltpch:l_has_partsupplier ?ps2 .
              ?ps2  ltpch:ps_has_part ?part .
```

```
          }
          group by
            ?part
        }
    filter (?l_linequantity < ?threshold && REGEX(?p_brand,"%BRAND%","i") && ?
        p_container = "%CONTAINER%")
}
```

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ((sum(?l_lineextendedprice) / 7.0) as ?avg_yearly)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_has_partsupplier ?ps .
    ?ps ltpch:part_partkey ?p_partkey.
    {
      select
        ?p_partkey
        ((0.2 * avg(?l2_linequantity)) as ?threshold)
      where {
        ?li2  a ltpch:lineitem ;
              ltpch:l_linequantity ?l2_linequantity ;
              ltpch:l_has_partsupplier ?ps2 .
        ?ps2 ltpch:part_partkey ?p_partkey ;
              ltpch:part_container ?p_container ;
              ltpch:part_brand ?p_brand  .
        filter (REGEX(?p_brand,"%BRAND%","i") && ?p_container = "%CONTAINER%" )
      }
      group by
        ?p_partkey
    }
    filter (?l_linequantity < ?threshold)
}
```

```
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ((sum(?l_lineextendedprice) / 7.0) as ?avg_yearly)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:partsupplier_part_partkey ?p_partkey .
    {
      select
        ?p_partkey
        ((0.2 * avg(?l2_linequantity)) as ?threshold)
      where {
        ?li2  a ltpch:lineitem ;
              ltpch:l_linequantity ?l2_linequantity ;
              ltpch:partsupplier_part_brand ?p_brand ;
              ltpch:partsupplier_part_container ?p_container ;
              ltpch:partsupplier_part_partkey ?p_partkey .
        filter (REGEX(?p_brand,"%BRAND%","i") && ?p_container = "%CONTAINER%" )
      }
      group by
        ?p_partkey
    }
    filter (?l_linequantity < ?threshold)
}
```

```
params:[
 {name:BRAND, class:Brand, default:'Brand#23'},
 {name:CONTAINER, class:Container, default:'MED BOX'}
],
```

Code Snippet 27: Query ones variable

G.18 Large Volume Customer Query (Q18)

The Large Volume Customer Query ranks customers based on their having placed a large quantity order. Large quantity orders are defined as those orders whose total quantity is above a certain level.

> The Large Volume Customer Query finds a list of the top 100 customers who have ever placed large quantity orders. The query lists the customer name, customer key, the order key, date and total price and the quantity for the order.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
   ?c_name
   ?c_custkey
   ?o_orderkey
   ?o_orderdate
   ?o_ordertotalprice
   (sum(?l_linequantity) as ?l_quantity)
where {
    ?li qb:dataSet ltpch:lineitemCube  ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_has_order ?ord .
    ?ord ltpch:o_orderkey ?o_orderkey ;
         ltpch:o_orderdate ?o_orderdate ;
         ltpch:o_ordertotalprice ?o_ordertotalprice ;
         ltpch:o_has_customer ?cust .
    ?cust ltpch:c_custkey ?c_custkey ;
          ltpch:c_name ?c_name .
    { select
        ?sum_order
        (sum (?l2_linequantity) as ?sum_q)
      where {
          ?li2 qb:dataSet ltpch:lineitemCube ;
               ltpch:l_linequantity ?l2_linequantity ;
               ltpch:l_has_order ?sum_order .
      }
      group by ?sum_order
    } .
    filter (?sum_order = ?ord && ?sum_q > %QUANTITY%)
}
group by
   ?c_name
   ?c_custkey
   ?o_orderkey
   ?o_orderdate
   ?o_ordertotalprice
order by
  desc (?o_ordertotalprice)
  ?o_orderdate
limit 100
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
   ?c_name
```

```
   ?c_custkey
   ?o_orderkey
   ?o_orderdate
   ?o_ordertotalprice
   (sum(?l_linequantity) as ?l_quantity)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_has_order ?ord .
    ?ord ltpch:order_orderkey ?o_orderkey ;
        ltpch:order_orderdate ?o_orderdate ;
        ltpch:order_ordertotalprice ?o_ordertotalprice ;
        ltpch:customer_custkey ?c_custkey ;
        ltpch:customer_name ?c_name .
    {
      select
        ?ord
        (sum (?l2_linequantity) as ?sum_q)
      where
      {
        ?li2 a ltpch:lineitem ;
             ltpch:l_linequantity ?l2_linequantity ;
             ltpch:l_has_order ?ord .
      }
      group by
        ?ord
    } .
    filter (?sum_q > %QUANTITY%)
}
group by
   ?c_name
   ?c_custkey
   ?o_orderkey
   ?o_orderdate
   ?o_ordertotalprice
order by
  desc (?o_ordertotalprice)
  ?o_orderdate
limit 100
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
   ?c_name
   ?c_custkey
   ?o_orderkey
   ?o_orderdate
   ?o_ordertotalprice
   (sum(?l_linequantity) as ?l_quantity)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:order_order_orderkey ?o_orderkey ;
        ltpch:order_order_orderdate ?o_orderdate ;
        ltpch:order_order_ordertotalprice ?o_ordertotalprice ;
        ltpch:order_customer_custkey ?c_custkey ;
        ltpch:order_customer_name ?c_name .
    {
      select
        ?o_orderkey2
        (sum (?l2_linequantity) as ?sum_q)
      where
      {
        ?li2 a ltpch:lineitem ;
             ltpch:l_linequantity ?l2_linequantity ;
             ltpch:order_order_orderkey ?o_orderkey2 ;
      }
      group by
        ?o_orderkey2
```

```
    } .
    filter (?sum_q > %QUANTITY%)
}
group by
   ?c_name
   ?c_custkey
   ?o_orderkey
   ?o_orderdate
   ?o_ordertotalprice
order by
  desc (?o_ordertotalprice)
  ?o_orderdate
limit 100
```

```
params:[
 {name:QUANTITY,
  class:Random, range:[312, 315], # but why default is 300?!
  default:300},
],
```

Code Snippet 28: Query ones variable

G.19 Discounted Revenue Query (Q19)

The Discounted Revenue Query reports the gross discounted revenue attributed to the sale of selected parts handled in a particular manner. This query is an example of code such as might be produced programmatically by a data mining tool.

> The Discounted Revenue query finds the gross discounted revenue for all orders for three different types of parts that were shipped by air and delivered in person. Parts are selected based on the combination of specific brands, a list of containers, and a range of sizes.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ((sum(?l_lineextendedprice * (1 - ?l_linediscount))) as ?revenue)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_shipmode ?l_shipmode ;
        ltpch:l_shipinstruct ?l_shipinstruct ;
        ltpch:l_has_partsupplier ?ps .
    ?ps ltpch:ps_has_part ?part .
     ?part ltpch:p_brand ?p_brand ;
          ltpch:p_size ?p_size ;
          ltpch:p_container ?p_container .
     filter (?l_shipmode in ("AIR", "AIR REG") &&
      ?l_shipinstruct = "DELIVER IN PERSON" &&
      ( ( (REGEX(?p_brand,"^%BRAND1%$","i")) &&
          (?p_container in ("SM CASE", "SM BOX", "SM PACK", "SM PKG")) &&
          (?l_linequantity >= %QUANTITY1%) &&
          (?l_linequantity <= %QUANTITY1% + 10) &&
          (?p_size >= 1) && (?p_size <= 5) ) ||
        ( (REGEX(?p_brand,"^%BRAND2%$","i")) &&
          (?p_container in ("MED BAG", "MED BOX", "MED PKG", "MED PACK")) &&
          (?l_linequantity >= %QUANTITY2%) &&
          (?l_linequantity <= %QUANTITY2% + 10) &&
          (?p_size >= 1) && (?p_size <= 10) ) ||
        ( (REGEX(?p_brand,"^%BRAND3%$","i")) &&
          (?p_container in ("LG CASE", "LG BOX", "LG PACK", "LG PKG")) &&
          (?l_linequantity >= %QUANTITY3%) &&
          (?l_linequantity <= %QUANTITY3% + 10) &&
          (?p_size >= 1) && (?p_size <= 15) ) ) )
  }
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ((sum(?l_lineextendedprice * (1 - ?l_linediscount))) as ?revenue)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_has_partsupplier ?ps ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_shipmode ?l_shipmode ;
        ltpch:l_shipinstruct ?l_shipinstruct .
     ?ps ltpch:part_brand ?p_brand ;
         ltpch:part_size ?p_size ;
         ltpch:part_container ?p_container .
     filter (?l_shipmode in ("AIR", "AIR REG") &&
      ?l_shipinstruct = "DELIVER IN PERSON" &&
      ( ( (REGEX(?p_brand,"^%BRAND1%$","i")) &&
          (?p_container in ("SM CASE", "SM BOX", "SM PACK", "SM PKG")) &&
          (?l_linequantity >= %QUANTITY1%) &&
          (?l_linequantity <= %QUANTITY1% + 10) &&
          (?p_size >= 1) && (?p_size <= 5) ) ||
        ( (REGEX(?p_brand,"^%BRAND2%$","i")) &&
          (?p_container in ("MED BAG", "MED BOX", "MED PKG", "MED PACK")) &&
          (?l_linequantity >= %QUANTITY2%) &&
          (?l_linequantity <= %QUANTITY2% + 10) &&
          (?p_size >= 1) && (?p_size <= 10) ) ||
        ( (REGEX(?p_brand,"^%BRAND3%$","i")) &&
          (?p_container in ("LG CASE", "LG BOX", "LG PACK", "LG PKG")) &&
          (?l_linequantity >= %QUANTITY3%) &&
          (?l_linequantity <= %QUANTITY3% + 10) &&
          (?p_size >= 1) && (?p_size <= 15) ) ) )
  }
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ((sum(?l_lineextendedprice * (1 - ?l_linediscount))) as ?revenue)
where {
    ?li qb:dataSet ltpch:lineitemCube ;
        ltpch:l_lineextendedprice ?l_lineextendedprice ;
        ltpch:l_linediscount ?l_linediscount ;
        ltpch:l_linequantity ?l_linequantity ;
        ltpch:l_shipmode ?l_shipmode ;
        ltpch:l_shipinstruct ?l_shipinstruct ;
        ltpch:partsupplier_part_brand ?p_brand ;
        ltpch:partsupplier_part_size ?p_size ;
        ltpch:partsupplier_part_container ?p_container .
     filter (?l_shipmode in ("AIR", "AIR REG") &&
      ?l_shipinstruct = "DELIVER IN PERSON" &&
      ( ( (REGEX(?p_brand,"^%BRAND1%$","i")) &&
          (?p_container in ("SM CASE", "SM BOX", "SM PACK", "SM PKG")) &&
          (?l_linequantity >= %QUANTITY1%) &&
          (?l_linequantity <= %QUANTITY1% + 10) &&
          (?p_size >= 1) && (?p_size <= 5) ) ||
        ( (REGEX(?p_brand,"^%BRAND2%$","i")) &&
          (?p_container in ("MED BAG", "MED BOX", "MED PKG", "MED PACK")) &&
          (?l_linequantity >= %QUANTITY2%) &&
          (?l_linequantity <= %QUANTITY2% + 10) &&
          (?p_size >= 1) && (?p_size <= 10) ) ||
        ( (REGEX(?p_brand,"^%BRAND3%$","i")) &&
          (?p_container in ("LG CASE", "LG BOX", "LG PACK", "LG PKG")) &&
          (?l_linequantity >= %QUANTITY3%) &&
          (?l_linequantity <= %QUANTITY3% + 10) &&
```

```
        (?p_size >= 1) && (?p_size <= 15) ) ) )
}
```

```
params:[
 {name:QUANTITY1, class:Random, range:[1, 10]
 , default:1},
 {name:QUANTITY2, class:Random, range:[10, 20]
 , default:10},
 {name:QUANTITY3, class:Random, range:[20, 30]
 , default:20},
 {name:BRAND1, class:Brand
 , default:'Brand#12'},
 {name:BRAND2, class:Brand
 , default:'Brand#23'},
 {name:BRAND3, class:Brand
, default:'Brand#34'}
],
```

G.20 Potential Part Promotion Query (Q20)

The Potential Part Promotion Query identifies suppliers in a particular nation having selected parts that may be can- didates for a promotional offer.

> The Potential Part Promotion query identifies suppliers who have an excess of a given part available; an excess is defined to be more than 50 % of the parts like the given part that the supplier shipped in a given year for a given nation. Only parts whose names share a certain naming convention are considered.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
  ?s_name
  ?s_address
where
{
  ?supp ltpch:s_name ?s_name ;
        ltpch:s_address ?s_address .
  {
    select distinct
      ?supp
    where
    {
      ?big_ps ltpch:ps_has_part ?part ;
              ltpch:ps_availqty ?big_ps_availqty ;
              ltpch:ps_has_supplier ?supp .
      ?supp ltpch:s_has_nation ?s_nation .
      ?s_nation ltpch:n_name ?n_name .
      ?part ltpch:p_name ?p_name .
      filter (REGEX (?p_name , "^%COLOR%") &&
              ?n_name = "%NATION%" &&
              ?big_ps_availqty > ?qty_threshold)
      {
        select
          ((0.5 * sum(?l_linequantity)) as ?qty_threshold)
          ?big_ps
        where
        {
          ?li qb:dataSet ltpch:lineitemCube ;
              ltpch:l_shipdate ?l_shipdate ;
              ltpch:l_linequantity ?l_linequantity ;
              ltpch:l_has_partsupplier ?big_ps .
          filter ((?l_shipdate >= "%YEAR%-01-01"^^xsd:date) &&
            (?l_shipdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date))
          )
        }
        group by
```

```
          ?big_ps
      }
    }
  }
}
order by ?s_name
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select distinct
  ?s_name
  ?s_address
where
{
  ?supp ltpch:supplier_name ?s_name ;
        ltpch:supplier_suppkey ?suppkey ;
        ltpch:supplier_address ?s_address .
  {
    select
      distinct ?suppkey
    where
    {
      ?big_ps ltpch:partsupplier_availqty ?big_ps_availqty ;
              ltpch:supplier_suppkey ?suppkey ;
              ltpch:nation_name ?n_name ;
              ltpch:part_name ?p_name .
      FILTER(REGEX (?p_name , "^%COLOR%") && ?n_name = "%NATION%") .
      {
        select
          ?big_ps
          ((0.5 * sum(?l_linequantity)) as ?qty_threshold)
        where
        {
          ?li qb:dataSet ltpch:lineitemCube ;
              ltpch:l_shipdate ?l_shipdate ;
              ltpch:l_linequantity ?l_linequantity ;
              ltpch:l_has_partsupplier ?big_ps .
          FILTER ((?l_shipdate >= "%YEAR%-01-01"^^xsd:date) &&
            (?l_shipdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date))
          )
        }
        group by
          ?big_ps
      } .
      FILTER(?big_ps_availqty > ?qty_threshold) .
    }
  }
}
order by ?s_name
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select distinct
  ?s_name
  ?s_address
where
{
  ?li2 ltpch:partsupplier_supplier_name ?s_name ;
       ltpch:partsupplier_supplier_suppkey ?suppkey ;
       ltpch:partsupplier_supplier_address ?s_address .
  {
    select
      distinct ?suppkey
```

```
    where
    {
      ?li     ltpch:partsupplier_partsupplier_availqty ?big_ps_availqty ;
              ltpch:partsupplier_supplier_suppkey ?suppkey ;
              ltpch:partsupplier_nation_name ?n_name ;
              ltpch:partsupplier_part_name ?p_name .
      FILTER(REGEX (?p_name , "^%COLOR%") && ?n_name = "%NATION%") .
      {
        select
          ?li
          ((0.5 * sum(?l_linequantity)) as ?qty_threshold)
        where
        {
          ?li qb:dataSet ltpch:lineitemCube ;
              ltpch:l_shipdate ?l_shipdate ;
              ltpch:l_linequantity ?l_linequantity .
          FILTER ((?l_shipdate >= "%YEAR%-01-01"^^xsd:date) &&
            (?l_shipdate < bif:dateadd ("year", 1, "%YEAR%-01-01"^^xsd:date))
          )
        }
        group by
          ?li
      } .
      FILTER(?big_ps_availqty > ?qty_threshold) .
    }
  }
}
order by ?s_name
```

```
params:[
 {name:COLOR, class:Color
 , default:forest},
 {name:YEAR, class:Random, range:[1993, 1997]
 , default:1994},
 {name:NATION, class:Nation
 , default:'CANADA'}
],
```

G.21 Suppliers Who Kept Orders Waiting Query (Q21)

This query identifies certain suppliers who were not able to ship required parts in a timely manner.

The Suppliers Who Kept Orders Waiting query identifies suppliers, for a given nation, whose product was part of a multi-supplier order (with current status of 'F') where they were the only supplier who failed to meet the committed delivery date.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>

select
    ?s_name
    (count(1) as ?numwait)
where {
          ?li1 qb:dataSet ltpch:lineitemCube;
              ltpch:l_receiptdate ?l1_receiptdate ;
              ltpch:l_commitdate ?l1_commitdate ;
              ltpch:l_has_partsupplier ?ps ;
              ltpch:l_has_order ?ord .
          ?ps ltpch:ps_has_supplier ?supp .
          ?supp ltpch:s_name ?s_name ;
                ltpch:s_has_nation ?s_nation .
          ?ord ltpch:o_orderstatus ?orderstatus .
          ?s_nation ltpch:n_name ?name
          filter (
            ?l1_receiptdate > ?l1_commitdate &&
            ?name = "%NATION%" &&
            ?orderstatus = "F"
```

```
            )
          filter exists {
            ?li2 ltpch:l_has_order ?ord ;
                 ltpch:l_has_partsupplier ?ps2 .
            ?ps2 ltpch:ps_has_supplier ?supp2 .
            filter (?supp != ?supp2)
          }
          filter not exists {
              ?li3 ltpch:l_has_order ?ord ;
                   ltpch:l_receiptdate ?l3_receiptdate ;
                   ltpch:l_commitdate ?l3_commitdate ;
                   ltpch:l_has_partsupplier ?ps3 .
              ?ps3 ltpch:ps_has_supplier ?supp3 .
              filter (
                 ?l3_receiptdate > ?l3_commitdate &&
                 ?supp3 != ?supp
              )
          }
       }
group by
   ?s_name
order by
    desc (count(1))
    ?s_name
limit 100
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
    ?s_name
    ((count(1)) as ?numwait)
where {
         ?li1 qb:dataSet ltpch:lineitemCube ;
              ltpch:l_receiptdate ?l1_receiptdate ;
              ltpch:l_commitdate ?l1_commitdate ;
              ltpch:l_has_partsupplier ?ps ;
              ltpch:l_has_order ?ord .
         ?ps ltpch:supplier_name ?s_name ;
             ltpch:supplier_suppkey ?suppkey ;
             ltpch:nation_name ?n_name .
         ?ord ltpch:order_orderstatus ?o_orderstatus .
         filter ( ?l1_receiptdate > ?l1_commitdate && ?n_name = "%NATION%" && ?
             o_orderstatus = "F")
         filter exists {
              ?li2 ltpch:l_has_order ?ord ;
                   ltpch:l_has_partsupplier ?ps2 .
              ?ps2 ltpch:supplier_suppkey ?suppkey2 .
              filter (?suppkey != ?suppkey2)
         }
         filter not exists {
              ?li3 ltpch:l_has_order ?ord ;
                   ltpch:l_receiptdate ?l3_receiptdate ;
                   ltpch:l_commitdate ?l3_commitdate ;
                   ltpch:l_has_partsupplier ?ps3 .
              ?ps3 ltpch:supplier_suppkey ?suppkey3 .
              filter (
                 ?l3_receiptdate > ?l3_commitdate &&
                 ?suppkey3 != ?suppkey
              )
         }
       }
group by
   ?s_name
order by
    desc (count(1))
    ?s_name
limit 100
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
    ?s_name
    ((count(1)) as ?numwait)
where {
         ?li1 qb:dataSet ltpch:lineitemCube ;
              ltpch:l_receiptdate ?l1_receiptdate ;
              ltpch:l_commitdate ?l1_commitdate ;
              ltpch:partsupplier_supplier_name ?s_name ;
              ltpch:partsupplier_supplier_suppkey ?suppkey ;
              ltpch:partsupplier_nation_name ?n_name ;
              ltpch:order_order_orderkey ?order_key ;
              ltpch:order_order_orderstatus ?o_orderstatus .
         filter ( ?l1_receiptdate > ?l1_commitdate && ?n_name = "%NATION%" && ?
             o_orderstatus = "F")
         filter exists {
              ?li2 ltpch:order_order_orderkey ?order_key ;
                   ltpch:partsupplier_supplier_suppkey ?suppkey2 .
              filter (?suppkey != ?suppkey2)
         }
         filter not exists {
              ?li3 ltpch:order_order_orderkey ?order_key ;
                   ltpch:l_receiptdate ?l3_receiptdate ;
                   ltpch:l_commitdate ?l3_commitdate ;
                   ltpch:partsupplier_supplier_suppkey ?suppkey3 .
              filter (
                 ?l3_receiptdate > ?l3_commitdate &&
                 ?suppkey3 != ?suppkey
              )
         }
       }
group by
   ?s_name
order by
    desc (count(1))
    ?s_name
limit 100
```

```
1  params:[
2   {name:NATION, class:Nation,
3    default:'SAUDI ARABIA'}
4  ],
```

G.22 Global Sales Opportunity Query (Q22)

The Global Sales Opportunity Query identifies geographies where there are customers who may be likely to make a purchase.

> This query counts how many customers within a specific range of country codes have not placed orders for 7 years but who have a greater than average âĂIJpositiveâĂİ account balance. It also reflects the magnitude of that balance. Country code is defined as the first two characters of c_phone.

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb: <http://purl.org/linked-data/cube#>



select
  ?cntrycode
  ((count (1)) as ?numcust)
  (sum (?c_acctbal) as ?totacctbal)
where {
```

```
    ?cust ltpch:c_acctbal ?c_acctbal ;
          ltpch:c_phone ?c_phone .
    {
      select ((avg(?c_acctbal2)) as ?acctbal_threshold)
      where
      {
        ?cust2 ltpch:c_acctbal ?c_acctbal2 ;
           ltpch:c_phone ?c_phone2 .
        filter ((?c_acctbal2 > 0.00) &&
          substr(?c_phone2, 1,2) in (%COUNTRY_CODE_SET%) ) .
      }
    }
    filter (
      (substr(?c_phone, 1,2)) in (%COUNTRY_CODE_SET%) &&
      (?c_acctbal > ?acctbal_threshold)
    ) .
    filter not exists { ?ord  qb:dataSet ltpch:ordersCube ;
                              ltpch:o_has_customer ?cust } .
  }
group by ((substr(?c_phone, 1,2)) as ?cntrycode )
order by (substr(?c_phone, 1,2))
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?cntrycode
  ((count (1)) as ?numcust)
  (sum (?c_acctbal) as ?totacctbal)
where
{
  ?cust ltpch:customer_acctbal ?c_acctbal ;
        ltpch:customer_phone ?c_phone .
  {
    select
      ((avg (?c_acctbal2)) as ?acctbal_threshold)
    where
    {
      ?cust2 ltpch:customer_acctbal ?c_acctbal2 ;
             ltpch:customer_phone ?c_phone2 .
      FILTER ((?c_acctbal2 > 0.00) &&
        SUBSTR(?c_phone2, 1, 2) in (%COUNTRY_CODE_SET%) )
    }
  }
  FILTER (SUBSTR(?c_phone, 1, 2) in (%COUNTRY_CODE_SET%) && ?c_acctbal > ?
      acctbal_threshold) .
  FILTER not exists { ?ord ltpch:o_has_customer ?cust } .
}
group by ((SUBSTR(?c_phone, 1, 2)) as ?cntrycode)
order by (SUBSTR(?c_phone, 1, 2))
```

```
prefix bif: <http://example.org/customfunction/>
prefix ltpch: <http://extbi.lab.aau.dk/ontology/ltpch/>
prefix qb:   <http://purl.org/linked-data/cube#>
prefix qb4o: <http://publishing-multidimensional-data.googlecode.com/git/index.html#
    ref_qbplus_>

select
  ?cntrycode
  ((count (1)) as ?numcust)
  (sum (?c_acctbal) as ?totacctbal)
where
{
  ?cust ltpch:customer_customer_acctbal ?c_acctbal ;
        ltpch:customer_customer_phone ?c_phone .
  {
    select
      ((avg (?c_acctbal2)) as ?acctbal_threshold)
```

```
    where
    {
      {
        select distinct
          ?custkey2 ?c_acctbal2
        where
        {
          ?cust2 ltpch:customer_customer_custkey ?custkey2 ;
                 ltpch:customer_customer_acctbal ?c_acctbal2 ;
                 ltpch:customer_customer_phone ?c_phone2 .
          FILTER ((?c_acctbal2 > 0.00) &&
            SUBSTR(?c_phone2, 1, 2) in (%COUNTRY_CODE_SET%) ) .
        }
      }
    }
  }
  FILTER (SUBSTR(?c_phone, 1, 2) in (%COUNTRY_CODE_SET%) && ?c_acctbal > ?
      acctbal_threshold) .
  FILTER not exists { ?cust qb:dataSet ltpch:ordersCube } .
}
group by ((SUBSTR(?c_phone, 1, 2)) as ?cntrycode)
order by (SUBSTR(?c_phone, 1, 2))
```

```
params:[
 {name:COUNTRY_CODE_SET, class:CountryCodeSet, range:[7],
  default:"'13', '31', '23', '29', '30', '18', '17'"}
],
```