# 報告書

## 1 Knowledge Graph

Knowledge Graph (KG) とは, データや情報の間の関係を表現するための構造化されたデータモデルである. 主にエンティティ (ものや概念) と, それらのエンティティ間の関係 (リレーション) を明示的に表現することに重点を置く. KG は, データを単なるリストや表として保存するのではなく, 意味 (セマンティクス) を持つネットワークとして表現することが特徴である. KG の基本構造で, ノードとしてエンティティ, エッジとしてリレーションがあり, それらに付随する追加情報として属性もある.

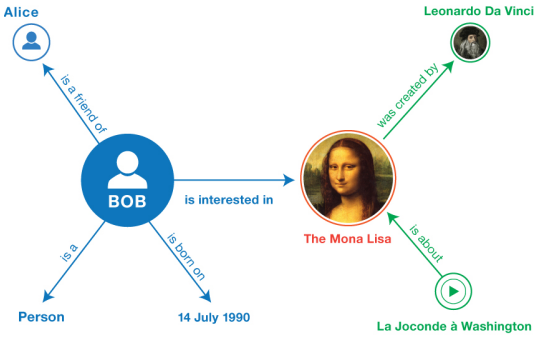


図 1: KG の例

### 1.1 機能と利点

KG の利点は, データの意味や関係を深く理解し, 複雑な情報を効果的に管理・活用できる点にある. 以下に KG の機能と利点を挙げる.

#### 1.1.1 意味的検索と質問応答

KG を利用すると, 単純なキーワード検索よりも文脈を理解した意味的な検索が可能になる. これにより, ユーザーが投げかける質問に対してより正確で関連性の高い情報を返すことができる.

#### 1.1.2 クエリ能力

SPARQL のようなクエリ言語を使うことで, 複雑なデータ関係を簡単に検索できる. 従来のリレーショナルデータベースでは困難な問い合わせでも, KG なら効率的に処理できる.

#### 1.1.3 データの統合と相互運用性

異なるデータソースからの情報を統合し, 関係性を明確にすることで, データの一貫性と相互運用性を高める, 例えば, 異なるデータベースから集められた情報を一つの KG に統合して使用することが可能である.

#### 1.1.4 データのリンクと推論

KG は, 異なるエンティティ間の新しい関係性を推論する能力を持っている. これにより, 明示的に記述されていない情報を推測し, 新たな知見を得ることができる.

#### 1.1.5 スケーラビリティとリアルタイム性

大規模なデータセットに対しても, KG はスケーラブルに対応でき, リアルタイムでのデータ処理や検索が可能である. これにより, ビッグデータや高速で変化するデータに対しても効果的に対応できる.

#### 1.1.6 AI や機械学習の強化

KG は, AI や機械学習アルゴリズムに豊かな文脈情報を提供するため, より高度な分析や推論を実現する. 例えば, 推薦システムや自然言語処理の性能向上に寄与する.

#### 1.1.7 説明可能な AI

KG を利用することで, AI システムの決定や予測がどのように導かれたのかを説明する根拠を提供しやすくなる. これにより, AI の透明性と信頼性が向上する.

### 1.2 活用例

- Google Knowledge Graph
- 推薦システム
- ビジネスインテリジェンス
- 医療分野での応用

KG は, データを意味的に豊かなネットワークとしてモデル化することで, 情報の検索, 統合, 推論を高度に行うための強力なツールである. これにより, 複雑なデータ環境においても, 情報の理解と活用が容易になる.

### 1.3 データセット

**aaa**
: aaa

**bbb**
: bbb

**ccc**
: ccc

**ddd**
: ddd

# 2 RDF

Resource Description Framework (RDF) は, ウェブ上のリソース (データ) を記述するためのフレームワークで, 主に KG やセマンティックウェブの基盤として使用される. RDF は, 情報を「主語-述語-目的語」の形式で記述し, データ間の関係を明示的に表現している.

## 2.1 基本概念

**3 つ組 (トリプル)**

RDF のデータは,「主語 (Subject) -述語 (Predicate) -目的語 (Object)」のトリプルで表現される. 例えば, “Alice :knows :Bob” というトリプルは,「アリスはボブを知っている」という関係を表現している.

**リソース**

RDF では, あらゆるものを「リソース」として扱う. リソースは主語や目的語として使われるもので, URI (Uniform Resource Identifier) で識別される. URI は, リソースを一意に識別するための標準的な方法である.

**プロパティ**

プロパティは, 主語と目的語の間の関係を表現する述語部分であり, これも URI で識別される. 例えば, “:knows” は「知っている」という関係を表すプロパティである.

**リテラル**

リテラルは, RDF のトリプル内で目的語として使用される値で, 文字列, 数値, 日付などの具体的なデータを表す. 例えば, “:age :hasValue ‘30’ ” のように, リテラルとして「30」という数値を目的語に使用できる.

## 2.2 データモデル

RDF のデータモデルは, グラフ構造として表現される. 各トリプルはグラフのエッジを構成し, 主語と目的語はノードとして扱われる. これにより, 複雑なネットワーク状のデータ関係を視覚化・分析することができる.

## 2.3 RDF スキーマ

RDF スキーマ (RDF Schema: RDFS) は, RDF データの語彙を定義し, クラスやプロパティの階層構造を規定するためのメカニズムである. RDFS を使うことで, RDF データに型付けや制約を加えることが可能となる.

**クラス**

エンティティの種類を定義する. 例えば, “:Person” クラスを定義すると, “:Alice” や “:Bob” はこのクラスのインスタンスとして扱われる.

**サブクラス**

クラス間の階層関係を定義する. 例えば, “:Student” は “:Person” のサブクラスとして定義される.

**プロパティの型**

プロパティに適用できるデータ型やクラスを指定する. 例えば, “:hasAge” プロパティは “xsd:integer” 型のリテラル値しか持てない, といった制約が可能である.

### 2.4 RDF の例

Listing 1: RDF の例

```
@prefix ex: <http://example.org/>.

ex:Alice ex:knows ex:Bob .
ex:Alice ex:age "30"^^xsd:integer .
ex:Bob ex:age "25"^^xsd:integer .
```

この例では, “ex:Alice” が “ex:Bob” を知っているという関係と, Alice が 30 歳, Bob が 25 歳であることが記述されている.

## 3 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) は, リソース記述フレームワーク (RDF) データを検索および操作するためのクエリ言語である. SPARQL は, この RDF データから特定の情報を抽出するために使用される.

### 3.1 特徴

**RDF データモデルのサポート**

SPARQL は, RDF トリプル (主語-述語-目的語の形式) に基づいてクエリを実行する. RDF トリプルは, リソース間の関係を表現するために使用され, SPARQL はこれらの関係を利用してデータを検索する.

**複雑なクエリ構造**

SPARQL は, 単純なクエリから非常に複雑なクエリまで対応できる. フィルタリング, 条件付きクエリ, データの集約, ソート, リミットなど, SQL に似た機能を備えている.

**柔軟なデータ照合**

RDF データの中から, 指定したパターンに一致するものを抽出できる. これにより, 特定のリソースやその属性, 関係性に基づいてデータを取得できる.

**標準化されたプロトコル**

SPARQL は W3C によって標準化されており, セマンティックウェブの主要なクエリ言語として広く採用されている. これにより, 異なるシステム間での互換性が確保されている.

World Wide Web Consortium (W3C) は, ウェブ技術の標準化を行う国際的な組織である. 1994 年に Tim Berners-Lee によって設立され, ウェブの発展と互換性を確保するための技術標準を策定している. HTML, CSS, XML, RDF, SPARQL など, 多くのウェブ技術の標準化を担当しており, これにより, ウェブ上での互換性と統一性が確保され, 開発者は様々なプラットフォームやブラウザで動作するウェブサイトやアプリケーションを作成できるようになっている.

### 3.2 SPARQL クエリの基本構造

Listing 2: SPARQL クエリの基本構造

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

```

```
SELECT ?name ?email
WHERE {
  ?person a foaf:Person .
  ?person foaf:name ?name .
  ?person foaf:mbox ?email .
}
```

**PREFIX**

RDF スキーマや語彙の名前空間を定義する. “foaf:” のように, クエリ内で省略形を使って URI を短縮することができる.

**SELECT**

返される結果の変数を指定する. この例では, “?name” と “?email” が結果に含まれる.

**WHERE**

クエリの条件を指定する. RDF トリプルのパターンを使って, どのデータを抽出するかを定義する. この部分で指定された条件に一致するデータが返される.

### 3.3 応用例

**KG の検索**

SPARQL を使って, KG から特定のエンティティ間の関係を検索できる.

**セマンティックウェブのデータ探索**

ウェブ上の RDF データを探索し, リソース間のリンクをたどることができる.

**データ統合と分析**

異なるデータソースから取得した RDF データを統合し, SPARQL を使ってそのデータを分析することができる.

## 4 URI

Uniform Resource Identifier (URI) は, Web 上にあるあらゆるファイルを認識するための識別子の総称で, Uniform Resource Name (URN) と Uniform Resource Locator (URL) で構成されている. インターネット上やその他の情報空間で, 特定のリソースを指し示すために使用される.

**URN**

URI の一種で, リソースの名前を指定するものである. リソースの場所に関係なく, そのリソースを一意に識別する. 例えば, ‘urn:isbn:0451450523’ は特定の書籍の国際標準図書番号 (International Standard Book Number: ISBN) を示す URN である.

**URL**

URI の一種で, リソースの場所を指定するものである. 例えば, ‘https://www.example.com’ は URL である.

### 4.1 構造

Listing 3: URI の構造

```
scheme://authority/path?query#fragment
```

**scheme**

リソースにアクセスするためのプロトコルや方法を指定する. 一般的な例には 'http', 'https', 'ftp', 'mailto', 'file' などがある.

**authority**

リソースが存在するホスト (ドメイン名や IP アドレス) や, オプションでポート番号を含む.

例: 'www.example.com', 'user:password@example.com:8080'

**path**

ホスト内でリソースが位置する場所を指定する. ファイルシステムの path と同様である.

例: '/path/to/resource'

**query**

リソースに追加の情報や指示を提供するためのオプションのパラメータである. 通常は "?" の後に 'key=value' の形式で記述され, 複数のパラメータは '&' で区切る.

例: '?id=123&sort=asc'

**fragment**

リソース内の特定の部分を指し示すために使われる. 通常は '#' の後に記述される.

例: '#section1'

### 4.2 役割

**ウェブ上のリソース識別**

ウェブ上のページ, 画像, 動画, ファイル, メールアドレスなど, あらゆるリソースを一意に識別するために使用される.

**リンクとナビゲーション**

ウェブページ間をリンクで繋ぐときに, URI を使って目的のリソースを指定する. 例えば, 'ja href="https://www.examp
では, 'https://www.example.com' がリンク先として使われている.

**データの統合**

KG やセマンティックウェブでは, RDF トリプルの中でリソースを識別するために URI を使う. これにより, 異なるデータソース間でのデータ統合やリンクが容易になる.

## 5 SQL

Structured Query Language (SQL) は, リレーショナルデータベース管理システム (Relational DataBase Management System: RDBMS) でデータを操作するための標準的なプログラミング言語である. SQL を使ってデータベースの作成, データの挿入, 検索, 更新, 削除などの操作ができる.

### 5.1 基本的な機能

**データベース操作**

SQL は, データベースのスキーマ (構造) を定義したり, 新しいデータベースやテーブルを作成するために使われる.

例: ‘CREATE DATABASE my_database;’

**データの挿入, 更新, 削除**

SQL は, データをデータベースに追加したり, 既存のデータを変更したり, 削除するために使用される.

例:

- 挿入: ‘INSERT INTO users (name, age) VALUES (’Alice’, 30);’
- 更新: ‘UPDATE users SET age = 31 WHERE name = ’Alice’;’
- 削除: ‘DELETE FROM users WHERE name = ’Alice’;’

**データの検索と取得**

SQL の強力な機能の一つがデータ検索である. 複雑なクエリを使って, 条件に合ったデータを効率的に抽出できる.

例: ‘SELECT name, age FROM users WHERE age ¿ 25;’

**データの結合**

SQL は, 複数のテーブル間で関連するデータを結合し, 一つの結果セットとして返すことができる. これにより, 複雑なデータ構造を容易に操作することができる. 例: ‘SELECT orders.id, customers.name FROM orders JOIN customers ON orders.customer_id = customers.id;’

### 5.2 利点

**標準化**

SQL は, ANSI と ISO によって標準化されており, 多くのリレーショナルデータベースシステム (MySQL, PostgreSQL, SQLite, Oracle など) で広くサポートされている.

**強力なクエリ機能**

SQL は, データの抽出, 変換, 集計を柔軟に行うための強力なクエリ機能を備えている. 複雑なビジネスロジックをデータベース内で効率的に処理できる.

**データの一貫性と整合性**

SQL は, データの一貫性と整合性を保つためのトランザクション処理機能や制約機能 (プライマリキー, 外部キー, ユニークキーなど) を提供する.

## 6 今後の課題

- Text to KG の再現実験
- データセットの調査
- 学会への参加 (情報知識学会, IIAI, KG 推論チャレンジ, 言語処理学会)

## 参考文献