Explainabilirty Query Based

Rosario Bongiorno, Daniele Franco

1 Introduzione

Your introduction goes here! Simply start writing your document and use the Recompile button to view the updated PDF preview. Examples of commonly used commands and features are listed below, to help you get started.

Once you're familiar with the editor, you can find various project settings in the Overleaf menu, accessed via the button in the very top left of the editor. To view tutorials, user guides, and further documentation, please visit our help library, or head to our plans page to choose your plan.

Ad ogni LLM viene affidato un ruolo ben preciso, e questo viene fatto tramite un prompt iniziale, cioè fornito una sola volta, chiamato *system*, in cui viene specificato il compito principale del modello e lo stile delle risposte che deve restituire.

2 Modello proposto

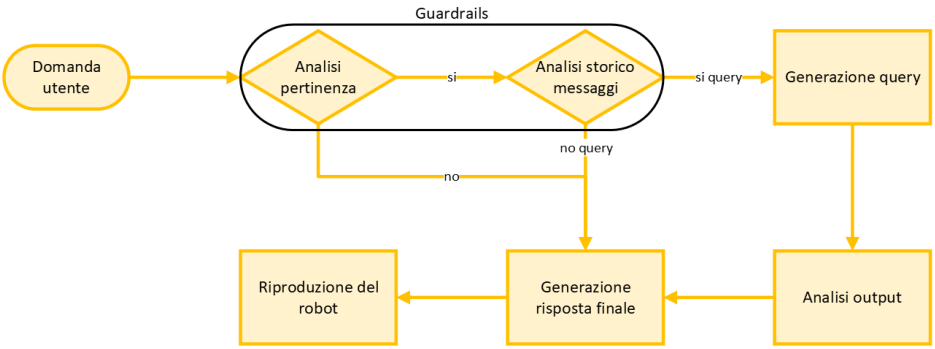

Figura 1: Flusso degli eventi

2.1 Guardrails

La componente dei *guardrails* garantisce una "protezione" al sistema e il suo nome, infatti, deriva proprio dalla funzione che svolge. Si assicura che l'utente ponga domande inerenti solo allo scenario corrente e che non vengano generate *query* anche quando il modello possiede già le informazioni per rispondere alla domanda.

In particolare, il secondo controllo viene effettuato analizzando lo storico della conversazione, per verificare di quali informazioni è a conoscenza il modello.

L'implementazione viene realizzata utilizzando un *Large Language Model*, in particolare il modello `gpt-4o`. Le due condizioni vengono fornite nel system nel seguente modo:

```
Devi verificare le seguenti condizioni:
1) La domanda posta è pertinente allo scenario corrente?
2) L'oggetto della domanda è già presente negli output precedenti?
Rispondi ad entrambe esclusivamente con "si" o "no" (esempio :si no)
```

Il prompt di ogni singola interrogazione, invece, è così composto:

```
Output precedente:
{output}

Domanda:  {question}
```

All'interno di `output` e `question` verranno inseriti rispettivamente l'output dell'ultima *query* e la domanda dell'utente. In questo modo, avendo fornito il *system*, l'LLM risponderà sempre allo stesso modo ad ogni interrogazione.

2.2 Generazione query

Il modulo di generazione delle query costituisce una parte fondamentale del sistema proposto, consentendo al robot di trasformare automaticamente le domande in linguaggio naturale in interrogazioni Cypher eseguibili su un database Neo4j. Questo processo, basato sull'impiego di Large Language Model (LLM), garantisce la generazione della query corretta che sia in grado di estrarre dalla base di conoscenza le informazioni di cui abbiamo bisogno. Per assicurare la correttezza sintattica delle query, è stata adottata la tecnica del Few Shot Prompting, che guida il modello attraverso esempi strutturati e contestuali.

Il prompt è stato progettato per essere generico e riutilizzabile in qualsiasi dominio, ed è suddiviso in tre sezioni: un prefisso che definisce il ruolo del robot e fornisce lo schema del database, una serie di esempi domanda-query e un suffisso che include informazioni contestuali (come il giorno e il risultato della query precedente), per preservare la coerenza nel dialogo. L'inclusione dell'output della query precedente consente al sistema di mantenere il contesto della conversazione, evitando ripetizioni o ambiguità e per ricavare informazioni che non sono state fornite nella domanda.

Il prompt utilizzato si può analizzare in Figura 2.

```
Tu sei un Robot di nome Pepper, esperto in Neo4j.
Data una domanda in input crea una query Cypher sintatticamente corretta
da eseguire.
Qui trovi lo schema con le informazioni del database neo4j:{schema}.
Sotto trovi un numero di esempi di domande e la loro rispettiva query
Cypher.

"question": "Cosa può mangiare rosario oggi?",
"query": "MATCH (r:Ricetta)..."
…
Oggi è {giorno}.
Output risposta precedente:{prec_res}
per ogni informazione che manca nella domanda utilizza l'output della
query precedente.
Ritornami esclusivamente la query da eseguire e non aggiungere altro
testo o altri simboli.

Domanda: {question}
Query:
```

Figura 2: Prompt Generazione Query

2.3 Analisi Output

Una volta che la *query* viene generata, deve essere eseguita per fornire all'utente una risposta effettiva, e non illustrare soltanto il ragionamento. Verrà restituito, quindi, un output che conterrà le informazioni da integrare nella risposta finale. In questo modo, l'LLM ha a disposizione informazioni aggiuntive, date dalle variabili esplicative presenti nell'output, e può fornire all'utente una spiegazione più precisa.

2.4 Generazione Risposta Finale

La componente di generazione della risposta finale è quella che si occupa di rielaborare tutte le informazioni provenienti dalle altre componenti, fornendo una risposta coerente ed esplicativa. Anche questa componente è stata implementata utilizzando un LLM, il modello `gpt-4o`, a cui è stato fornito il seguente *system*:

```
Sei un robot di nome Pepper che assiste una persona per l'alimentazione e
utilizzi come base di conoscenza un database Neo4j che interroghi con le
query Cypher.
Questa persona ti fa delle domande e per rispondere ricavi le informazioni
dal database, rielaborandole in un discorso di senso compiuto, spiegando i
motivi della risposta e il ragionamento che hai utilizzato.
```

In esso vengono utilizzate due tecniche di *prompting*:

- **Persona-based Prompting**: è stato descritto il compito che deve svolgere, impersonificando il robot Pepper.
- **Chain-of-Thougth Prompting**: viene chiesto di esporre il ragionamento utilizzato per arrivare alla risposta, e di fornire i motivi che portano a quella decisione.

Per quanto riguarda il singolo prompt, invece, si differenzia a seconda della casistica riscontrata: poiché tutte le componenti convergono in questa, a seconda del percorso intrapreso, si presenterà un caso diverso, e il sistema deve essere in grado di fornire una risposta adeguata per ognuna di essi.

2.4.1 Caso 1: Domanda non pertinente

Nel caso in cui il controllo della pertinenza effettuato dal guardrail dovesse fallire, il sistema comunica all'utente che la domanda che è stata posta non è strettamente legata allo scenario, esortandolo a formularne una pertinente.

2.4.2 Caso 2: Output vuoto

Potrebbe capitare che l'output prodotto dalla *query* sia vuoto o nullo, e quindi l'LLM non ha informazioni da poter restituire. Questo può succedere in due casi:

- la *query* presenta degli errori sintattici, e quindi, quando si prova ad eseguirla, riporta un errore e di conseguenza non viene prodotto nessun output.
- la *query* è corretta, ma eseguendola non viene trovata nessuna corrispondenza nella base di conoscenza per mancanza di dati o informazioni, e quindi, pur non presentando errori, l'output prodotto risulta vuoto.

Il primo caso può verificarsi, per esempio, quando la domanda posta si discosta notevolmente dalle domande di esempio, oppure è poco chiara o confusionaria, e quindi l'LLM prova a generare la *query* interpretando la domanda, non sempre riuscendo in maniera ottimale.

Il secondo caso, invece, può verificarsi quando la domanda posta fa riferimento a delle entità non presenti nella base di conoscenza, come una ricetta o una persona, non trovando corrispondenze.

In entrambi i casi, il sistema comunica all'utente che la domanda posta non è stata compresa, e che quindi non è in grado di rispondere correttamente, invitandolo a riformularla.

2.4.3 Caso 3: Query non generata

Nel caso in cui la *query* non venga generata, a causa del controllo del guardrail sulle informazioni già possedute, l'LLM non deve leggere la *query* e formulare un nuovo ragionamento, in quanto è stato già effettuato nella risposta precedente. Ciò che deve fare, quindi, è rispondere alla domanda basandosi sull'output della *query* precedente.

Il prompt fornito all'LLM è composto nel modo seguente:

```
Attenendoti esclusivamente agli output precedenti, rispondi alla seguente
domanda in modo esaustivo spiegando e motivando il perché della risposta e
il ragionamento che segui per rispondere.
Domanda: {state.get("question")}
Fornisci una risposta sintetica e discorsiva come se la dovesse
pronunciare il robot a voce al paziente in questione, quindi senza
elenchi puntati e senza dettagli tecnici sul database e simulando una
conversazione.
```

La prima parte specifica il fatto di doversi attenere esclusivamente alle informazioni già possedute, per evitare il fenomeno delle allucinazioni tipico degli LLM, cioè rispondere inventando, e quindi facendo riferimento ad informazioni non presenti nella base di conoscenza. La seconda parte, invece, dà delle indicazioni sulla forma che deve assumere la risposta per adattarla al tipo di interlocutore, in questo caso discorsiva e semplificata per il paziente.

La variabile `question`, invece, è la domanda dell'utente che viene recuperata dallo stato persistente del sistema.

2.4.4 Caso 4: Query generata

Questo rappresenta il caso classico in cui l'utente pone una domanda, i *guardrails* stabiliscono che risulta pertinente ed è necessario effettuare la *query*, e quindi deve essere letta dall'LLM. Il prompt fornito è composto come segue:

```
Data una query, creami un discorso che spieghi i vari controlli che
effettua e infine la conclusione a cui sei arrivato.  Per ogni step della
query fornisci anche il risultato, considerando il suo output, e infine
dai un resoconto.
Query:  {state.get("query")}
Output:  {state.get("output")}
Domanda:  {state.get("question")}
Fornisci una risposta sintetica e discorsiva come se la dovesse
pronunciare il robot a voce al paziente in questione, quindi senza
elenchi puntati, senza dettagli tecnici sul database e simulando una
conversazione.
```

Nella prima parte viene specificato l'obiettivo dell'LLM, cioè quello di leggere la *query* spiegando i controlli effettuati *step-by-step*, e di arrivare ad una conclusione considerando l'output della stessa. La seconda parte, invece, rimane invariata con il caso precedente.

Le variabili `query`, `output` e `question` rappresentano rispettivamente la *query*, l'output e la domanda dell'utente, che sono memorizzate nello stato persistente del sistema.

2.5 Riproduzione del robot

Per l'implementazione del modello nel robot è stato utilizzato il *framework* UnipaQ. L'idea alla base è quella che, per permettere all'utente di interagire con il robot, esso deve innanzitutto presentarsi e comunicare la sua intenzione nell'assistere l'utente nelle sue richieste. Successivamente, deve mettersi in modalità ascolto attivando i microfoni. Una volta che l'utente avrà finito di parlare, verrà effettuato lo *speech-to-text* in modo da convertire la domanda in una stringa e poterla dare in pasto al modello. Quando quest'ultimo avrà generato una risposta da poter pronunciare, il robot ne effettuerà il *text-to-speech* in modo da rispondere all'utente oralmente, simulando una conversazione.

Per la realizzazione, sono stati creati due *controller*, STT Controller ed Explainability Controller, che, attraverso uno scambio di messaggi su dei topic ROS2, comunicano tra loro e con il *framework* UnipaQi.

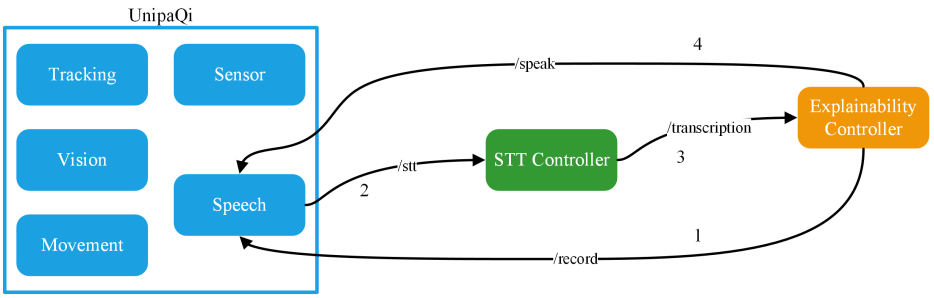

Figura 3: Rappresentazione comunicazione nodi

L'esecuzione, come evidenziato in Figura 3, è costituita da 4 passi:

1. L'Explainability Controller, che manda un messaggio al nodo Speech di UnipaQi sul topic `/record`, per *triggerare* la funzione di *callback*.

2. Il nodo Speech fa pronunciare la frase di benvenuto al robot e successivamente attiva i microfoni ed inizia l'ascolto. Esso termina nel momento in cui rileva silenzio, quindi quando l'utente avrà terminato di parlare. Una volta che la domanda è stata pronunciata, viene salvato il file della registrazione in formato `.wav` e viene inviato in locale dal robot in un percorso predefinito. Questo percorso verrà inviato dal nodo Speech all'STT Controller sotto forma di stringa utilizzando il topic `/stt`.

3. L'STT Controller potrà avere accesso al file audio e lo manderà in pasto al modello di intelligenza artificiale specializzato nel riconoscimento dei suoni, Whisper. Esso restituirà la stringa corrispondente alla domanda posta dall'utente, che verrà mandata all'Explainability Controller attraverso il topic `/transcription`.

4. L'Explainability Controller consegna la trascrizione della domanda al modello per poterla processare e ricevere la risposta finale, che verrà mandata al nodo Speech tramite il topic `/speak` che la farà pronunciare al robot ad alta voce, aggiungendo delle animazioni del corpo.

Una volta descritte tutte le componenti del modello, sono state assemblate per valutare effettivamente la loro efficienza, effettuando prima dei test mirati per comprendere meglio il ruolo da assegnare ad ogni LLM, e poi delle esecuzioni complete per confrontare le diverse risposte finali fornite.

3 Fasi di implementazione del metodo

In questa sezione vengono analizzate le fasi da seguire per riproporre il metodo proposto in altri scenari applicativi, quindi come progettare al meglio la base di conoscenza per schematizzare tutte le informazioni del dominio applicativo al fine di dare al modello tutte le informazioni per poter rispondere alle domande poste. Come si nota nello schema seguente le fasi da seguire sono le seguenti: Individuazione delle Regole, Progettazione dell'ontologia, Instanziazione della base di conoscenza, Ricerca delle domande frequenti, progettazione delle Query e Progettazione dei Guardrails. Nei paragrafi seguenti verrà illustrato per ogni fase cosa andare a produrre per applicare il metodo.

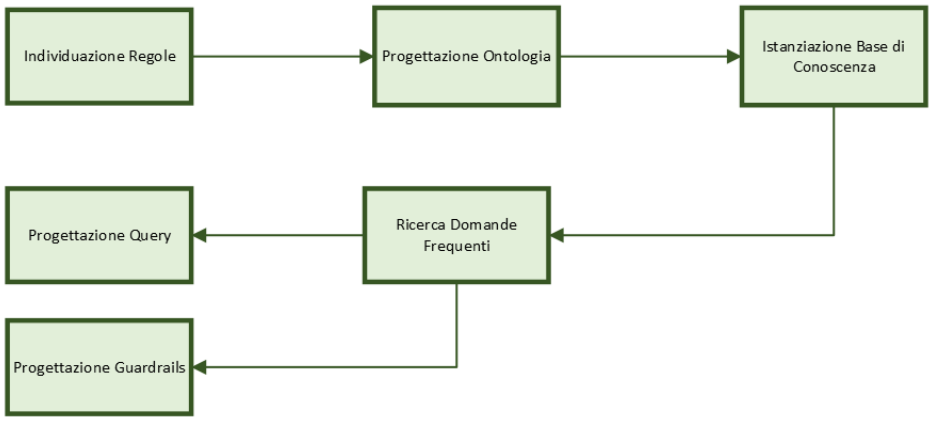

Figura 4: Fasi di implementazione

3.1 Individuazione Regole

La prima fase dell'implementazione del modello proposto è l'analisi del dominio applicativo, con particolare attenzione all'individuazione delle regole e dei vincoli che guidano il comportamento del robot assistenziale. Queste regole rappresentano conoscenza di alto livello, in quanto definiscono principi generali e vincoli comportamentali senza entrare nei dettagli tecnici dell'implementazione.

Queste regole sono fondamentali per garantire che il robot operi in modo coerente con gli obiettivi del sistema e con le esigenze dell'utente. Esse vengono definite in fase di progettazione attraverso l'analisi del dominio e possono essere affinate sulla base delle osservazioni sul campo, assicurando che l'assistenza fornita sia adeguata ed efficace, rappresentano la base per la progettazione dell'ontologia al fine di strutturare al meglio i dati.

3.1.1 Esempio:

A titolo di esempio il metodo è stato implementato sullo scenario riguardante un sistema robotico che assiste dei pazienti con determinate patologie e quindi degli stili alimentari da seguire e con determinare intolleranze, nello scegliere giornalmente i cibi più adatti. Inoltre al fine di una alimentazione sana è necessario variare il più possibile le scelte mantenendo sempre i limiti imposti dallo stile alimentare, quindi dall'analisi dello scenario sono state scelte le seguenti regole:

- Il paziente non può mangiare ciò che ha mangiato nei giorni precedenti con il limite di una settimana
- Il paziente non può mangiare ciò a cui è intollerante
- il paziente in base alla sua patologia deve rispettare un massimo numero di calorie giornaliere e un massimo numero di macronutrienti per pasto

3.2 Progettazione Ontologia

La seconda fase del metodo proposto riguarda la progettazione dell'ontologia, elemento fondamentale per strutturare le informazioni presenti nella base di conoscenza. L'ontologia definisce formalmente le entità coinvolte nel dominio applicativo, le loro proprietà e le relazioni esistenti tra di esse. La progettazione richiede un'analisi approfondita delle regole individuate nel modulo precedente, al fine di identificare le categorie concettuali essenziali e le loro interconnessioni. Dalle regole bisogna astrarre le entità e le proprietà necessarie affinché l'ontologia copra tutti i vincoli e tutte le informazioni sulla conoscenza, quindi si individuano le relazioni che collegano le entità.

3.2.1 Esempio:

In relazione all'esempio proposto e alle regole individuate prima sono state individuate le seguenti entità:

Ricetta, Macronutriente, Stile Alimentare, Patologia, Persona, Intolleranza

E le seguenti relazioni:

contiene(Ricetta, Macronutriente), ha_mangiato(Persona, Ricetta), ha_intolleranza(Persona, Intolleranza), vieta(Intolleranza, Ricetta), vincola(Stile Alimentare, Macronutriente), segue(Patologia, Stile Alimentare), ha_patologia(Persona, Patologia).

Si può notare che da questa struttura ogni regola è stata schematizzata in quanto esiste la relazione ha mangiato con i suoi attributi che garantisce lo storico dei pasti così da poter evitare la ripetizione settimanale di essi, l' entità Intolleranza con la relazione vieta contengono le informazioni affinchè questi piatti non vengano proposti all'utente, le entità patologia e stile alimentare contengono i dati necessari al garantire il vincolo sui macronutrienti e le calorie.

3.3 Istanziazione Base di Conoscenza

La terza fase del metodo proposto riguarda l'implementazione della base di conoscenza, che rappresenta un'istanza concreta dell'ontologia progettata nel modulo precedente. Mentre l'ontologia definisce la struttura concettuale del dominio applicativo, la base di conoscenza ne fornisce una rappresentazione popolata con dati reali, permettendo al sistema di elaborare informazioni specifiche sui pazienti e sulle loro esigenze nutrizionali.

Per l'implementazione della base di conoscenza è stato scelto Neo4j, un database a grafo che si adatta perfettamente alla modellazione delle entità e delle relazioni definite nell'ontologia. La scelta di Neo4j è motivata dalla sua capacità di gestire in modo efficiente strutture di dati complesse, consentendo interrogazioni rapide e flessibili tramite Cypher, il suo linguaggio di query dichiarativo. Grazie alla

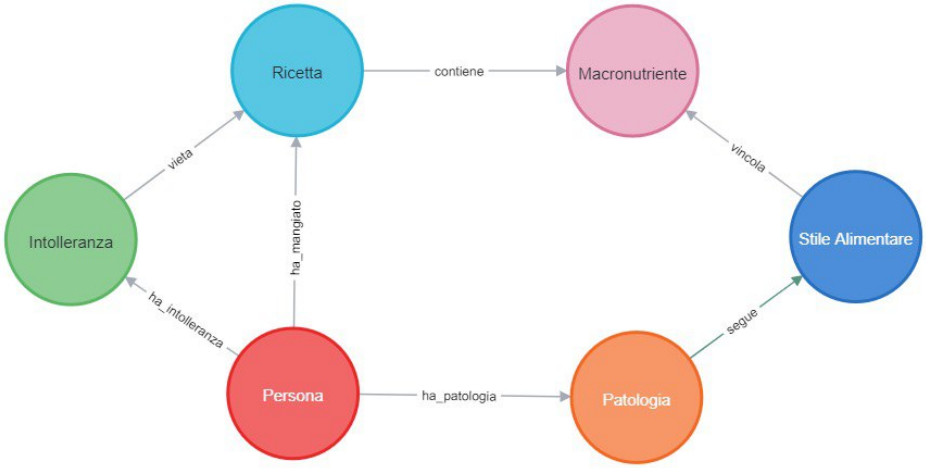

Figura 5: ontologia

rappresentazione a grafo, è possibile navigare facilmente tra le informazioni, ottimizzando il processo decisionale del robot.

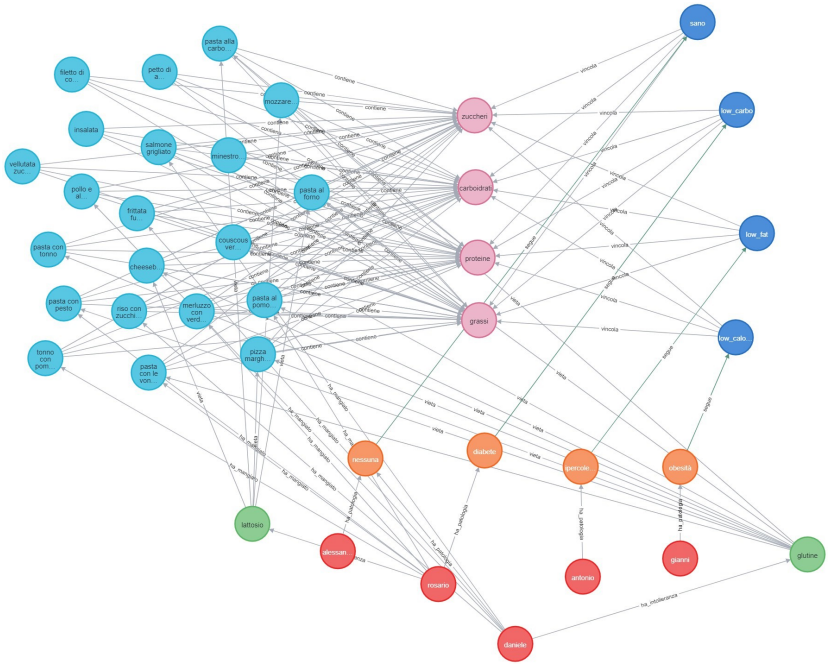

Figura 6: Esempio Base di Conoscenza

3.4 Ricerca Domande Frequenti

Nella prossima fase del metodo, si analizza il dominio di riferimento per identificare le possibili domande che un utente potrebbe rivolgere al sistema robotico, garantendo una copertura completa dei casi possibili ed evitando ripetizioni. Poiché il modello è in grado di riconoscere il significato di una frase indipendentemente dalla sua formulazione, non sarà necessario includere varianti che esprimono lo stesso concetto. Ad esempio, le domande "Sono Rosario, cosa posso mangiare oggi?" e "Sono Rosario, ho fame, che ricetta mi consigli?" veicolano lo stesso significato, ovvero la richiesta di una ricetta compatibile con le caratteristiche alimentari di Rosario. Di conseguenza, sarà sufficiente considerare una sola di queste formulazioni, poiché la query generata dal modello sarà identica in entrambi i casi.

Analogamente, non sarà necessario fornire una domanda per ogni possibile istanza di un'entità, ma basterà includere un solo esempio rappresentativo. Ad esempio, le domande "Quali ricette può mangiare una persona intollerante al glutine?" e "Quali ricette può mangiare una persona intollerante al lattosio?" fanno entrambe riferimento all'entità Intolleranza. Pertanto, sarà sufficiente fornire una sola di queste formulazioni, lasciando che la query generata dal modello si adatti in base alla specifica intolleranza considerata.

3.5 Progettazione Query

La progettazione delle query risulta la fase cardine di questo metodo in quanto la successiva generazione della spiegazione si basa su di esse, è quindi importante il modo in cui vengono generate al fine dell'Explainability. Per lo sviluppo delle query si sono attenzionati tre aspetti importanti:

- la struttura delle query
- i nomi delle variabili
- l'output restituito

La struttura della query, è molto importante in quanto all'interno di essa vengono applicate le regole e i vincoli analizzati prima, Nell'ontologia si è curata la schematizzazione dei dati da salvare all'interno della base di conoscenza, adesso bisogna prenderli e confrontarli in maniera ottimale. A questo fine si è deciso di strutturare le query in sezioni, ognuna delle quali riguarda un vincolo, quindi vengono analizzati i dati in modo da selezionare solo quelli che rientrano nei vincoli, successivamente si passa alla sezione successiva analizzando il vincolo successivo.

Si è deciso di suddividere le query in questo modo al fine di ottenere una spiegazione sequenziale che analizzasse passo passo il ragionamento fatto per selezionare i dari corretti, l'LLM che andrà a generare la risposta finale dovrà quindi trasformare la query in un discorso di senso compiuto analizzando passo passo le sezioni.

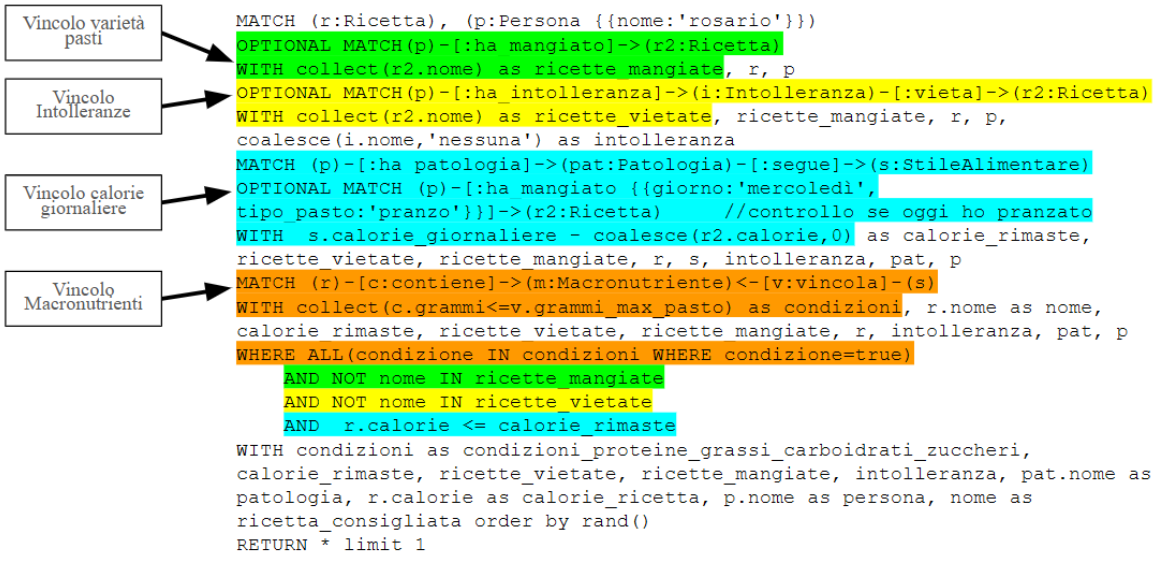

Figura 7: Esempio struttura query

La seconda scelta progettuale è stata relativa ad i nomi delle variabili all'interno delle query. Le query, come anticipato prima, saranno analizzate nel modulo finale di generazione della risposta da un LLM, nasce quindi il bisogno di chiarezza delle variabili per essere facilmente interpretabili dal modello, per questo motivo si è scelto di dare nomi autoesplicativi e in italiano. Adottando questa accortezza ci si è accorti che il modello, fornendogli i valori di output della query e la query stessa riesce a dare una spiegazione esaustiva spiegando i singoli casi e il perché vengono rispettati i vincoli o meno.

```
MATCH (r:Ricetta), (p:Persona {{nome:'rosario'}})
OPTIONAL MATCH(p)-[:ha_mangiato]->(r2:Ricetta)
WITH collect(r2.nome) as ricette_mangiate, r, p
OPTIONAL MATCH(p)-[:ha_intolleranza]->(i:Intolleranza)-[:vieta]->(r2:Ricetta)
WITH collect(r2.nome) as ricette_vietate, ricette_mangiate, r, p, coalesce(i.nome,'nessuna') as
intolleranza
MATCH (p)-[:ha_patologia]->(pat:Patologia)-[:segue]->(s:StileAlimentare)
OPTIONAL MATCH (p)-[:ha_mangiato {{giorno:'mercoledì', tipo_pasto:'pranzo'}}]->(r2:Ricetta)
WITH  s.calorie_giornaliere - coalesce(r2.calorie,0) as calorie_rimaste, ricette_vietate,
ricette_mangiate, r, s, intolleranza, pat, p
MATCH (r)-[c:contiene]->(m:Macronutriente)<-[v:vincola]-(s)
WITH collect(c.grammi<=v.grammi_max_pasto) as condizioni, r.nome as nome, calorie_rimaste,
ricette_vietate, ricette_mangiate, r, intolleranza, pat, p
WHERE ALL(condizione IN condizioni WHERE condizione=true)
    AND NOT nome IN ricette_mangiate
    AND NOT nome IN ricette_vietate
    AND  r.calorie <= calorie_rimaste
WITH condizioni as condizioni_proteine_grassi_carboidrati_zuccheri, calorie_rimaste,
ricette_vietate, ricette_mangiate, intolleranza, pat.nome as patologia, r.calorie as
calorie_ricetta, p.nome as persona, nome as ricetta_consigliata order by rand()
RETURN * limit 1
```

Figura 8: Esempio nomi variabili

La terza scelta progettuale è stata legata all'output restituito dalle query, come prima analisi si potrebbe pensare che alla domanda l'output necessario sia esclusivamente la risposta, ma ai fini dell'Explainability si è scelto di restituire tutte le liste di dati che sarebbero stati necessari per spiegare al meglio il processo deduttivo.

```
MATCH (r:Ricetta), (p:Persona {{nome:'rosario'}})
OPTIONAL MATCH(p)-[:ha_mangiato]->(r2:Ricetta)
WITH collect(r2.nome) as ricette_mangiate, r, p
OPTIONAL MATCH(p)-[:ha_intolleranza]->(i:Intolleranza)-[:vieta]->(r2:Ricetta)
WITH collect(r2.nome) as ricette_vietate, ricette_mangiate, r, p, coalesce(i.nome,'nessuna') as
intolleranza
MATCH (p)-[:ha_patologia]->(pat:Patologia)-[:segue]->(s:StileAlimentare)
OPTIONAL MATCH (p)-[:ha_mangiato {{giorno:'mercoledì', tipo_pasto:'pranzo'}}]->(r2:Ricetta)
WITH  s.calorie_giornaliere - coalesce(r2.calorie,0) as calorie_rimaste, ricette_vietate,
ricette_mangiate, r, s, intolleranza, pat, p
MATCH (r)-[c:contiene]->(m:Macronutriente)<-[v:vincola]-(s)
WITH collect(c.grammi<=v.grammi_max_pasto) as condizioni, r.nome as nome, calorie_rimaste,
ricette_vietate, ricette_mangiate, r, intolleranza, pat, p
WHERE ALL(condizione IN condizioni WHERE condizione=true)
    AND NOT nome IN ricette_mangiate
    AND NOT nome IN ricette_vietate
    AND  r.calorie <= calorie_rimaste
WITH condizioni as condizioni_proteine_grassi_carboidrati_zuccheri, calorie_rimaste,
ricette_vietate, ricette_mangiate, intolleranza, pat.nome as patologia, r.calorie as
calorie_ricetta, p.nome as persona, nome as ricetta_consigliata order by rand()
RETURN * limit 1
```

Figura 9: Esempio valori di ritorno

3.6 Progettazione Guardrails

Per la progettazione dei guardrails vengono utilizzate diverse tecniche di *prompting* e viene strutturato un *system* in diverse parti, affinché l'LLM si comporti nel modo desiderato.

- **Persona-based Prompting**: viene detto all'LLM di impersonificare il robot Pepper che assiste una persona.
- **Esempi**: vengono forniti degli esempi sugli argomenti che possono trattare le domande dell'utente.
- **Condizioni**: vengono poi effettivamente posti i *guardrails* che decretano le condizioni finali, fornendo anche un esempio di risposta in modo da avere sempre uno schema fisso e poterlo analizzare sempre allo stesso modo.

- **Schema**: viene fornito lo schema del database a grafo, che contiene i nodi e le relazioni con i loro attributi.