SPARQLを対象としたクエリ書き換えによる異種データ統合

佐藤　祥吾†　　天笠　俊之††

† 筑波大学　情報理工学位プログラム　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学　計算科学研究センター　　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †s.sato@kde.cs.tsukuba.ac.jp, ††amagasa@cs.tsukuba.ac.jp

あらまし　全ての人が望むように利用・再掲載できるような形で入手できるデータをはじめとして, 複数のデータを組み合わせ, 活用することで, データに新たな価値を見出すことができる. しかし, 活用するにはデータの統合が必要であり, その統合には導入のコスト以外にも, 必要なデータの組合せの変化, 情報源の更新といった, 柔軟性も求められる. また, 統合したデータに対して統一的なデータ表現と検索方法が必要である. そこで本研究では, 情報源ごとにURIを生成する方法をデータ管理者が指定することができ, 異種の情報源に対してSPARQLクエリで問合せることができるデータ統合システムを提案する. 本研究のデータ統合システムでは, 同じエンティティを含む複数の情報源を柔軟に扱うことができ, より実用的なデータ統合をおこなうことができる.

キーワード　データ統合, SPARQL, RDF, クエリ書き換え

1 はじめに

オープンデータとは, 全ての人が望むように利用・再掲載できるような形で入手できるデータのことである. 例として, 日本政府が公開しているオープンデータであるDATA.GO.JP [1]は, 気象や人口など, 日本政府が関わる分野の多くの部分をデータとして公開している.

オープンデータをはじめとする複数のデータを組み合わせ, 活用することで, データに新たな価値を見出すことができる. オープンデータの活用事例が, オープンデータ100 [2]にて公開されている.

ここで，複数のデータを活用するためには, データ統合 [3]が必要である. データ統合とは, 複数の情報源から得られるデータを統合することで統一的に扱えるようにする処理を指す. データ統合をおこなうことで, 複数の情報源に対し, 一括してデータの検索をおこなうことができる.

データ統合を実現する際に, RDF [4]及びSPARQL [5]が注目されている. RDFとは, 主語-述語-目的語からなるトリプルとして構成され， RDFのデータはトリプルの集合として表現される. RDFは, 自身が持つデータ構造の柔軟性と, その他のデータ構造との互換性の高さから, データ統合を実現する際に有効なモデルとして注目される. また, SPARQLとは, RDFデータに対し基本グラフパターン (BGP) から問合せ条件を指定し, 照合するグラフパターンからRDFデータを検索することができるクエリ言語である. SPARQLは, SQLクエリに比べ, ユーザが直感的によりクエリを直感的に書くことができ, RDFのクエリ言語でもあることから, データ統合を実現する際に有効なクエリ言語として注目される.

実際に, SPARQLを問合せ言語としたデータ統合システムの手法が提案されている [6] [7] [8]. これらの手法では, 入力をSPARQLクエリとし, 入力されたBGPからトリプルパターンへ分解する. 分解されたトリプルパターンと実際の情報源の対応関係を用いることで, 入力されたSPARQLクエリは, 実際の情報源の問合せクエリに書き換えられる. 書き換えられた問合せクエリから, 情報源の検索結果を得る. そして, あらかじめ定義されているマッピング言語をもとに, 情報源が持つ非RDFデータを, RDFデータが持つURI(Uniform Resource Identifier) に変換し, ユーザが求める検索結果を出力する.

非RDFデータからRDFデータへ変換する際には, エンティティに対応するURIを適切に生成する必要がある. 既存研究では, IDなどをはじめとする, 非RDFである情報源に含まれる識別子情報に, 適当なURIのプレフィックスを付加することで, URIが生成できることを前提としている. しかし, 適切なURIの生成方法は情報源ごとに異なる可能性があり, 同じエンティティを含む複数の情報源を扱う際には, 情報源ごとにURIを生成する方法をデータ管理者が指定することができなければ, 多くの場合で実用的なデータ統合は困難である.

そこで本研究では, 情報源ごとにURIを生成する方法をデータ管理者が指定することができ, 異種の情報源に対してSPARQLクエリで問合せることができるデータ統合システムを提案する. 本研究のデータ統合システムでは, 同じエンティティを含む複数の情報源を柔軟に扱うことができ, より実用的なデータ統合をおこなうことができる.

本稿の構成は以下の通りである. まず， 2節で本研究における前提知識について説明する. 3節では関連研究について説明する. 4節では提案手法の説明を行い， 最後に5節で本研究のまとめと今後の方針について記す.

2 前提知識

2.1 RDF (Resource Description Framework)

情報源が持つデータを統一的に表現するためのデータ形式として, RDF(Resource Description Framework) [4]がある. RDF

とは, Web 上に存在する情報源を記述するために提案された, 統一された枠組みのことである.

RDF データの最小単位は, 主語 (Subject)-述語 (Predicate)-目的語 (Object) からなる RDF トリプルで構成され, RDF トリプル内の各要素は, 基本的に Web 上のデータの識別子である URI を用いて表現される.

URI 以外で RDF トリプル内の各要素を表現する方法は, RDF トリプル内の要素によって制約が異なる. 主語では URI または空白ノードで構成される. 述語は URI のみで構成される. 目的語は, URI, リテラルあるいは, 空白ノードで構成される. リテラルとは, URI のように形式化された識別子ではなく, 文字列をそのまま記述したものことをいう.

この RDF トリプルを組み合わせることで RDF グラフが構成される. RDF グラフは, 主語と目的語がノード, 述語がエッジを示すラベル付き有向グラフとして表現される.

2.2 オントロジ

統合した RDF データに用いられている URI が, どのような意味を持つか説明するものとして, オントロジがある. オントロジの代表的な例として, OWL(Web Ontology Language) [10] がある. OWL では, ウェブに存在するものごとの分類体系（クラス）やその関係、さらにはそれを推論していくためのルールを定義している.

2.3 SPARQL (SPARQL Protocol and RDF Query Language)

RDF によって統一的に表現されたデータから, 目的のデータを取得するために問合せを行う言語が, SPARQL(SPARQL Protocol and RDF Query Language) [5] である. ユーザは, SPARQL クエリで特定の基本グラフパターンを指定することで, 照合するグラフパターンを生成できる情報源のデータを検索することができる.

2.4 GAV (Global-As-View)

GAV（Global-As-View）[11] は，ビューを用いた情報統合の手法の一つである. 統合後のスキーマ（統合スキーマ）を,（統合スキーマとは異なる）情報源に対する問合せ結果で定義されるビューとして与える. その特徴として，情報源と統合スキーマとの対応関係を直感的な問合せとして与えられる事に加えて，統合スキーマに対する問合せを，単純なアルゴリズムによって情報源に対する問合せに変換することができる点があげられる. 一方，情報源の追加や変更に対して，マッピングルールの更新コストが大きい点が欠点とされている.

本研究では, 統合後のデータは RDF データであるため，RDF データをリレーションとして扱う. 具体的には，RDF トリプルをプロパティごとに分割し，同じプロパティを持つトリプルを，プロパティをリレーション名，属性がサブジェクトおよびオブジェクトとするリレーションとして扱う. これによって，任意の RDF トリプル集合を複数の固定された属性（サブジェクトおよびオブジェクト）からなるリレーションとして表現することができ，GAV での統合が可能となる.

3 関 連 研 究

3.1 情報源と RDF トリプルの対応付け

情報源を統一的に表現するデータモデルとして RDF を用いることは, RDF の持つデータ構造の柔軟性から有効である. ところが, RDF を用いて情報源を統合するには, 統合先の RDF データを適切に表現する必要がある. 特に，RDF では，表現対象の実体が URI として適切に表現されている必要がある. その一方で，情報源のレコードから適切な URI を導出することは単純には行えない.

そこで本研究では，情報源のレコードを対応する URI に変換する複数の手法を想定する. 一つめは，情報源の特定の属性値を元に，単純な変換で URI を導出可能，あるいは URI が直接格納されているケースである. 二つめは, 情報源のデータに含まれるキーワード等から, Wikidata や DBpedia 等の知識ベース中の実体に対応付ける方法である. 具体的には，実体リンキング（Entity Linking）等の既存の技術を利用して，対応する URI を特定する.

3.1.1 RDF マッピング言語

RDF トリプルと RDF 形式でない情報源の対応関係を記述し, それに基づき情報源のデータから RDF データを生成する手法が開発されている. RDF トリプルと情報源の対応関係を記述目的で開発された言語をマッピング言語と呼ぶ. R2RML（RDB to RDF Mapping Language）[9], RML（RDF Mapping Language）[12] がある. R2RML は, リレーショナルデータ形式となっている情報源を対象に, 対応関係を記述するためのマッピング言語である. また, RML は, R2RML よりも対応できるデータ形式を増やし, CSV や JSON, TSV, XML 等の情報源も対象にした, R2RML を拡張したマッピング言語である.

なお，RDF を生成する際に必要となる実体に対応する URI は，情報源の特定の属性値に対して定められた URI 接頭辞（プレフィックス）を付加することで生成することが想定されている. 実用性を考えると，これ以外の URI 生成方法についても対応することが必要であると考えられる.

3.1.2 実体リンキング（Entity Linking）

R2RML や RML がサポートする，URI 接頭辞を付加する方法による URI 生成以外にも，テキストやデータベースのレコードに対して，それが対応する実体への参照を推定する手法が研究されている. このような，テキストやレコードが指す実体を推定する手法を総称して実体リンキング（Entity Linking）と呼ぶ. 具体的には，テキスト（あるいはレコード）が与えられると，ターゲットとなる知識ベースに含まれる実体集合のうちマッチする実体が結果として返される. その例として, Wikification [13], DBpediaSpotlight [14] があり, これらはそれぞれ, キーワードから Wikipedia の URI, キーワードから DBpedia の URI の候補を算出している.

本研究では，統合後の RDF データにおける URI 生成において，URI 接頭辞を付加する方法や実体リンキングを利用する方法を含め，管理者が自由に URI を生成する方法を規定するこ

とを許す．これにより，より実用的なデータ統合が可能となる．

3.2 SPARQL を対象にしたデータ統合システム

リンクトオープンデータ（LOD）を代表として，RDF データ形式でのデータ公開が期待される一方，既存のデータベースのほとんどはリレーショナルデータベースや CSV，XML 等の非 RDF データとして存在している．そこで，RDF 以外の情報源を対象として，仮想的な RDF データとして統合し SPARQL による問合せを可能にするシステムが提案されている [6]?[8]．

3.2.1 Ultrawap

データ統合システムである Ultrawap [7], [8] では，入力を SPARQL クエリとし，入力された BGP からトリプルパターンへ分解する．分解されたトリプルパターンと実際の情報源の対応関係を用いることで，入力された SPARQL クエリは，実際の情報源の問合せクエリに書き換えられる．書き換えられた問合せクエリから，情報源の検索結果を得たのち，あらかじめ定義されているマッピング言語をもとに，RDF データが持つ URI(Uniform Resource Identier) に変換し，ユーザが求める検索結果を出力する．

3.2.2 Ontop

データ統合システムである Ontop [6] は，Ultrawap と同様，入力を SPARQL クエリとし，SPARQL クエリの検索結果を出力する.SPARQL クエリかつ，マッピングルールが GAV であるデータ統合システムのうち，Ontop は最新の手法となっている．

4 提 案 手 法

4.1 提案手法の方針

本研究では，リレーショナルデータベースを情報源としたデータ統合の結果を RDF データの仮想的なビューとしてユーザに提供し，SPARQL による問合せを可能にするシステムを提案する．データ統合は基本的には GAV（Global-as-View）による統合を基本とし，システム管理者は，情報源に対する問合せの結果として統合後の RDF データを記述する．その際，情報源の各レコードに対応する実体に対して適切な URI を生成する必要がある．上で述べたように，URI の生成方法は複数存在し，情報源ごとに適切に設定する必要があることから，本研究ではユーザ定義関数 `URI` およびその逆関数である $\texttt{URI}^{-1}$ をシステム管理者が適切に実装することを前提として，データ統合および問合せ処理を行うフレームワークを提案する．これにより，情報源ごとの事情に踏まえた実用的な統合が可能となる．

4.2 システムの構成

本研究が提案するデータ統合システムは次の 4 つの要素から，図 1 のように構成される．四つの要素は，統合するデータの情報源，統合したデータを RDF として表現するためのターゲットオントロジ，ターゲットオントロジと統合するデータの情報源の対応関係を表現するマッピング，マッピングを定義する際に用いられる `URI` 関数である．

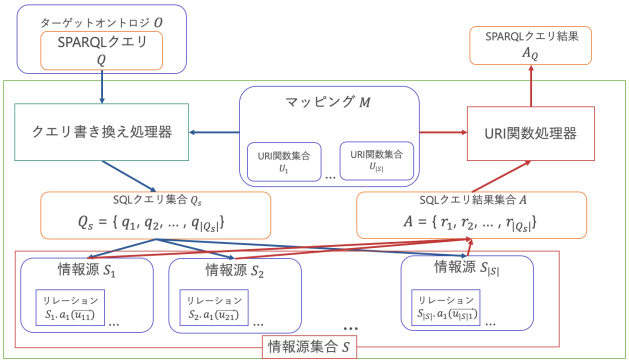

図 1 システムの全体像

4.2.1 情 報 源

本研究では，リレーショナルデータベースの情報源を想定する．情報源の集合を $S = \{S_1, S_2, ..., S_{|S|}\}$ と定義する．また，各情報源 S_i は，リレーション s とその属性集合 $\mathbf{u}$ を用いて，$S_i = \{S_i.s_1(\mathbf{u_1}), ..., S_i.s_{|S_i|}(\mathbf{u_{|S_i|}})\}$ と表現する．

4.2.2 ターゲットオントロジ

統合したデータを RDF として統一的に表現する際には，ターゲットオントロジを用いる．本稿では，ターゲットオントロジを $O = \{t_1, t_2, ..., t_{|O|}\}$ と定義する．また，$t_i = p_i(s_i, o_i)$ は，ターゲットオントロジの各要素であり，統合したデータを表現するための RDF トリプルである．このとき，s_i, p_i, o_i はそれぞれ，RDF トリプルの主語，術語，目的語である．

4.2.3 URI 関 数

情報源とターゲットオントロジとのマッピングを定義する際，レコードが指す実体に対応する URI を生成する必要がある．本手法では，これをユーザ定義関数 `URI` 関数としてシステム管理者が定義する．URI 関数とは，入力レコードに対して対応する URI を返す関数である．最も単純な例としては，情報源の特定の属性に含まれる URI をそのまま返す，あるいは，ID 属性に特定の URI 接頭辞を加える場合などが考えられる．また，`URI` 関数の中で Wikification [13] や DBpediaSpotlight [14] のような実体リンキングを呼び出す事によって，レコードやテキストに対応する実体およびその URI を得ることも可能である．

4.2.4 マッピング

ターゲットオントロジと統合するデータの情報源の対応関係を問合せによって記述する．ここで，リレーションではないターゲットオントロジに対して問合せ記述を可能とするため，RDF トリプル (s, p, o) をリレーション $p(s, o)$ とみなすことで，SQL での扱いを可能にする．すなわち，ターゲットトントロジに出現するプロパティに対応するマッピングルールを記述することで，情報源のリレーションとの対応を明示する．

本稿では，マッピングの集合を $M = \{m_1, m_2, ..., m_{|M|}\}$ と定義する．各マッピング m_i に関しては，以下の (1) 式と表現する．

$$m_i : p(URI(s), f(o)) \supseteq q \qquad (f \in \{URI, R\}) \qquad (1)$$

ここで，q は統合する情報源で記述された datalog クエリである．$p(URI(s), f(o))$ は，ターゲットオントロジの要素である

RDF トリプルであり，主語は URI，目的語は URI もしくはリテラルであることを示している．すなわち関数 f は，目的語が URI の場合 URI 関数，リテラルの場合文字列を返す関数 R である．

$$m_i\colon \quad \forall s, o(\exists y_1, ..., y_n(S_k.s_a(\mathbf{u_a})), ..., S_l.s_b(\mathbf{u_b})) \\ \Rightarrow p(URI(s), f(o))) \qquad (f \in \{URI, R\}) \tag{2}$$

このとき, s, o はクエリの検索結果に用いられる変数であり, 属性集合 **u** 内で用いられる属性変数でもある．また, $y_1, ..., y_n$ は, 属性集合 **u** 内で用いられる属性変数であり, s, o とは違い, 検索結果に用いられない箇所である．

4.3 URI 関数の分類

URI 関数は, 関数の定義域 (統合元の情報源のデータ) と その値域 (統合先の URI) の対応関係によって分類することができる．

4.3.1 1:1 関 数

統合元の情報源のデータと統合先の URI が全て 1:1 で対応しているとき, その URI 関数は 1:1 関数という．図 2 は, 日本の山の情報源と, wikidata の URI を対応付ける URI 関数の例である．この図 2 では, 富士山という情報源内のデータが与えられると, wd:Q39231 を返すような URI 関数である．

図 2 1:1 対応をする URI 関数例

4.3.2 n:1 関 数

統合元の情報源のデータと統合先の URI が n:1 で対応しているとき, その URI 関数は n:1 関数という．図 3 は, 日本の山の情報源と, wikidata の URI を対応付けるような URI 関数例である．この図 3 では, 富士山という情報源内のデータが与えられると, wd:Q39231 を返すような URI 関数である．

前述した 1:1 関数と異なるのは, Mt.Fuji という情報源内のデータが与えられても, 富士山と同じ wd:Q39231 を返す点である．

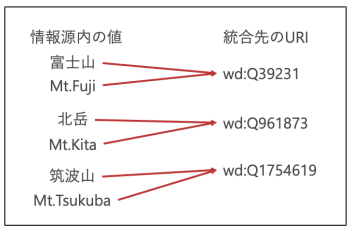

図 3 n:1 対応をする URI 関数例

Algorithm 1 クエリ処理

Input: SPARQL クエリ Q, データ統合システム $I =< S, O, M >$
Output: SPARQL クエリの回答 A_Q

1: Initialize Q_{sql}
2: **for** $(p_s(s_s, o_s)) \in Q$ **do**
3: Initialize Q_{or}
4: **for** $((p_m(URI(s_m), f(o_m)) \supseteq q(s_m, o_m)) \quad (f \in \{URI, S\})) \in M$ **do**
5: **if** $p_s == p_m$ **then**
6: $Q_{or} \leftarrow addORQuery(Q_{or}, q(s_m, o_m))$
7: **end if**
8: **end for**
9: $Q_{sql} \leftarrow addANDQuery(Q_{sql}, Q_{or})$
10: **end for**
11: $A \leftarrow Query(Q_{sql}, S)$
12: $A_Q \leftarrow Mapping(A, M)$
13: **return** A_Q

4.4 クエリ処理

本研究で提案したデータ統合システムでは, ユーザが統合したデータに問合せるときには SPARQL 言語を用いる．SPARQL 言語で記述されたクエリは, BGP と FILTER で構成される．BGP は, トリプルパターンの集合であり, クエリの本体部分である．FILTER は, BGP が持つトリプルパターンで用いた変数が持つ条件を記述する部分である．

提案したデータ統合システムは, SPARQL クエリを入力として受け取ると, SPARQL クエリで指定した SELECT 句における, 変数の組の結果を 1 レコードとした表形式のデータを出力する．

また, 提案したデータ統合システムは, 仮想の RDF データを扱うため, SPARQL クエリを直接適用することはできない．そこで, マッピング M を用いて, SPARQL クエリと同等の SQL クエリを作成し, SQL クエリのクエリ結果に URI 関数を適用する．よって, ユーザは, SPARQL クエリに問合せたときと同じ検索結果を得ることができる．

4.4.1 クエリ処理の流れ

マッピング M を用いて, SPARQL クエリと同等の SQL クエリを作成することを考える．このとき, SQL クエリを作成には Query Unfolding アルゴリズムを用いる．Algorithm 1 に, クエリ処理のアルゴリズムを示す．

クエリ処理では, データ統合システム $I =< S, O, M >$ に対して, SPARQL クエリ Q が与えられたとき, SPARQL クエリ Q の回答を返す．まず, 実際に情報源へ問合せるクエリとなる Q_{sql} の初期化を行う (1 行目)．次に, 各トリプルパターンに対して, そのトリプルパターンに合うマッピング m を用いることで, トリプルパターンは, トリプルパターンが包含する SQL クエリに置き換えることができる (2-10 行目)．各マッピング m は, RDF トリプルと情報源の対応関係を表現しており, トリプルパターンと情報源のクエリの対応関係にもなる．

ここで, ある 1 つのトリプルパターンが複数の SQL クエリを包含している場合があるため, このときは, その複数の SQL クエリで論理和をとることで, 情報源に対するクエリの回答を最大限得ることができる. アルゴリズム内では, 4-8 行目に対応し, 関数 $addORQuery(Q_{or}, q(s,o))$ を用いて上記のパターンを表現している. 具体例でこの関数を説明すると, $addORQuery((q_1 \cap q_2), q_3)$ の返り値は, $((q_1 \cap q_2) \cup q_3)$ である.

SPARQL クエリの BGP に含まれる各トリプルパターン間の関係は, 論理積となっているため, トリプルパターンから SQL クエリに置き換える際には, 異なるトリプルパターン間の SQL クエリは論理積で結合させる. アルゴリズム内では, 9 行目に対応し, 関数 $addANDQuery(Q_{sql}, q(s,o))$ を用いて上記のパターンを表現している. 具体例でこの関数を説明すると, $addANDQuery((q_1 \cap q_2), q_3)$ の返り値は, $((q_1 \cap q_2) \cap q_3)$ である.

ここまでの動作で, 実際の情報源に対する問合せを実行するために, SPARQL クエリは SQL クエリに変換することができる. 得られた SQL クエリを実際に情報源に対して実行することで, SQL クエリの回答 A を得ることができる. このとき得られた SQL クエリの回答 A は, URI 関数などを適用していないため, SPARQL に対する回答 A_Q ではない.

最後に, マッピング M を用いて, SQL クエリの回答 A を, SPARQL クエリに対して適した回答にマッピングすることで, SPARQL クエリに対する回答 A_Q を得ることができる.

具体例として, 図 4 の SPARQL クエリが得られ, マッピングが以下のときの場合, どのように SQL クエリに置き換えられるか説明する.

```
SELECT ?x1 , ?x2
WHERE {
?y1 p1 ?x1.
?y1 p2 ?y2.
?y2 p3 ?x2.
}
```

図 4　SPARQL クエリ例 1: FILTER を含まない SPARQL クエリ

- マッピング
  - $p_1(URI(s), R(o)) \supseteq q_1(s,o)$
  - $p_1(URI(s), R(o)) \supseteq q_2(s,o)$
  - $p_2(URI(s), URI(o)) \supseteq q_3(s,o)$
  - $p_3(URI(s), R(o)) \supseteq q_4(s,o)$

与えられた SPARQL クエリは, マッピングの左辺に合わせて表記すると, 以下のように表記できる. ただし, SPARQL クエリで変数を表す記号である？は省略している.

$$\forall x_1, x_2(\exists y_1, y_2(p_1(y_1, x_1), p_2(y_1, y_2), p_3(y_2, x_2)))$$

与えられたマッピングをもとに, 上記の SPARQL クエリを SQL クエリに置き換えると以下のように表記できる.

$$\forall x_1, x_2(\exists y_1, y_2((q_1(y_1, x_1) \cup q_2(y_1, x_1)) \cap q_2(y_1, y_2) \cap q_3(y_2, x_2)))$$

情報源に対して上記の SQL クエリを実行し, そして, クエリ結果にマッピングを実行すれば, SPARQL クエリの回答が得られる.

4.4.2　FILTER に URI を含む条件がある場合の処理

前項で紹介したクエリ処理では, FILTER の探索条件に URI を含まなければ, 問題なく SPARQL クエリの回答を得ることができる.

しかし, FILTER の探索条件に URI を含む場合, SPARQL クエリ内のトリプルパターンから SQL クエリに置き換える場合には, URI を含む探索条件をそのまま SQL クエリに引き継ぐことはできない. なぜならば, 実際の情報源のデータは全て, URI 関数を適用する前の文字列で表現されているからである. そこで, 提案するデータ統合システムでは, URI 関数は, 情報源の規模に合わせてデータ管理者が決定できることとする. 本稿では, SPARQL クエリに対応する, URI を含む探索条件から, SQL クエリに対応する探索条件の変換関数を, $URI^{-1}(\cdot)$ と表記する. ただし, 前述した関数の入力は URI のみとする.

具体例として, 図 5 の FILTER を含む SPARQL クエリが得られ, マッピングが以下のときの場合, どのように SQL クエリに置き換えられるか説明する.

```
SELECT ?x1 , ?x2
WHERE {
?y1 p1 ?x1.
?y1 p2 ?y2.
?y2 p3 ?x2.
?y2 p3 ?y3.
FILTER(?y3 = ex:c)
}
```

図 5　SPARQL クエリ例 2:
FILTER に URI を含む SPARQL クエリ

- マッピング
  - $p_1(URI(s), R(o)) \supseteq q_1(s,o)$
  - $p_1(URI(s), R(o)) \supseteq q_2(s,o)$
  - $p_2(URI(s), URI(o)) \supseteq q_3(s,o)$
  - $p_3(URI(s), R(o)) \supseteq q_4(s,o)$
  - $p_4(URI(s), URI(o)) \supseteq q_5(s,o)$

SPARQL クエリ内の FILTER に関して, ex:c は URI である. 与えられた SPARQL クエリは, マッピングの左辺に合わせて表記すると, 以下のように表記できる. ただし, SPARQL クエリで変数を表す記号である？は省略している.

$$\forall x_1, x_2(\exists y_1, y_2, y_3(p_1(y_1, x_1), p_2(y_1, y_2), p_3(y_2, x_2), p_3(y_2, y_3), (y_3 = \text{ex:c})))$$

与えられたマッピングをもとに, 上記の SPARQL クエリを SQL クエリに置き換えると以下のように表記できる.

$$\forall x_1, x_2(\exists y_1, y_2, y_3((q_1(y_1, x_1) \cup q_2(y_1, x_1)) \cap$$
$$q_3(y_1, y_2) \cap q_4(y_2, x_2) \cap (q_5(y_2, y_3), y_3 \theta URI^{-1}(\text{ex:c}))))$$

SPARQL クエリ内の FILTER で記載されていた探索条件は, SQL クエリでは, $y_3 \theta URI^{-1}(\text{ex:c})$ と表記される. ただし, θ は演算子である. $=$ ではなく θ と表記するのは, URI 関数がデータの取り得る数値の範囲などによって URI が決定する場合, SQL クエリで情報源に問合せる際は $=$ ではなく, 別の演算子になる可能性があるためである.

5 評価実験

本実験では, 提案手法に基づいて実装したシステムのクエリ処理が、現実的な時間で処理できるか検証する.

5.1 データセット

本研究では, wikidata から観光に関する抽出したデータセットを用いる. データセットは合計 45000 タプルのデータがあり, ジャンルごとに 4 つの情報源に分割されている.

5.2 実験環境

提案手法の実装には, Python3.9.2 および sqlite3(3.39.4) を用いた. 実行環境は, Intel® Core™ i5-1038NG7 CPU を搭載した Mac OS 13.0.1 である.

また, 既存手法のデータ統合システムとして, Ontop CLI 4.2.2 を用いた.

5.3 実験方法

本研究では, 表現するマッピング技術を RDF トリプルと情報源の対応関係ごとに指定できるデータ統合システムを実現することを目的としている. 提案したデータ統合システムのクエリ処理が, 現実的な実行時間で処理できるか確認するために, マッピングルールが GAV であるデータ統合システムの Ontop と比較する.

実験方法は, 評価クエリに対して同じ問合せ結果が得られるように, 比較手法である Ontop と提案手法でマッピングを記述する. そして, それぞれのデータ統合システムに対して評価クエリを実行し, 問合せ結果が得られるまでの実行時間を比較する.

5.4 評価クエリ

結果の規模が異なるクエリを 4 つ作成し, これを用いて評価を行う. クエリ結果の規模はそれぞれ, 約 800, 約 2000, 約 4000, 約 20000 タプルとなっている.

5.5 実験結果

2 つの手法について, 評価クエリごとの実行時間の比較を行う. 各手法についての, 評価クエリごとの実行時間の比較を図 1 に示す. 図 1 から, 全ての評価クエリに対して, 提案手法が既存手法である Ontop に比べ, 短い実行時間で問合せ結果を返せることが示された.

表 1 評価クエリごとの実行時間 [単位：s]

クエリ	提案手法	比較手法 (Ontop)
Q1	0.468	3.135
Q2	0.529	9.700
Q3	0.601	15.471
Q4	0.640	51.096

6 まとめと今後の課題

本研究では, ユーザ定義関数である URI 関数を導入することで, 表現するマッピング技術を RDF トリプルと情報源の対応関係ごとに指定でき, 異種の情報源に対して SPARQL クエリで問合せることができるデータ統合システムを提案した. 実験から, 提案手法はマッピングと URI 関数を適切に設定することで, 既存手法である Ontop と同じ問合せ結果を得ることができることが示され, 更に既存手法よりも短い実行時間で問合せ処理をすることが示された.

今後の課題として, ユーザ定義関数である URI 関数と, マッピングルールに LAV(Local As View) を用いたデータ統合システムの検討が考えられる. 提案手法のマッピングルールは GAV を用いているため, 提案手法の実装は容易であるものの, 情報源が持つリレーションが頻繁に追加, 削除される場合, 提案手法のデータ統合システムは, その都度マッピングや URI 関数を更新しなければならない. そこで, もう一つのマッピングルールである LAV を用いることで, 情報源が持つリレーションが頻繁に追加, 削除されても, GAV に比べて少ない頻度で, マッピングの更新が済むことが期待できる.

文　献

[1] データカタログサイト: https://www.data.go.jp/

[2] オープンデータ 100: https://www.digital.go.jp/resources/data_case_study/

[3] Serge Abiteboul, et al. Web Data Management. http://webdam.inria.fr/Jorge/files/wdm.pdf

[4] Graham Klyne. Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/2004/RECrdf-concepts-20040210/,2004.

[5] Steve Harris, Andy Seaborne, EricPrud'hommeaux. Sparql 1.1 query language. W3C recommendation, Vol. 21, No. 10, p. 778, 2013.

[6] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL Queries over Relational Databases. Semantic Web Journal. 2017.

[7] J. F. Sequeda, M. Arenas, and D. P. Miranker. OBDA: query rewriting or materialization? in practice, both! In ISWC, 2014.

[8] Juan F. Sequeda, Daniel P. Miranker. Web Semantics: Science, Services and Agents on the World Wide WebVolume 22October, 2013 pp 19–39

[9] Souripriya Das, Seema Sundara, Richard Cyganiak, https://www.w3.org/TR/r2rml/

[10] OWL(Web Ontology Language): https://www.w3.org/TR/owl-features/

[11] Katsis, Y., Papakonstantinou, Y. (2009). View-based Data Integration. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems.

[12] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, Rik Walle Proceedings of the 7th Workshop on Linked Data on the Web 2014

[13] Rada Mihalcea, Andras Csomai. Wikify! Linking Documents to Encyclopedic Knowledge. CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management

[14] DBpediaSpotlight: https://www.dbpedia-spotlight.org/