

ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA  
CAMPUS DI CESENA

---

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA Corso di  
Laurea in Ingegneria e Scienze Informatiche

**Analisi delle funzionalità di ChatGPT  
nella progettazione e sviluppo  
di database relazionali**

**Relatore:**  
**Prof.ssa Annalisa Franco**

**Presentata da:**  
**Sara Romeo**

**Sessione I**  
**Anno Accademico 2024-2025**

*”...Non potete unire i puntini guardando avanti;  
potete unirli solo guardando indietro.  
Quindi dovete avere fiducia che i puntini si uniranno  
in qualche modo, nel vostro futuro. ...”*

*~ Steve Jobs*



# Sommario

La presente tesi esplora il potenziale di integrazione di ChatGPT all'interno delle diverse fasi di progettazione e sviluppo di DBMS relazionali, mettendo in luce i benefici che questa sinergia può portare, analizzando anche i limiti e possibili sviluppi futuri. Dopo una breve introduzione relativa al contesto della ricerca, si tratterà dei seguenti contenuti:

- **Capitolo 1:** Per poter comprendere a fondo l'importanza di ChatGPT come rivoluzione tecnologica con un impatto su vari aspetti della nostra società, si percorre la storia e i traguardi principali nel campo dell'intelligenza artificiale che hanno reso tutto ciò possibile. Dai concetti fondamentali di Machine Learning e Deep Learning, si esaminano i modelli che si occupano di linguaggio naturale, per poi soffermarsi su ChatGPT e le rivoluzioni che ha portato.
- **Capitolo 2:** Prendendo in analisi un ipotetico caso studio, si va ad analizzare la capacità del modello GPT di inferenza del linguaggio naturale, delle conoscenze tecniche e della capacità rappresentativa di concetti all'interno di diagrammi per lo sviluppo di DBMS relazionali.
- **Capitolo 3:** Dopo una prima fase di prompting con ChatGPT in relazione alle query richieste, si analizzano le potenzialità del modello all'interno di un applicativo per l'estrazione di informazioni, interrogando il database direttamente sui dati posseduti.

In base ai risultati emersi durante lo studio, si traggono infine le conclusioni.



# Indice

<b>Sommario</b>	<b>i</b>
<b>Introduzione</b>	<b>i</b>
<b>1 Dal Machine Learning alla generazione del linguaggio naturale</b>	<b>1</b>
1.1 Intelligenza Artificiale e Machine Learning . . . . .	2
1.1.1 Intelligenza Artificiale . . . . .	2
1.1.1.1 Definizione di AI . . . . .	2
1.1.1.2 Origini dell'AI . . . . .	3
1.1.2 Machine Learning . . . . .	4
1.1.2.1 Potenzialità dell'apprendimento automatico . . . . .	4
1.1.2.2 L'importanza dei dati . . . . .	5
1.1.3 Apprendimento e Reinforcement Learning . . . . .	6
1.1.3.1 Tecniche di apprendimento . . . . .	6
1.1.3.2 Fine-tuning . . . . .	7
1.2 Deep Learning e modelli più avanzati . . . . .	8
1.2.1 Reti Neurali Feedforward e Ricorrenti . . . . .	8
1.2.1.1 Neuroni alla base delle reti . . . . .	8
1.2.1.2 Reti Neurali Feedforward . . . . .	10
1.2.1.3 Reti Neurali Ricorrenti . . . . .	11
1.2.2 Deep Learning . . . . .	12
1.2.3 Modelli di Deep Learning Feedforward . . . . .	13
1.2.3.1 Multilayer Perceptron . . . . .	13
1.2.3.2 Convolutional Neural Network . . . . .	13

1.2.4	Architetture di RNN . . . . .	14
1.2.4.1	Seq2Seq[1] . . . . .	14
1.2.4.2	Neural Machine Translation . . . . .	15
1.3	Generazione di linguaggio naturale e LLM . . . . .	15
1.3.1	Analisi e generazione del linguaggio . . . . .	15
1.3.1.1	Tokenization, vocabolari e Word Embedding . . . . .	16
1.3.2	Transformer: un'architettura neurale innovativa . . . . .	16
1.3.2.1	Dalle RNN ai Transformer . . . . .	16
1.3.2.2	Attention . . . . .	18
1.3.2.3	Encoder e Decoder . . . . .	18
1.3.3	Large Language Model . . . . .	19
1.4	Il modello GPT . . . . .	20
1.4.1	Evoluzione di GPT . . . . .	20
1.4.1.1	GPT-1 . . . . .	20
1.4.1.2	GPT-2 e GPT-3 . . . . .	20
1.4.1.3	GPT specifici . . . . .	21
1.4.1.4	ChatGPT: GPT-3.5 e GPT-4 . . . . .	22
1.4.2	Prompt engineering . . . . .	25
1.4.2.1	Formulazione dei prompt . . . . .	25
<b>2</b>	<b>Integrare ChatGPT nella progettazione di database</b>	<b>27</b>
2.1	Presentazione del caso studio . . . . .	27
2.1.1	Ambito di studio . . . . .	27
2.1.2	Intervista e diagramma ER . . . . .	27
2.2	Progettazione concettuale . . . . .	32
2.2.1	Analisi dei requisiti e progettazione ERD . . . . .	32
2.2.1.1	Valutazione capacità di inferenza del linguaggio naturale . . . . .	32
2.2.1.2	Valutazione comprensione linguaggio tecnico: capacità di esecuzione diretta . . . . .	54
2.3	Progettazione logica . . . . .	70
2.3.1	Schema relazionale . . . . .	70

---

<b>3</b>	<b>Interrogazione di database in linguaggio naturale</b>	<b>73</b>
3.1	Elaborazione query SQL . . . . .	73
3.1.1	Schema relazionale completo: analisi cumulativa . . . . .	73
3.1.2	Schema relazionale completo: analisi singola . . . . .	78
3.1.3	Schema parziale . . . . .	81
3.1.3.1	Composizione filato . . . . .	82
3.1.3.2	Categorie . . . . .	83
3.1.3.3	Ordini . . . . .	84
3.2	Automatizzazione query SQL: dialogare con i dati . . . . .	91
3.2.1	Langchain . . . . .	91
3.2.2	Applicativo proposto . . . . .	92
3.2.2.1	Descrizione dell'applicativo . . . . .	92
3.2.2.2	Analisi delle query . . . . .	93
	<b>Conclusioni</b>	<b>97</b>
	<b>Bibliografia</b>	<b>99</b>
	<b>Ringraziamenti</b>	<b>103</b>





# Elenco delle figure

1.1	Schema apprendimento per rinforzo . . . . .	7
1.2	Neurone biologico . . . . .	8
1.3	Collegamenti di una FNN . . . . .	10
1.4	Rete neurale ricorrente a 1 livello dal libro <i>Python Machine Learning</i> [2] . . . . .	11
1.5	Architettura Transformer dal paper <i>Attention is all you need</i> [3] . . . . .	17
1.6	Vista semplificata dei sottolivelli di encoder e decoder . . . . .	18
2.1	ERD ideato in principio per il caso studio . . . . .	31
2.2	ERD produzione . . . . .	38
2.3	ERD fornitura filati . . . . .	38
2.4	ERD sconti utente . . . . .	39
2.5	ERD completo iniziale seconda conversazione GPT 3.5 . . . . .	44
2.6	Sezione di ERD per le vendite . . . . .	46
2.7	Sezione di ERD con associazione produzione . . . . .	47
2.8	Sezione di ERD acquisto di filato . . . . .	47
2.9	Disegno di GPT 3.5 del diagramma per gli utenti . . . . .	48
2.10	ERD completo finale seconda conversazione GPT 3.5 . . . . .	53
2.11	ERD bottomp-up: filato . . . . .	65
2.12	ERD bottomp-up: prodotto . . . . .	66
2.13	ERD bottomp-up: vendite e utente . . . . .	67
2.14	ERD bottomp-up: sconti . . . . .	67
2.15	ERD finale approccio bottom-up GPT 3.5 . . . . .	69

2.16 Schema relazionale finale . . . . .	72
--	----

# Elenco delle tabelle

1.1	Comparazione dei componenti neuronali . . . . .	9
1.2	Tabella di comparazione delle versioni di GPT estratta dall'articolo ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope [4] . . . . .	24



# Introduzione

Il rapido progresso dell'intelligenza artificiale (AI) e dell'elaborazione del linguaggio naturale (NLP) ha portato allo sviluppo di modelli linguistici sempre più sofisticati, precisi e versatili. L'AI generativa si riferisce a una classe di modelli di intelligenza artificiale in grado di creare nuovi dati sulla base di strutture apprese da dati esistenti. I modelli di intelligenza artificiale generativa si basano su tecniche di deep learning (DL) e reti neurali per analizzare, comprendere e generare contenuti il più possibile somiglianti a ciò che un essere umano sarebbe in grado di produrre.

In un contesto in cui l'informazione è una risorsa a valore crescente e la quantità e la complessità dei dati continuano ad aumentare in maniera esponenziale, diventa sempre più critico implementare sistemi informativi in grado di gestire, analizzare e interpretare l'ingente quantità di dati a disposizione.

Per realizzare un sistema informativo efficace, in grado di produrre in tempo utile le sintesi necessarie ai processi decisionali e di gestione di una determinata attività, sono fondamentali le basi di dati.

È quindi un componente essenziale all'interno del sistema informatico a supporto del sistema informativo di un ente un DBMS (Database Management Systems) che possa collezionare e svolgere operazioni sui dati in modo tempestivo, accessibile e completo.

I Database Management Systems (DBMS) relazionali sono da tempo il pilastro su cui si basa la gestione dei dati strutturati. Tuttavia, con l'aumento esponenziale del volume e della complessità dei dati, sorgono sfide sempre più impegnative ed è qui che l'integrazione di tecnologie innovative come ChatGPT assume un ruolo

di svolta.

ChatGPT è un modello avanzato di intelligenza artificiale sviluppato da OpenAI che offre la possibilità di interagire con i dati in modo più naturale e intuitivo. Grazie alla sua capacità di comprensione del linguaggio naturale, ChatGPT può facilitare l'accesso ai dati e alle informazioni all'interno delle basi di dati rivoluzionando l'accesso all'informazione.

Attraverso casi di studio, esempi pratici e analisi approfondite, questa tesi mira a valutare come l'integrazione di ChatGPT possa essere un catalizzatore per l'evoluzione dei DBMS relazionali, consentendo loro di rimanere all'avanguardia nell'era dell'informazione digitale.

# Capitolo 1

## Dal Machine Learning alla generazione del linguaggio naturale

Nel vasto panorama dell'Intelligenza Artificiale (AI), il Machine Learning ha svolto un ruolo chiave nell'aprire la strada a una serie di straordinarie innovazioni. Tra queste, una delle più sorprendenti è stata la capacità di generare linguaggio naturale in modo coerente e contestualmente rilevante. Questo processo, ha rivoluzionato molteplici settori, dall'assistenza virtuale alla creazione di media generati dall'AI, ridefinendo le interazioni umane con la tecnologia.

Al centro di questa rivoluzione si trova la famiglia di modelli di linguaggio basati sui transformers, in particolare il modello GPT (Generative Pre-trained Transformer) che spicca per la sua versatilità e potenza. ChatGPT è il chatbot sviluppato da OpenAI che rende queste tecnologie accessibili all'utenza pubblica ed è lo strumento tramite il quale si eseguirà l'analisi in oggetto.

In questo capitolo, esploreremo l'evoluzione che ha portato dal Machine Learning alla generazione del linguaggio naturale.



## 1.1 Intelligenza Artificiale e Machine Learning

### 1.1.1 Intelligenza Artificiale

#### 1.1.1.1 Definizione di AI

Per introdurre correttamente l'Intelligenza Artificiale (tradotto dall'inglese Artificial Intelligence - AI), è utile partire dalle diverse definizioni di intelligenza per spiegare come si inserisce in questo contesto.

L'intelligenza si può definire come il complesso di facoltà psichiche e mentali che consentono all'uomo di pensare, comprendere o spiegare i fatti o le azioni, elaborare modelli astratti della realtà, intendere e farsi intendere dagli altri, giudicare, e lo rendono insieme capace di adattarsi a situazioni nuove e di modificare la situazione stessa quando questa presenta ostacoli all'adattamento.[5] Quindi possiamo affermare che in generale l'intelligenza, può essere definita come la capacità di comprendere, apprendere, ragionare e risolvere problemi in modo efficiente adottando determinati comportamenti in base alle conclusioni raggiunte. Tuttavia possiamo identificare tre prospettive principali:

- Intelligenza razionale: è la capacità di elaborare informazioni complesse e risolvere problemi in modo logico. È infatti strettamente correlata alla capacità di risolvere problemi matematici, scientifici e logici. Il ragionamento deduttivo ne è alla base, in quanto implica il raggiungimento di conclusioni partendo da premesse generali.
- Intelligenza adattativa: è l'abilità di adattarsi all'ambiente circostante e apprendere da nuove esperienze. Questa si può considerare l'applicazione del ragionamento induttivo, il quale da osservazioni specifiche permette di trarre conclusioni generali.
- Intelligenza comportamentale: viene espressa dall'interazione efficace con l'ambiente circostante. È spesso associata anche alla comprensione sociale e di comunicazione; usa un ragionamento di tipo abduttivo, ovvero osservando il risultato genera spiegazioni per via di un processo di inferenza.

Queste sono quindi le nozioni di intelligenza su cui si basa l'AI per ottenere una riproduzione parziale dell'attività intellettuale propria dell'uomo.

L'obiettivo delle tecnologie AI è quindi non solo rendere un algoritmo in grado di risolvere un determinato problema, ma vuole riprodurre il più possibile i processi cognitivi per capire il problema, apprendere e adattarsi nel tempo. Cerca quindi di tradurre tali concetti, attraverso modelli e tecnologie informatiche avanzate, in applicazioni pratiche. L'attività dell'AI è infatti presente in molteplici ambiti dal riconoscimento di testo, voce o immagini, assistenti virtuali, sistemi di guida semiautonoma, sistemi di previsione, domotica e molto altro ancora.

Possiamo affermare che questo sia uno dei campi più recenti e innovativi della scienza e dell'ingegneria e le nuove possibili frontiere suscitano oggi ancora di più un forte interesse.

#### 1.1.1.2 Origini dell'AI

Sin dalle prime fasi della computazione ci si è chiesto se un computer potesse essere creato in grado di apprendere.

È il 1943 quando i due ricercatori Warren McCulloch e Walter Pitts proposero al mondo scientifico il primo neurone artificiale, cui seguì poi nel 1949 il libro di Donald Olding Hebb [6], psicologo canadese, grazie al quale vennero analizzati in dettaglio i collegamenti tra i neuroni artificiali e i modelli complessi del cervello umano.

I primi prototipi funzionanti di reti neurali arrivano già verso la fine degli anni '50 dove Alan Turing cercava di spiegare come un computer potesse comportarsi come un essere umano.[7] Ufficialmente il termine Artificial Intelligence esordì con il matematico statunitense John McCarthy nel 1956, quando un gruppo di scienziati e matematici si riunì presso il Dartmouth College. Ciò venne seguito dal lancio dei primi linguaggi di programmazione specifici per l'AI (Lisp nel 1958 e Prolog nel 1973).[8]

Intorno agli anni '80 e '90 del secolo scorso la ricerca passò attraverso una nuova stagione nota come "inverno dell'AI". Questo periodo si contraddistingue per un calo dei finanziamenti e della fiducia nelle sue effettive capacità tecnologiche,

durante il quale il progresso è stato rallentato. Tuttavia, nonostante le sfide presenti, sono state sviluppate importanti teorie e metodologie, come le reti neurali artificiali e gli algoritmi di apprendimento automatico.

Con l'avvento delle GPU (Graphics Processing Unit) si entra in una nuova fase di svolta; originariamente progettate per elaborare grafica 3D nei videogiochi negli anni 2000, le GPU hanno dimostrato di essere estremamente efficienti nell'esecuzione di calcoli paralleli, il che le ha rese ideali per l'addestramento di modelli di machine learning, che implicano una grande quantità di operazioni matematiche parallele. Questo ha permesso un'accelerazione sostanziale nei tempi di addestramento dei modelli e ha reso possibili applicazioni di AI precedentemente impensabili.

Con lo sviluppo di hardware moderno e del Deep Learning, oggi l'AI è sempre più integrata nella nostra vita quotidiana, alimentando una rivoluzione tecnologica che sta trasformando il modo in cui lavoriamo, viviamo e interagiamo con il mondo intorno a noi.

## 1.1.2 Machine Learning

### 1.1.2.1 Potenzialità dell'apprendimento automatico

Il campo del Machine Learning (ML), tradotto come apprendimento automatico, vuole risolvere una sfida cruciale nell'ambito dell'AI: come sviluppare software in grado di migliorare autonomamente le proprie prestazioni nello svolgere determinati task, imparando attraverso l'esperienza accumulata.[9]

Data la specificità dei problemi risolti dai modelli di ML, questi più comunemente rientrano nella categoria di intelligenza artificiale debole (weak AI).

In sintesi, l'“intelligenza artificiale debole” rappresenta sistemi computazionali che si esibiscono come se possedessero l'intelligenza umana, ma hanno una comprensione globale del mondo. Al contrario, l'“intelligenza artificiale forte” rappresenta i sistemi computazionali dotati di intelligenza umana.[10]

Tuttavia il ML è spesso considerato come una via verso la realizzazione di un'intelligenza artificiale forte (strong AI), poiché fornisce un metodo per creare

sistemi che possono apprendere e adattarsi in modo autonomo, anche se non è ancora in grado di raggiungere una vera intelligenza generale.

### 1.1.2.2 L'importanza dei dati

L'esperienza è l'insieme delle informazioni passate disponibili a chi apprende, e in questo contesto prende tipicamente la forma di dati digitalmente raccolti e resi disponibili all'analisi dei modelli.

Per “dati” si intende qualsiasi rappresentazione digitale di atti, fatti o informazioni e qualsiasi raccolta di tali atti, fatti o informazioni, anche sotto forma di registrazione sonora, visiva o audiovisiva[11]

È evidente quindi come alla base di un modello che sia funzionante e di alta qualità, vi sia una raccolta dati efficace e mirata, per poter svolgere un ruolo decisivo nelle fasi di configurazione e apprendimento.

Il trattamento e la convalida di dati possiamo affermare che non sia un problema nuovo o esclusivo del settore del ML. Vediamo infatti come in molte applicazioni aziendali, all'interno dei corrispettivi RDBMS (sistema di gestione per database relazionali), i dati sono organizzati in forma tabellare, nelle cui colonne troviamo gli attributi e nelle righe i record.

Similmente i dati vengono raccolti in dataset forniti di features (attributi) e record (data point), i quali a loro volta vengono suddivisi in sottoinsiemi con scopi differenti. Solitamente si creano tre sottoinsiemi distinti:

- **Set di Addestramento (Training Set)**: usato per addestrare il modello, rappresenta circa l'80% del dataset
- **Set di Validazione (Validation Set)**: impiegato per la taratura del modello, modificando parametri per identificare la configurazione migliore
- **Set di Test (Test Set)**: utile a valutare le prestazioni finali

Alcuni punti chiave per formare un dataset funzionale sono la qualità, la quantità, la rappresentazione e la varietà dei dati. Queste caratteristiche influenzano direttamente le prestazioni dei modelli e la loro capacità di adattarsi e generalizzare su nuovi dati.

### 1.1.3 Apprendimento e Reinforcement Learning

#### 1.1.3.1 Tecniche di apprendimento

Tra i vari metodi adottati per la fase di apprendimento riconosciamo quattro macro categorie principali di addestramento.

- **Supervisionato**

L'addestramento viene effettuato su un dataset etichettato, ovvero su un insieme di dati di cui si conoscono sia le caratteristiche di input che le corrispondenti etichette di output. Il modello ricrea quindi una mappatura tra input e output osservati in questa fase, per elaborare previsioni.

- **Non supervisionato**

In questo caso l'obiettivo è identificare pattern e correlazioni nei dati senza l'ausilio di etichette. È particolarmente efficace quando si desidera esplorare la struttura intrinseca dei dati senza pregiudizi o quando semplicemente non si dispone di etichette di output.

- **Semi supervisionato**[12]

Questa tecnica si colloca a metà strada tra le prime due citate; il dataset qui contiene dati parzialmente etichettati. In particolare, i dati etichettati forniscono informazioni dirette sui pattern e sulle relazioni nei dati, mentre i dati non etichettati possono essere utilizzati per estrarre pattern più ampi e complessi e per generalizzare meglio il modello su nuovi dati. Questa scelta si rivela utile specialmente quando l'etichettatura dei dati è costosa o difficile da ottenere, poiché consente di sfruttare al massimo anche i dati non etichettati disponibili ed è largamente adottato in una varietà di applicazioni.

- **Reinforcement Learning**

Nell'apprendimento rinforzato (reinforcement learning) il modello riceve feedback in forma di ricompense o punizioni in base alle azioni che intraprende, e utilizza questo feedback per regolare il proprio comportamento e migliorare nel tempo. L'ambiente, ovvero il contesto in cui l'agente apprende, invia a

quest'ultimo lo stato in cui si trova accompagnato da una ricompensa più o meno remunerativa in base alla correttezza dell'azione compiuta dall'agente.

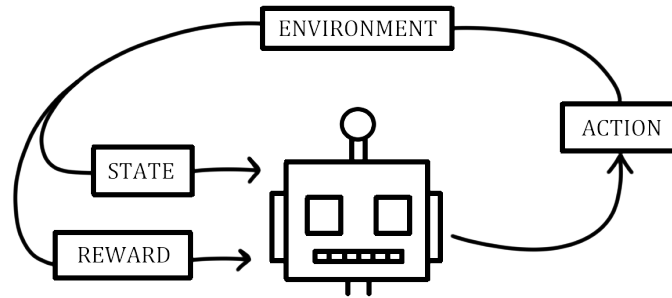


Figura 1.1: Schema apprendimento per rinforzo

L'obiettivo è quindi massimizzare una ricompensa cumulativa, facendo scelte adeguate in base allo stato corrente dell'ambiente. In questo modo, attraverso l'interazione con un ambiente dinamico, il modello viene allenato a prendere decisioni sequenziali.

Queste tecniche forniscono agli algoritmi la capacità di estrarre pattern, relazioni e informazioni significative dai dati, consentendo loro di fare previsioni, prendere decisioni e svolgere compiti complessi senza essere esplicitamente programmati.

### 1.1.3.2 Fine-tuning

Con la fase di fine-tuning si intende riallenare un modello già addestrato, viene effettuato per adattare il modello a un nuovo set di dati specifico per un nuovo dominio.

Risulta particolarmente utile per occuparsi di un compito molto specifico o il nuovo set di dati è di piccole dimensioni, risparmiando tempo e risorse di addestramento.

## 1.2 Deep Learning e modelli più avanzati

### 1.2.1 Reti Neurali Feedforward e Ricorrenti

#### 1.2.1.1 Neuroni alla base delle reti

McCulloch e Pitts nel 1943 svilupparono il primo modello di neuroni artificiali cercando di comprendere e simulare il comportamento del sistema nervoso degli animali. Vediamo quindi molto brevemente come funziona un neurone biologico, per poter comprendere meglio questo sviluppo tecnologico.

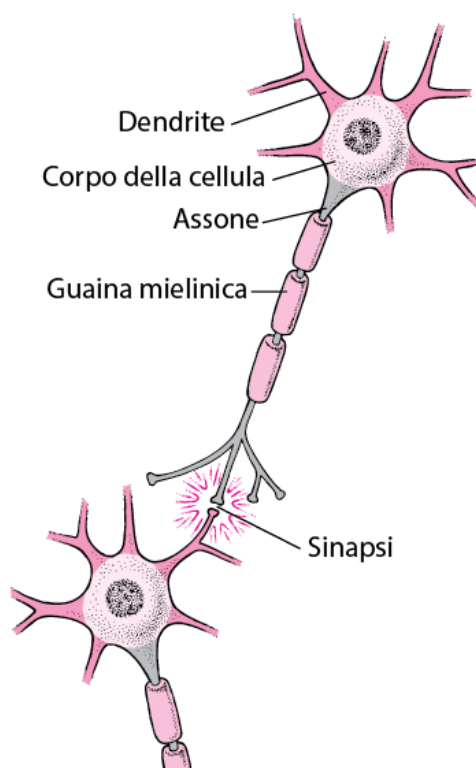


Figura 1.2: Neurone biologico

In un cervello biologico, i neuroni sono le cellule più importanti dell'intero sistema nervoso. Specializzate nella comunicazione attraverso le sinapsi, trasmettendo segnali elettrici e chimici, permettono il passaggio delle informazioni tra loro. Ogni neurone è dotato di un corpo cellulare (o soma), filamenti che ricevono segnali da

altri neuroni chiamati dendriti e un assone che trasmette segnali ad altri neuroni (Figura 1.2).

I neuroni artificiali (o nodi) si possono definire come modelli matematici semplificati dei neuroni biologici, progettati per eseguire operazioni di calcolo su dati di input. Un neurone artificiale riceve input ponderati da altri neuroni o da fonti esterne, li elabora utilizzando una funzione di attivazione e produce un output.

Le funzioni di attivazione introducono una non linearità nel modello, consentendo alla rete di approssimare funzioni complesse che non sarebbero altrimenti possibili con una combinazione lineare dei pesi. Ogni nodo ha una soglia di attivazione: questa, basandosi sull'input, determina se il neurone deve essere attivato o meno. Solo nel caso in cui l'input riesce a superare la soglia il neurone viene attivato e produce un output, altrimenti rimane inattivo.

I neuroni sono organizzati in strati (o livelli) e connessioni per formare reti neurali artificiali.

La corrispondenza tra neurone biologico e artificiale adottata la vediamo quindi sintetizzata nella tabella 1.1 a seguire.

<b>Biologico</b>	<b>Artificiale</b>
Corpo cellulare	Corpo cellulare
Assone	Canale di output
Dendriti	Canale di input
Sinapsi	Pesi

Tabella 1.1: Comparazione dei componenti neuronali

Sebbene si sia cercato di riprodurre artificialmente il funzionamento di un neurone nel modo più efficace possibile, c'è ancora molta ricerca da svolgere.

I neuroni artificiali sono estremamente versatili e possono essere utilizzati per una vasta gamma di compiti specifici, ma non possono ancora eguagliare la complessità e l'adattabilità dei neuroni biologici nel processare informazioni, apprendere da esperienze e adattarsi. Anche la velocità di elaborazione è una differenza sostanziale, inoltre mentre i neuroni biologici sono capaci di apprendimento conti-



nuo, i neuroni artificiali sono al contrario più rigidi e richiedono un addestramento specifico per apprendere da dati.

Per avanzare tecnologicamente nel campo delle reti neurali artificiali, sono necessari progressi significativi nella comprensione dei principi fondamentali del funzionamento del cervello, nonché nello sviluppo di algoritmi di apprendimento più avanzati e efficienti.

Nonostante le capacità del cervello umano siano lungi dall'essere raggiunte, la ricerca in corso nel campo dell'intelligenza artificiale, potrebbe portare a progressi significativi nell'avvicinare i neuroni artificiali alle loro controparti biologiche e nell'aprire nuove opportunità nell'elaborazione intelligente dei dati.

La struttura delle reti neurali artificiali è basata sulla suddivisione di gruppi composti uno o più neuroni in strati. Tipicamente si possono riconoscere un livello di input, uno o più livelli intermedi o nascosti (hidden layer) che si occupano dell'elaborazione e un livello di output. In particolare esistono due principali tipologie di reti, ovvero quelle feedforward e le reti neurali ricorrenti.

### 1.2.1.2 Reti Neurali Feedforward

Una Feedforward Neural Network (FNN) è letteralmente tradotta «rete neurale con flusso in avanti» e questo ne denota subito la caratteristica chiave: l'informazione da elaborare si muove in una sola direzione, ovvero in avanti.

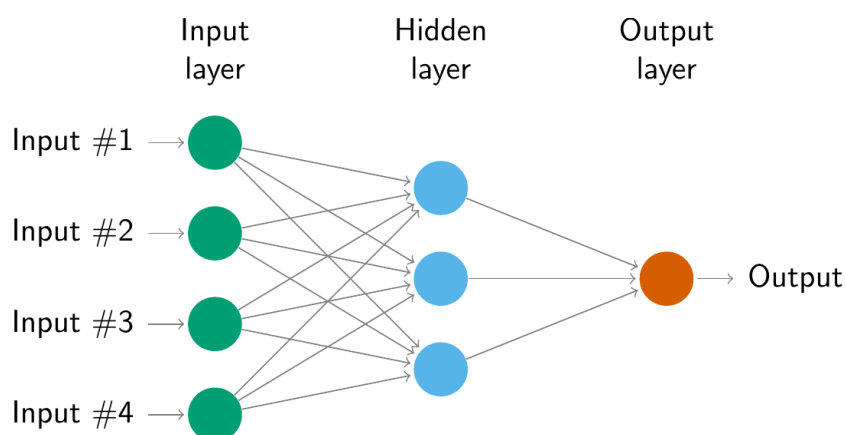


Figura 1.3: Collegamenti di una FNN

L'input viene inserito nel primo livello e, passando attraverso uno o più strati nascosti, raggiunge l'ultimo strato che produce l'output. Come si può denotare anche dalla figura 1.3 non sono consentite connessioni all'indietro o connessioni verso lo stesso livello.

È di gran lunga il tipo di rete più utilizzata ed è particolarmente utile nel momento in cui è richiesta un'elaborazione di apprendimento automatico. Ad esempio è efficace nei problemi di classificazione dove al modello è richiesto di riconoscere e dividere in classi degli elementi dati in input.

### 1.2.1.3 Reti Neurali Ricorrenti

L'elemento distintivo delle Recurrent Neural Network (RNN) rispetto alle FNN, è la capacità di elaborare dati in maniera sequenziale e quindi creare una specie di "effetto temporale". Con questo si intende un effetto di memoria (di breve termine) dei dati processati in passato durante l'elaborazione corrente dei dati.

Questo è possibile grazie ai collegamenti ciclici tra i nodi come si può evincere dalla figura 1.4 a seguire. Le connessioni sono quindi rivolte non solo in avanti come nelle FNN, ma anche verso neuroni dello stesso livello o del precedente.

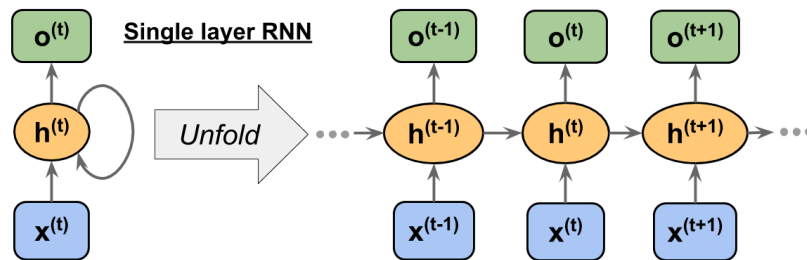


Figura 1.4: Rete neurale ricorrente a 1 livello dal libro *Python Machine Learning*[2]

Per superare la problematica della scomparsa del gradiente, e quindi dell'effetto di una memoria breve, presente nelle RNN, sono state introdotte le Long Short-Term Memory (LSTM).

La caratteristica chiave delle celle in queste architetture neurali sono i meccanismi di porta che filtrano cosa dimenticare (forget gate) e cosa considerare un input rilevante.

Sono state sviluppate anche le GRU (Gated Recurrent Unit), che riprendono la struttura delle celle LSTM, ma ne riducono parametri e complessità, cercando di mantenerne i vantaggi.

Le RNN sono ampiamente utilizzate per svolgere task che coinvolgono dati sequenziali quali il riconoscimento del parlato e l'elaborazione di testo linguaggio naturale, la traduzione automatica o per trattare media come audio e video, solo per citare alcuni esempi.

Questo complica notevolmente il flusso delle informazioni e l'addestramento, richiedendo di considerare il comportamento in più istanti temporali.

### 1.2.2 Deep Learning

Dopo aver esaminato brevemente le fondamenta delle reti neurali e le due principali tipologie di strutture dei collegamenti tra nodi (FNN e RNN), possiamo affermare che queste abbiano rivoluzionato il campo dell'apprendimento automatico.

Tuttavia, nel contesto del Deep Learning, le reti neurali hanno raggiunto nuovi livelli di complessità e performance, consentendo la creazione di modelli ancora più avanzati ed efficienti. Il Deep Learning è un insieme di tecniche e algoritmi utilizzati per l'apprendimento automatico su dati complessi.

Con il termine DNN (Deep Neural Network) si intendono reti neurali profonde, ovvero composte da molti livelli (almeno 2 hidden) organizzati in maniera gerarchica. Ciò significa nel FNN ad esempio una progressione nell'estrarre caratteristiche sempre più elaborate rispetto ai livelli precedenti, o nel caso delle RNN può essere una gerarchia attraverso il passaggio del tempo, attraverso i cicli di feedback tra gli strati.

Il Deep Learning si può quindi definire come una sotto categoria dell'intelligenza artificiale, che si occupa dell'uso di reti neurali profonde (DNN), tramite l'addestramento di modelli più complessi ed avanzati.

Al suo interno, il Deep Learning comprende una vasta gamma di modelli, tra cui le Multilayer Perceptron (MLP), le Reti Neurali Convolutionali (CNN) e molte altre varianti, alcune delle quali andremo ad analizzare a successivamente.

### 1.2.3 Modelli di Deep Learning Feedforward

#### 1.2.3.1 Multilayer Perceptron

Una rete neurale profonda feedforward o perceptrone multistrato (MLP) deve mappare un insieme di valori di input per restituire valori con una funzione matematica formata dalla composizione di molte funzioni più semplici in ogni strato.[13]

Una rete MLP ha almeno 3 livelli, tutti completamente connessi, di cui almeno 1 hidden, e applica le funzioni di attivazione dei nodi alla somma ponderata dei suoi input.

Tipicamente viene addestrato usando un approccio di apprendimento supervisionato. Nel 1986 Rumelhart, Hinton e Williams hanno introdotto l'algoritmo di Error Backpropagation (tradotto retropropagazione dell'errore)[14] che oggi è alla base della correzione dei pesi del modello MLP. La backpropagation consiste nel ripercorrere gli strati nella direzione opposta, calcolando il gradiente dell'errore rispetto ai pesi della rete e aggiornando i pesi in modo da minimizzare l'errore.

#### 1.2.3.2 Convolutional Neural Network

Il passaggio dal Multilayer Perceptron (MLP) alle Convolutional Neural Networks (CNN), soprattutto per applicazioni che coinvolgono l'elaborazione di immagini e compiti di visione artificiale, rappresenta un'importante evoluzione nel campo del Deep Learning.

Una CNN è una rete neurale profonda feedforward, con strati convoluzionali, strati di pooling e strati completamente connessi.[13]

Il termine "convoluzionale" deriva dal concetto di convoluzione, che è un'operazione matematica che combina due funzioni per produrne una terza. Nell'ambito delle CNN si riferisce all'operazione di convoluzione, applicata alle immagini di

input attraverso filtri digitali, ovvero maschere di pesi, con lo scopo di estrarre caratteristiche locali, senza perdere la loro struttura spaziale.

Con gli strati di pooling locale e/o globale la dimensionalità dei dati viene ridotta mediante un processo di sottocampionamento. Ciò significa che di tutte le informazioni contenute nelle feature map di input, solo quelle considerate più rilevanti vengono conservate. Questo processo serve a mantenere le informazioni significative ai fini della discriminazione dei pattern consentendo un risultato invariato rispetto a semplici trasformazioni dell'input.

## 1.2.4 Architetture di RNN

I modelli RNN, grazie alla loro caratteristica capacità di analizzare sequenze, si rivelano ottimi strumenti per l'elaborazione di linguaggio naturale. Si tratterà più ampiamente nella sezione successiva come questa tipologia di modelli si sia evoluta per dare spazio a nuove tecnologie di analisi e generazione del linguaggio come ChatGPT. Prendiamo in analisi comunque due architetture RNN precursori in quest'ambito: il Seq2Seq e il NMT.

### 1.2.4.1 Seq2Seq[1]

Il Sequence-to-Sequence (Seq2Seq)[1] è un'architettura di rete neurale, utilizzata principalmente per la traduzione automatica e altri compiti come riassunto di testi o generazione di sottotitoli, che si avvale di due componenti principali: un encoder e un decoder.

L'encoder è una RNN che prende in input una sequenza di parole estrapolate dal testo e ne cattura le informazioni rilevanti in un vettore di contesto. Quest'ultimo va al decoder, un'altra RNN che si occupa di generare l'output parola per parola finché non raggiunge la fine della frase.

Questo sistema encoder-decoder, ha contribuito a stabilire le fondamenta per molti modelli successivi, tra cui quelli basati su Transformer come ChatGPT, attore principale di questa tesi, che verrà descritto più nel dettaglio nelle prossime sezioni.

#### 1.2.4.2 Neural Machine Translation

La traduzione automatica neurale (NMT) è una variante dell'architettura Seq2Seq che si concentra specificamente sulla traduzione automatica sfruttando le RNN. Fondamentalmente, la NMT consiste in un'unica rete neurale profonda con milioni di neuroni che imparano a mappare direttamente le frasi sorgente in frasi target.[15]

Lo Statistical Machine Translation (SMT) è un approccio alla traduzione automatica che utilizza modelli statistici basati su regole linguistiche e allineamenti probabilistici per tradurre testi da una lingua di origine a una lingua di destinazione.

Rispetto a sistemi tradizionali di traduzione come le SMT, i modelli con architettura NMT si sono dimostrati nettamente migliori; la loro capacità di cogliere le dipendenze a lungo raggio nelle frasi, permette di contestualizzare correttamente la traduzione di frasi anche più complesse.

### 1.3 Generazione di linguaggio naturale e LLM

#### 1.3.1 Analisi e generazione del linguaggio

Il Natural Language Processing (NLP) e il Natural Language Understanding (NLU) rappresentano due aree complementari e interconnesse dell'intelligenza artificiale.

Se da un lato il NLP racchiude una vasta gamma di attività di analisi del linguaggio (ad esempio l'analisi sintattica o la traduzione automatica), il NLU si focalizza sull'interpretazione e quindi sulla comprensione semantica del contesto e del significato implicito delle frasi.

La Natural Language Generation (NLG) si focalizza sulla generazione automatica di testo di senso compiuto, che sia coerente e significativo per le richieste effettuate. La NLG si può dire che racchiude NLP e NLU in un unico ramo innovativo dell'AI.

In quest'ambito le RNN sono spesso utilizzate per la loro capacità di analisi sequenziale dei dati. Queste consentono ai modelli di apprendere e generare testo basandosi sequenzialmente sul contesto di parole e frasi.

### 1.3.1.1 Tokenization, vocabolari e Word Embedding

La segmentazione del testo in parti più piccole chiamate token è un processo fondamentale per rendere atomici i dati in input.

Nonostante l'apparente semplicità di questo passaggio, esistono diversi standard per la tokenizzazione del flusso di caratteri. I token possono corrispondere a singole parole, simboli (quali la punteggiatura e i caratteri) o sottostringhe, dipendendo dal tipo di applicazione e dallo standard scelto per la suddivisione del testo.

I token combinati tra loro consentono di creare un vocabolario utilizzato a sua volta dal modello.

Tra le tecniche esistenti il Byte Pair Encoding (BPE) (Gage, 1994)[16] è una semplice tecnica di compressione dei dati che sostituisce iterativamente la coppia di byte più frequente in una sequenza con un singolo byte non utilizzato. Adattiamo questo algoritmo per la segmentazione delle parole. Invece di unire coppie frequenti di byte, uniamo i caratteri o sequenze di caratteri.[17]

Infine vi sono tecniche di word embedding che raggruppano i token in base al loro significato per poter disporre efficacemente un vocabolario. Tra queste il Word2Vec, sviluppato nel 2013[18], rappresenta le parole come vettori e le categorizza in base alla similarità del significato associandole ad altri vettori situati vicini nello spazio.

## 1.3.2 Transformer: un'architettura neurale innovativa

### 1.3.2.1 Dalle RNN ai Transformer

Il primo tentativo di inserire il concetto di memoria nelle reti è avvenuto con le RNN, ma nonostante le evoluzioni con modelli come il LSTM per catturare

relazioni semantiche complesse e gestire sequenze di testo più lunghe con maggiore efficienza, queste presentano ancora delle limitazioni.

Le RNN utilizzano un tipo di allenamento sequenziale, ma non hanno un parametro vero e proprio di rilevazione dell'importanza del token in analisi. Inoltre, quando le sequenze di testo da analizzare sono particolarmente lunghe emergono ancor di più degli ostacoli.

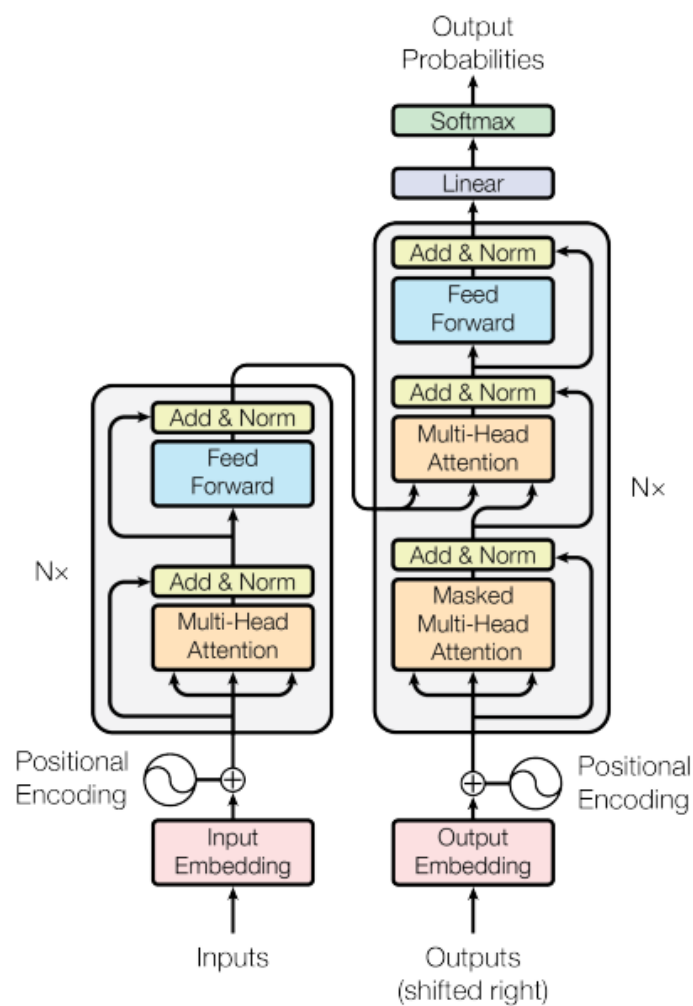


Figura 1.5: Architettura Transformer dal paper *Attention is all you need*[3]



Ed è qui che nel 2017, con l'introduzione dei Transformer da parte dei ricercatori di Google Brain nel paper "Attention is All You Need" [3] (Figura 1.5), vengono superate le limitazioni di RNN e LSTM, grazie a una nuova architettura basata sulla Multi-Head Attention.

### 1.3.2.2 Attention

Una funzione di attention è ciò che permette di parametrizzare l'importanza semantica del token ai fini della produzione di un output coerente.

Per ottenere una visione più ampia e al contempo più dettagliata sul piano semantico, un transformer utilizza più strati di attention in parallelo. Questo processo è definito come multi-head attention, combinando gli output delle "head", si ottiene un'attenzione complessiva che consente quindi di ottenere anche un "effetto temporale" più dilatato.

### 1.3.2.3 Encoder e Decoder

Alla base di un transformer vi è un encoder e un decoder che lavorano per codificare l'input ricevuto in una rappresentazione contestuale e decodificarlo in un output efficace.

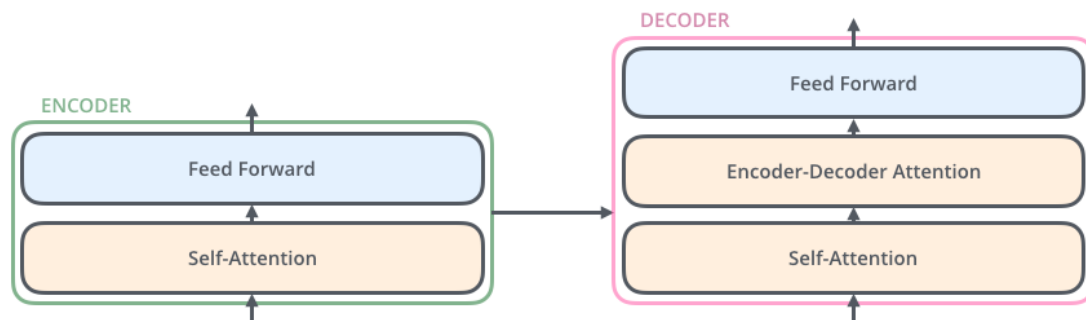


Figura 1.6: Vista semplificata dei sottolivelli di encoder e decoder

Un encoder ha  $N$  livelli, ognuno composto allo stesso modo da due sotto livelli incontrati dall'input entrante nel seguente ordine:

- **Self-attention:** per considerare le interazioni tra tutti i token in una sequenza durante il processo di encoding. Per farlo si calcolano il vettore di query, il vettore di chiave e il vettore di valore e se ne calcola il punteggio complessivo di attenzione rispetto alle altre della sequenza. Questo processo consente di focalizzare l'attenzione su parti specifiche della sequenza di input, utilizzando informazioni contestuali dalla sequenza di output generata fino a quel momento per indirizzare il processo di decodifica.
- **FNN:** per apprendere relazioni più complesse tra le parole, applica una trasformazione lineare ai vettori di output del livello di self-attention, seguita da una funzione di attivazione non lineare.

Prima della FNN il decoder si avvale, oltre della self-attention, anche della source-attention che si occupa di considerare le relazioni tra le parole nella sequenza di input originale.

Ogni sottolivello, sia nell'encoder che nel decoder, è a sua volta seguito da uno strato di normalizzazione.

### 1.3.3 Large Language Model

La possibilità di addestrare modelli su corpi di testo di dimensioni sempre più ingenti grazie al passaggio da RNN a Transformer ha portato alla nascita dei Large Language Model (LLM).

Negli ultimi anni sono stati sviluppati sempre più LLM che, sebbene i costi di addestramento siano piuttosto considerevoli, hanno ottenuto risultati sorprendenti.

Tra questi riconosciamo BERT e GPT che hanno svolto un ruolo decisivo nell'aprire le porte a una nuova era di modelli LLM più potenti e capaci.

Nel 2018 Google ha sviluppato BERT (Bidirectional Encoder Representations from Transformers) usato per classificare e comprendere il linguaggio.

Nello stesso anno OpenAI rilascia la prima versione di GPT (Generative Pre-trained Transformer), LLM costituito da uno stack di transformer, che si concentra sulla generazione di testo. GPT è addestrato a prevedere la parola successiva in una frase date le parole precedenti attraverso la modellazione linguistica.

## 1.4 Il modello GPT

### 1.4.1 Evoluzione di GPT

#### 1.4.1.1 GPT-1

È il 2018 quando OpenAI dimostra che si possono compiere grandi progressi su diversi task di NLU grazie al preaddestramento generativo di un modello linguistico su un corpus diversificato di testo non etichettato, seguito da un fine-tuning discriminativo su ciascun compito specifico.[19]

GPT-1 è la prima versione del modello GPT creata da OpenAI; con 117 milioni di parametri è stato uno dei primi modelli a dimostrare l'efficacia del preaddestramento nel campo dell'elaborazione del linguaggio naturale.

Si sceglie di sfruttare un approccio semi-supervisionato combinando 2 fasi:

1. **Preaddestramento non supervisionato**, per impostare i parametri iniziali
2. **Fine-tuning supervisionato**, dove il modello viene adattato a determinati task tramite l'uso di dati etichettati

Nonostante le innovazioni che ha portato, GPT-1 possiede comunque una comprensione del testo ridotta in quanto prone alla generazione di risposte tendenzialmente fuorvianti o incoerenti.

#### 1.4.1.2 GPT-2 e GPT-3

Tra il 2019 e il 2020 escono rispettivamente GPT-2 e GPT-3, entrambe evoluzioni del modello GPT, che affrontano le limitazioni della prima versione aumentando notevolmente il numero dei parametri e la loro complessità arrivando ai 175 miliardi di GPT-3.

Queste versioni sono in grado di generare testo ancora più lungo, mantenendo però maggiore coerenza dell'output rispetto al loro progenitore. Ciò avviene anche per merito della maggiore complessità che presentano le nuove architetture adottate.

### 1.4.1.3 GPT specifici

Nel 2022 vengono pubblicate altre versioni improntate verso task molto specifici quali InstructGPT, ProtGPT2 e BioGPT. Vediamo in breve di cosa si occupano:

- **InstructGPT**[20]: addestrato su una vasta gamma di materiale didattico per assistere nella creazione di contenuti educativi e quindi nell'insegnamento, fornendo spiegazioni chiare e comprensibili su argomenti complessi.
- **ProtGPT2**: addestrato su una grande quantità di sequenze di proteine per affrontare le sfide della predizione delle proteine e la modellazione della struttura delle proteine.
- **BioGPT**: addestrato, nell'ambito della bioinformatica e l'analisi di sequenze biologiche, su un vasto corpus di dati biologici, usato per la predizione della struttura delle proteine, l'analisi delle sequenze di DNA e RNA e la classificazione delle malattie genetiche.

I risultati sono stati molto positivi, ad esempio i biologi hanno ottenuto strumenti molto importanti per affiancare il loro lavoro. Nello specifico, un esemplare relativo a questi successi è ProtGPT, il quale riesce a generare sequenze di proteine realistiche e funzionali, in un contesto dove la comprensione delle proteine è cruciale per lo sviluppo di nuovi farmaci e trattamenti.

Ognuno di questi tre modelli può essere considerato un altro importante precursore di ChatGPT, in quanto hanno messo tutti in luce le potenzialità di queste nuove soluzioni AI non solo per questi settori, ma possibilmente anche per molti altri di diverso genere e hanno introdotto aggiustamenti utili a sviluppare le versioni successive.

Vediamo come ad esempio InstructGPT ha colmato un divario cruciale tra GPT-3 e GPT-3.5, grazie all'implementazione del RLHF (Reinforcement Learning from Human Feedback). Usare un RL corretto da un responso umano risulta essere più efficace per compiti con obiettivi complessi, mal definiti o difficili da specificare in termini matematici/logici.

#### 1.4.1.4 ChatGPT: GPT-3.5 e GPT-4

Il 30 novembre del 2022 viene pubblicato ChatGPT, un chatbot che permette di sfruttare le capacità del modello GPT-3.5 e a partire dal 2023 anche di GPT-4[21].

In particolare, ChatGPT utilizza il modello GPT-3.5-turbo, ottimizzato appositamente per le funzionalità di una chat. Le frasi generate risultano quindi ancora più complete delle precedenti varianti.

ChatGPT è stato uno dei più grandi e avanzati modelli di generazione del linguaggio, con 175 miliardi di parametri e capacità straordinarie di comprensione del contesto e generazione di testo coerente.

Per creare il proprio vocabolario e codificare le subword, GPT-3.5 usa il BPE[22]; ciò consente anche il supporto in diverse lingue dell'output, senza aumentare il quantitativo di token.

L'accesso gratuito al servizio da parte degli utenti ha creato diversi dibattiti sia sul piano della ricerca, suscitando interesse e attenzione, sia sulla responsabilità etica nell'uso di modelli così avanzati da parte e nei confronti di un'utenza così vasta e diversificata.

Infatti, dopo il rilascio iniziale, ChatGPT ha subito una serie di aggiornamenti e miglioramenti per affinare le sue capacità di generazione del linguaggio e affrontare problemi come la generazione di contenuti offensivi o fuorvianti.

Ciò avviene anche grazie all'approccio dei Rule-Based Reward Models (RBRM), che migliora le prestazioni e la sicurezza dei modelli linguistici, come GPT-4.[23]

GPT-4, dopo la prima fase di preaddestramento sui vasti corpus di testi in dataset, affronta il fine-tuning sia con RLHF che con modelli di ricompensa basati su regole (RBRM).

L'RBRM richiede tre input: il prompt (facoltativo), l'output del modello di policy e una rubrica scritta da esseri umani (ad esempio, un insieme di regole in stile a scelta multipla) su come valutare questo output. Quindi, l'RBRM classifica l'output in base alla rubrica.[23] I miglioramenti sono significativi, ma resta comunque ancora molta ricerca da compiere.

Il modello GPT-4, disponibile al pubblico non più gratuitamente, ma previo abbonamento mensile a ChatGPT Plus, ha portato a un livello successivo le po-

tenzialità di ChatGPT: non solo la precisione continua a incrementare riuscendo a gestire meglio anche prompt più complessi, ma l'altra caratteristica altrettanto decisiva è l'aggiunta capacità di interpretare immagini.

GPT-4 segna quindi il passaggio da un modello LLM a un Large Multimodal Model (LMM) proprio grazie a questa sua peculiarità di integrare più media accettati come input in uno stesso ambiente.

La tabella 1.2 nella pagina a seguire, mostra le principali differenze tra le versioni di GPT citate precedentemente.

Il modello GPT presenta ancora limitazioni importanti, quali i costi elevati sia ambientali che economici e il problema fondamentale delle "allucinazioni", ovvero la generazione di output fuorvianti e sbagliati, il che è più sottile rispetto alle prime versioni, ma comunque presente.

Nonostante ciò possiamo affermare che, ad oggi, ChatGPT e modelli simili hanno dato il via a un periodo di boom per l'AI, ovvero di rapido progresso, iniziando una vera e propria corsa da parte delle grandi aziende e degli stati per sviluppare e possedere le migliori tecnologie AI sul mercato.

Version	Uses	Architecture	Parameter count	Year
GPT-1	General	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax with Book Corpus: 4.5 GB of text	117 million	2018
GPT-2	General	GPT-1, but with modified normalization with Web Text: 40 GB of text	1.5 billion	2019
GPT-3	General	GPT-2, but with modification to allow larger scaling with 570 GB plaintext	175 billion	2020
InstructGPT	Conversation	GPT-3 fine-tuned to follow instructions using human feedback model	175 billion	2022
ProtGPT2	Protein Sequences	As GPT-2 large (36 layers) with Protein sequences from UniRef50 of total 44.88 million	738 million	2022
BioGPT	Biomedical Content	As GPT-2 medium (24 layers, 16 heads) with non-empty items from PubMed total 1.5 million	347 million	2022
ChatGPT	Dialogue	Uses GPT-3.5, and fine-tuned with both supervised learning and reinforcement learning from human feedback (RLHF)	175 billion	2022
GPT-4	General	Trained with both text prediction and RLHF and accepts both text and images as input, third party data	100 trillion	2023

Tabella 1.2: Tabella di comparazione delle versioni di GPT estratta dall'articolo ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope [4]

## 1.4.2 Prompt engineering

### 1.4.2.1 Formulazione dei prompt

Per raffinare il rapporto uomo-macchina, è bene creare prompt strutturati con l'intenzione di raggiungere il più alto livello di comprensione possibile per il modello e quindi ottenere un responso il più preciso possibile rispetto all'output richiesto.

Per scrivere un prompt efficace bisogna adattare la struttura al tipo di richiesta. In generale, per ottenere l'output desiderato, vi sono alcuni elementi che è consigliato esplicitare, a seconda dello scopo.

**Comando** Iniziare la richiesta con un verbo che indichi l'azione da compiere da parte del modello è un punto chiave per ottenere un risultato coerente con l'output desiderato

**Contesto** Descrivere, cercando di coprire i punti fondamentali, il contesto della richiesta: chi è il soggetto, in che ambiente si trova, e così via.

**Analogie** L'output può migliorare drasticamente se si fornisce uno o più esempi analoghi di ciò che si vorrebbe come risposta.

È infatti un'effettiva strategia di analisi di un modello testarlo con zero, one e few-shot prompting; rispettivamente si forniscono zero, una o alcune analogie relative all'output atteso.

**Chain-of-Thought** [24] Come è utile mostrare un esempio, lo è descrivere il pensiero critico che si vuole riprodurre per un determinato problema.

Scrivere il flusso di pensieri che porta ad un output, può risultare piuttosto vantaggioso; il modello cercherà di ripercorrere lo stesso processo e di riprodurlo.

**Autocritica** Un'idea per il debugging di risposte ottenute è provare a provocare una specie di autocritica nel modello, chiedendo di rianalizzare l'output generato ed esporre possibili problemi riscontrati.

**Formattazione** Non è fondamentale, ma può comunque migliorare la leggibilità e/o l'usabilità della risposta, richiedere una certa formattazione. Ad



esempio si potrebbe specificare di volere un elenco puntato, un blocco di codice o una tabella formattata in un certo modo.

**Identità e ruoli** Dichiarare un "personaggio" che l'AI dovrebbe impersonare; anche questo non è fondamentale, ma può cambiare il tono e la precisione della risposta ricevuta, basandosi sullo stile di una determinata identità fornita.

Oltre a definire l'identità del chatbot si può anche dichiarare che ruolo ha il pubblico/l'utente a cui la risposta è rivolta per suggerire un certo scenario.

**Tono** Richiedere direttamente di usare un certo tipo di formalità o anche emozione (ad es. pessimista, positivo, ecc.) può essere utile in alcuni casi. Per ottenere una descrizione del tono efficace, si potrebbe prima chiedere a ChatGPT alcune parole chiave/sinonimi di ciò che si vuole, per poi includere queste parole nell'effettivo prompt.

**Sintesi** Infine è fondamentale fornire tutte le informazioni necessarie rimanendo comunque il più concisi possibile; un prompt che non sia né troppo breve né troppo lungo è importante per evitare che l'output fuoriesca troppo dall'obiettivo richiesto.

Riuscire a formulare un buon prompt si può paragonare all'effettuare ricerche efficienti su Google: si studia e si acquisisce con la pratica il miglior modo di scrivere per risparmiare tempo e ottenere risultati precisi ed efficaci.

A seguire si prendono in analisi casi studio per analizzare alcuni possibili usi di ChatGPT nell'ambito della gestione delle basi di dati.

# Capitolo 2

## Integrare ChatGPT nella progettazione di database

### 2.1 Presentazione del caso studio

#### 2.1.1 Ambito di studio

Il caso di studio scelto riguarda la gestione di uno small business che si occupa di creare diversi prodotti usando la tecnica dell'uncinetto. L'obiettivo principale è la gestione dei filati acquistati a disposizione e delle vendite effettuate.

A seguire si valuta come ChatGPT possa essere utilizzato per assistere in varie fasi della progettazione di un database, dall'analisi dei requisiti e definizione dello schema fino all'ottimizzazione delle query.

#### 2.1.2 Intervista e diagramma ER

Per fornire il miglior contesto a ChatGPT si esplicitano i requisiti nel dettaglio in un'ipotetica intervista a seguire.

L'azienda CraftyCrocheters necessita di un sistema informativo per la gestione dei propri prodotti, delle transazioni con clienti e fornitori, degli utenti e relativi sconti di cui possono

beneficiare. Questo sarà utile a migliorare l'efficienza nella gestione delle operazioni, dalla tracciabilità alla produzione.

Alla base della produzione vi sono i filati: serve che siano registrati con dettagli precisi come il peso in grammi, la lunghezza in metri, la misura delle maglie, la misura dell'uncinetto e dei ferri, e la quantità disponibile in magazzino. Ogni filato inoltre è caratterizzato da una marca e un colore specifico. I filati possono essere composti da materiali diversi in determinate percentuali.

L'azienda collabora con diversi fornitori per l'approvvigionamento dei filati necessari, tenendo traccia delle relative fatture. Ogni fornitore deve essere registrato nel sistema con la partita IVA, denominazione sociale, dettagli di contatto (telefono, indirizzo, email) e ulteriori informazioni eventuali come il sito web.

I prodotti finiti sono basati su modelli specifici, i quali appartengono a determinate categorie e sottocategorie che ne definiscono la tipologia. Ogni modello deve essere registrato con un codice, un pattern (schema di lavorazione) scritto per esteso e deve permettere di registrare eventuali note utili aggiuntive. Di uno stesso modello esistono diverse versioni in base al colore e alla taglia (se applicabile). Un modello può essere effettivamente lavorato per la creazione di un prodotto oppure può essere riferito a un kit composto dai filati necessari alla creazione di un prodotto. Quindi l'azienda vende sia prodotti finiti sia kit per cucire le proprie creazioni.

Per incentivare gli acquisti, l'azienda offre sconti basati sulla quantità acquistata di un certo prodotto: superando una soglia minima si ottiene un preciso sconto. Esiste anche un

programma fedeltà, che prevede l'accumulo di punti per ogni utente che si registra. Ogni utente registrato dovrà fornire mail e password per effettuare l'accesso al sistema e controllare il proprio saldo di punti fedeltà, il quale può essere utilizzato per ottenere ulteriori buoni sconto di un certo importo. Esistono tre buoni in totale: con 20 punti accumulati si riceve un buono di 20 euro, con 40 uno da 50 euro, infine con 80 punti si ottengono 100 euro. Ogni 10 euro di spesa si guadagna un punto. I buoni sconti che vengono assegnati a ogni utente intitolato a riceverne uno sono univoci: ciò significa che sono destinati a un utente specifico e dotati di un codice che li rende monouso. Utenti registrati e non possono quindi effettuare un ordine per una o più versioni prodotto ed eventualmente usufruire dei propri buoni sconto.

Si ipotizzando delle operazioni richieste per il database in oggetto:

1. Visualizzare:

- 1.1 Filati composti da una certa percentuale di un determinato materiale
- 1.2 Fatturato vendite di questo mese (al netto di sconti quantità e buoni eventualmente applicati nelle vendite)
- 1.3 Categorie padre di una determinata sottocategoria, ordinando in base al livello di gerarchia
- 1.4 Vendite per ciascun modello con fatturato minore di 100 dell'ultimo mese, ordinando dal fatturato maggiore al minore (al netto degli sconti quantità, ma non dei buoni sconto che sono relativi all'ordine)
- 1.5 Versioni dei prodotti che non sono stati ordinati negli ultimi 6 mesi
- 1.6 Utenti che hanno ordinato prodotti di tutti i modelli
- 1.7 Tutte le categorie, riportando anche il numero di prodotti che eventualmente vi appartengono

1.8 L'utente con il maggior numero di ordini per ogni anno

2. Registrare:

2.1 Vendita a un utente non registrato (non c'è accumulo di punti) con controllo sconto quantità migliore

2.2 Vendita a un utente registrato (ogni 10 euro accumula un punto fedeltà) con controllo sconto quantità migliore

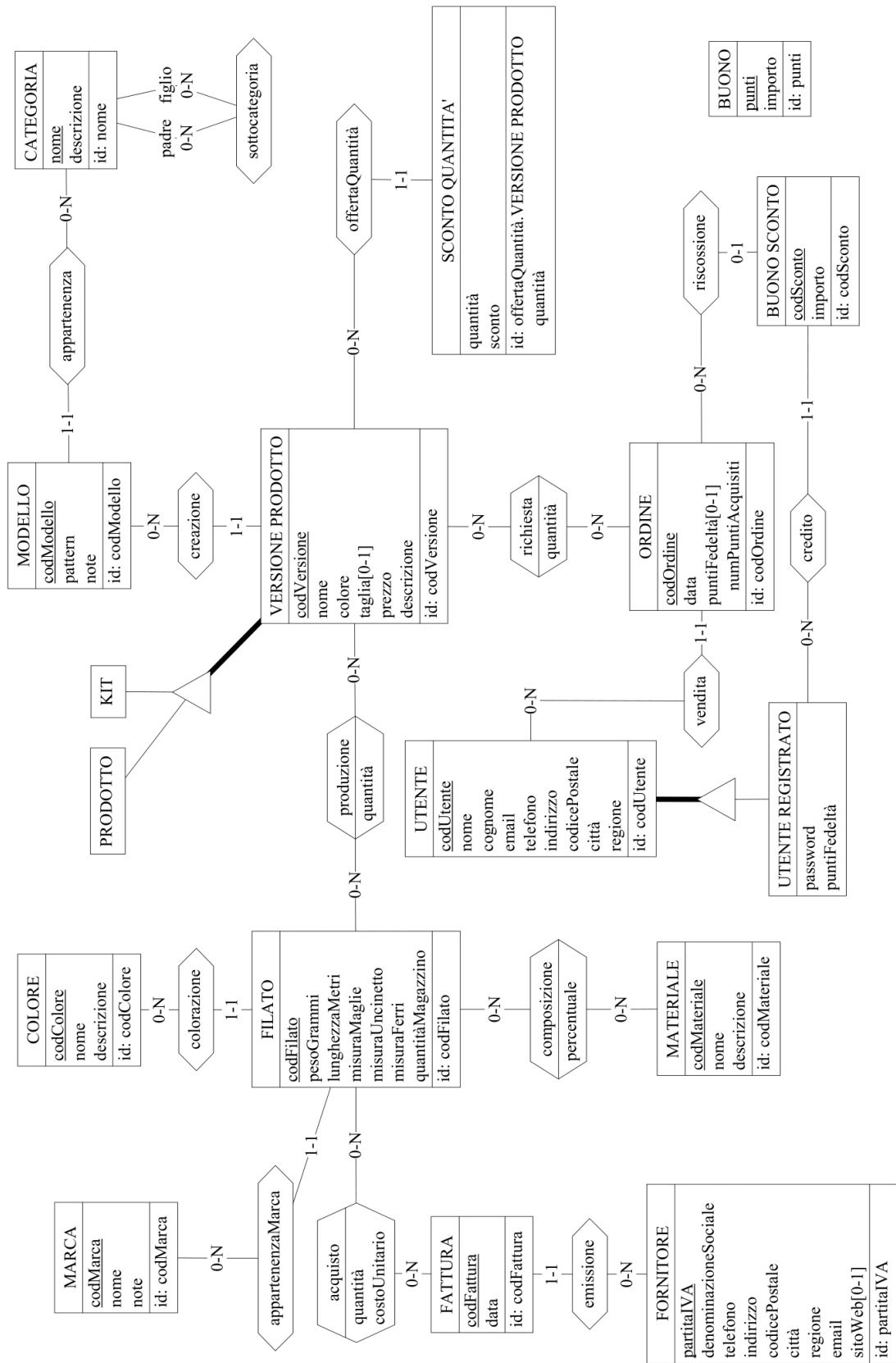


Figura 2.1: ERD ideato in principio per il caso studio

## 2.2 Progettazione concettuale

### 2.2.1 Analisi dei requisiti e progettazione ERD

#### 2.2.1.1 Valutazione capacità di inferenza del linguaggio naturale

In questa sezione si analizza sia la capacità di GPT 3.5 di deduzione ed estrazione dal linguaggio naturale dei concetti necessari a produrre uno schema ERD (Entity-Relationship Diagram), sia le sue conoscenze tecniche a riguardo.

#### Primo approccio

Nel primo prompt fornito viene riportata l'intera intervista come contesto per la creazione dell'ERD (Entity-Relationship Diagram):

Sei un database architect, leggi l'intervista e fornisci il relativo diagramma delle relazioni tra entità (ERD).

"L'azienda CraftyCrocheters necessita . . . propri buoni sconto."

Di seguito la risposta di ChatGPT:

Dal tuo racconto emergono diverse entità e le relazioni tra di esse. Ecco un diagramma delle entità e delle loro relazioni:

Descrizione delle entità:

1. **Filato:** Rappresenta i filati utilizzati per creare i prodotti. Contiene attributi come peso, lunghezza, misura delle maglie, misura dell'uncinetto, quantità in magazzino, marca, colore e composizione.
2. **Fornitore:** Rappresenta i fornitori che forniscono i filati. Contiene attributi come partita IVA, denominazione sociale, contatti e dettagli aggiuntivi.
3. **Fattura:** Rappresenta le transazioni di acquisto dei filati dai fornitori. Ogni fattura è collegata a un fornitore e può contenere più articoli di filati.
4. **Modello:** Rappresenta i modelli utilizzati per creare i prodotti. Contiene attributi come codice, pattern, note.
5. **Categoria:** Rappresenta le categorie a cui appartengono i modelli.
6. **Sottocategoria:** Rappresenta le sottocategorie a cui appartengono i modelli.
7. **Versione Prodotto:** Rappresenta le diverse versioni di un modello basate sul colore e sulla taglia.

8. **Prodotto**: Rappresenta i prodotti finiti. Ogni prodotto è collegato a una versione di un modello.
9. **Kit**: Rappresenta i kit composti dai filati necessari alla creazione di un prodotto. Ogni kit è collegato a una versione di un modello.
10. **Utente**: Rappresenta gli utenti registrati nel sistema. Contiene attributi come email e password.
11. **Punto Fedeltà**: Rappresenta i punti fedeltà accumulati dagli utenti.
12. **Buono Sconto**: Rappresenta i buoni sconto ottenuti dagli utenti in base ai punti fedeltà accumulati.

Relazioni:

- Un **Filato** può essere fornito da uno o più **Fornitori**.
- Una **Fattura** è associata a un singolo **Fornitore** e può contenere uno o più articoli di **Filati**.
- Un **Modello** appartiene a una **Categoria** e a una **Sottocategoria**.
- Un **Modello** può avere una o più **Versioni Prodotto**.
- Un **Prodotto** può essere associato a una **Versione Prodotto**.
- Un **Kit** può essere associato a una **Versione Prodotto**.
- Gli **Utenti** accumulano **Punti Fedeltà**.
- Gli **Utenti** possono ottenere **Buoni Sconto** in base ai **Punti Fedeltà** accumulati.

Questo diagramma fornisce una panoramica delle entità coinvolte e delle relazioni tra di esse nel sistema informativo proposto per CraftyCrocheters.

Sebbene la suddivisione sia apparentemente appropriata, andando ad approfondire si può notare come l'elenco sia carente di alcuni concetti centrali per modellare il problema, come la rappresentazione degli ordini delle vendite effettuate. Inoltre, il contenuto viene descritto concettualmente senza fornire indicazioni pratiche per la definizione del diagramma tecnico richiesto.

Per ottenere un primo schema scheletro, prima di raffinare il contenuto, si chiede maggiore dettaglio per riuscire a riprodurlo.



Fornisci in elenco tutti i dettagli utili per poter disegnare l'intero diagramma.

Si sceglie di non fornire indicazioni troppo specifiche per lasciare spazio alle scelte del modello per la risoluzione del problema e testarne le conoscenze in ambito di modellazione ERD. La risposta prodotta:

Certamente, ecco un elenco dettagliato di tutte le entità e dei loro attributi necessari per disegnare l'intero diagramma:

**Entità e Attributi:**

**1. Filato:**

- ID\_Filato (identificatore univoco)
- Peso (in grammi)
- Lunghezza (in metri)
- Misura\_Maglie
- Misura\_Uncinetto
- Quantità\_Disponibile
- Marca
- Colore
- Composizione

**2. Fornitore:**

- Partita\_IVA (identificatore univoco)
- Denominazione\_Sociale
- Telefono
- Indirizzo
- Email
- Sito\_Web

**3. Fattura:**

- Numero\_Fattura (identificatore univoco)
- Data\_Emissione
- Totale\_Ammontare

**4. Modello:**

- Codice\_Modello (identificatore univoco)

- Pattern
- Note

5. **Categoria:**

- ID.Categoria (identificatore univoco)
- Nome.Categoria

6. **Sottocategoria:**

- ID.Sottocategoria (identificatore univoco)
- Nome.Sottocategoria

7. **Versione Prodotto:**

- ID.Versione (identificatore univoco)
- Colore
- Taglia

8. **Prodotto:**

- ID.Prodotto (identificatore univoco)

9. **Kit:**

- ID.Kit (identificatore univoco)

10. **Utente:**

- ID.Utente (identificatore univoco)
- Email
- Password

11. **Punto Fedeltà:**

- ID.Punto (identificatore univoco)
- Quantità.Punti

12. **Buono Sconto:**

- Codice.Buono (identificatore univoco)
- Importo

**Relazioni:**

- Un **Filato** è fornito da uno o più **Fornitori**.
- Una **Fattura** è associata a un singolo **Fornitore** e può contenere uno o più articoli di **Filati**.

- Un **Modello** appartiene a una **Categoria** e a una **Sottocategoria**.
- Un **Modello** può avere una o più **Versioni Prodotto**.
- Un **Prodotto** è associato a una **Versione Prodotto**.
- Un **Kit** è associato a una **Versione Prodotto**.
- Gli **Utenti** accumulano **Punti Fedeltà**.
- Gli **Utenti** possono ottenere **Buoni Sconto** in base ai **Punti Fedeltà** accumulati.

Questi dettagli ti daranno tutte le informazioni necessarie per disegnare un diagramma delle entità e delle relazioni (ERD) completo e accurato per il sistema informativo di CraftyCrocheters.

Nonostante l'output sia migliorato nella definizione degli attributi per ogni entità, le relazioni sono ancora prive di cardinalità, fattore determinante per la creazione del diagramma. Si chiede quindi al modello di esprimere tecnicamente anche questo concetto.

Fornisci tutte le cardinalità delle relazioni per disegnare il diagramma entity-relationship

Ecco le cardinalità delle relazioni tra le entità per disegnare il diagramma Entity-Relationship:

1. **Filato - Fornitore:**

- Filato può essere fornito da uno o più fornitori. (1:N)
- Un fornitore può fornire molti filati. (N:1)

2. **Fornitore - Fattura:**

- Un fornitore può emettere molte fatture. (1:N)
- Una fattura è emessa da un solo fornitore. (N:1)

3. **Filato - Fattura:**

- Una fattura può contenere uno o più filati. (1:N)
- Un filato può essere presente in una o più fatture. (N:M)

4. **Modello - Categoria:**

- Un modello appartiene esattamente a una categoria. (N:1)
- Una categoria può contenere molti modelli. (1:N)

**5. Modello - Sottocategoria:**

- Un modello appartiene esattamente a una sottocategoria. (N:1)
- Una sottocategoria può contenere molti modelli. (1:N)

**6. Modello - Versione Prodotto:**

- Un modello può avere molte versioni prodotto. (1:N)
- Una versione prodotto è associata a un solo modello. (N:1)

**7. Versione Prodotto - Prodotto:**

- Una versione prodotto può essere associata a molti prodotti. (1:N)
- Un prodotto è associato a una sola versione prodotto. (N:1)

**8. Versione Prodotto - Kit:**

- Una versione prodotto può essere associata a molti kit. (1:N)
- Un kit è associato a una sola versione prodotto. (N:1)

**9. Utente - Punto Fedeltà:**

- Un utente può accumulare molti punti fedeltà. (1:N)
- I punti fedeltà appartengono a un solo utente. (N:1)

**10. Utente - Buono Sconto:**

- Un utente può ricevere molti buoni sconto. (1:N)
- Un buono sconto è assegnato a un solo utente. (N:1)

Spero che queste informazioni ti aiutino a creare il diagramma Entity-Relationship! Fammi sapere se hai bisogno di ulteriori chiarimenti.

Con questo primo scambio si è testata la capacità del modello di interpretare un linguaggio completamente naturale e di rappresentare i concetti dedotti nel diagramma ER.

Nel tentativo di ricreare, il più fedelmente possibile, lo schema proposto da GPT 3.5, si può notare che lo schema fornito non è completamente connesso, bensì trattasi di 3 diagrammi non proprio correttamente separati (figure 2.2 - 2.3 - 2.4).

Ciò che risalta è una comprensione parzialmente limitata del testo fornito e uno sfruttamento ridotto del potere rappresentativo degli ERD. Nel definire le entità categoria e sottocategoria (figura 2.2), ad esempio sarebbe stato più efficiente usare

una relazione ricorsiva su categoria, in quanto le due entità identificano lo stesso concetto.

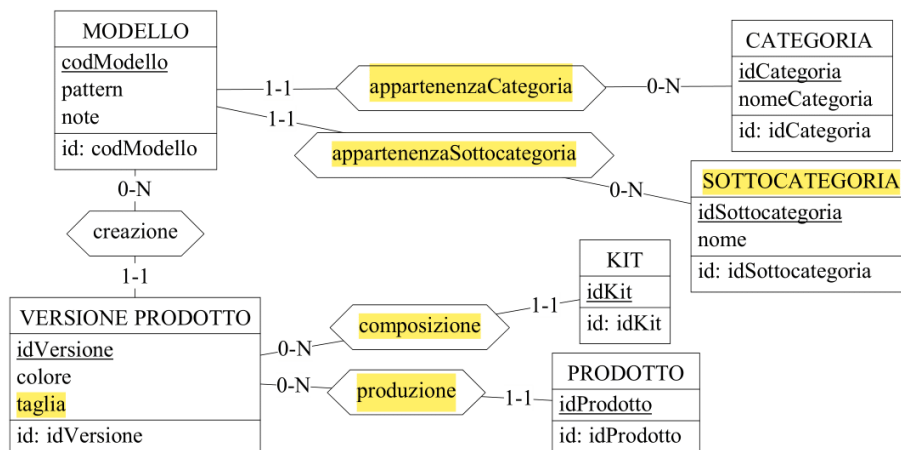


Figura 2.2: ERD produzione

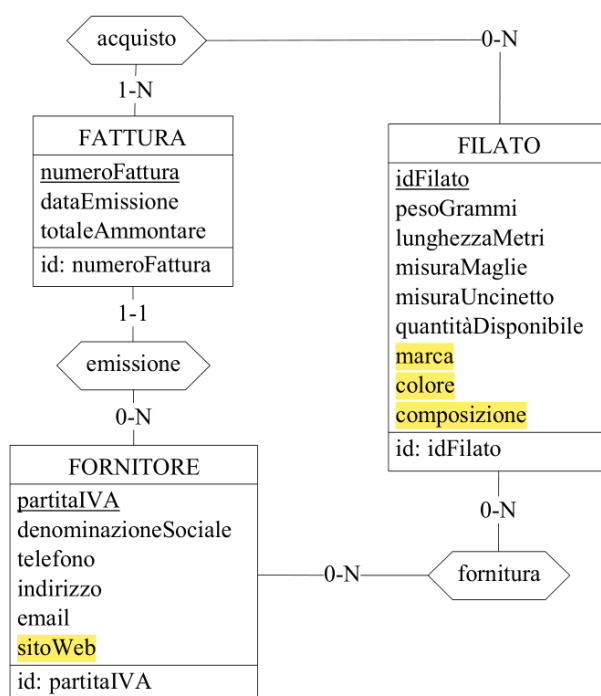


Figura 2.3: ERD fornitura filati

Un altro esempio, relativo invece alla comprensione del linguaggio, si può notare dallo schema in figura 2.4: i punti fedeltà posseduti sono un valore numerico che potrebbe essere direttamente assegnato all'utente come attributo anziché renderlo un'entità separata. Sarebbe infatti più appropriato, per esprimere al meglio la dipendenza funzionale esistente, creare un'entità che modella i tipi di buoni che si possono ottenere, ovvero quale sarebbe l'importo in base a quanti punti, e tenere il saldo dei punti come attributo di utente.

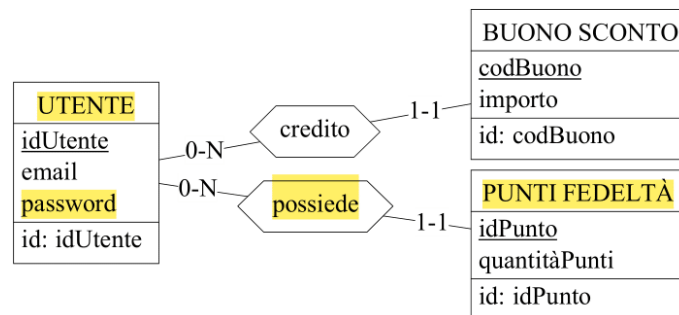


Figura 2.4: ERD sconti utente

## Seconda conversazione

Si prova quindi a iniziare una nuova conversazione, utilizzando una versione meno verbosa dell'intervista, con l'obiettivo di guidare maggiormente il modello nel processo di definizione del diagramma.

Sei un database architect e devi creare lo schema ERD. Estrai i concetti principali.

”I filati vanno registrati con il peso in grammi, la lunghezza in metri, la misura delle maglie, la misura dell'uncinetto e dei ferri, e la quantità disponibile in magazzino. Ogni filato inoltre è caratterizzato da una marca e un colore specifico. Un filato è composto da uno o più materiali diversi in determinate percentuali. I filati sono acquistati da diversi fornitori tenendo traccia delle relative fatture. Ogni fornitore deve essere registrato nel sistema con la partita IVA, denominazione sociale, dettagli di contatto (telefono, indirizzo, email) e ulteriori informazioni eventuali come il sito web.

Un modello appartiene a determinate categorie e sottocategorie. Ogni modello deve essere registrato con un codice, un pattern scritto per esteso e deve permettere di registrare eventuali note utili aggiuntive.

Ogni prodotto è relativo a un modello specifico. Un prodotto ha diverse versioni in base al colore e alla taglia (se applicabile). Un prodotto può essere venduto come prodotto finito oppure in formato kit. Sia il kit che il prodotto finito sono creati con uno o più filati.

Utenti registrati e non possono effettuare un ordine per una o più versioni prodotto ed eventualmente usufruire dei propri buoni sconto nel momento della vendita. Degli utenti si registrano tutti i dati anagrafici e di contatto utili e solo per chi è registrato anche la password e si conteggiano i punti fedeltà.

Esistono due tipi di buoni sconto: sconto quantità e buoni.

- Superata una soglia minima sulla quantità acquistata di un certo prodotto, si ottiene lo sconto quantità. Qualsiasi utente può ottenere questo sconto.
- Solo gli utenti registrati ogni 10 euro di acquisti accumulano 1 punto fedeltà. Esistono tre buoni in totale: con 20 punti accumulati si riceve un buono di 20 euro, con 40 uno da 50 euro, infine con 80 punti si ottengono 100 euro.

I buoni sconti che vengono assegnati a ogni utente intitolato a riceverne uno sono univoci: ciò significa che sono destinati a un utente specifico e dotati di un codice che li rende monouso.”

La risposta è molto più pertinente della precedente:

Per creare lo schema ERD (Entity-Relationship Diagram) per il sistema descritto, possiamo identificare i seguenti concetti principali:

1. **Filato:**

- **Attributi:**
  - Peso (grammi)
  - Lunghezza (metri)
  - Misura delle maglie
  - Misura dell’uncinetto
  - Misura dei ferri
  - Quantità disponibile in magazzino
- **Relazioni:**
  - Marca (1 - N)
  - Colore (1 - N)
  - Composizione (N - M) [Con i materiali e le relative percentuali]
  - Fornitore (N - 1)

- Fattura (N - M)

**2. Marca:**

- Attributi:
  - Nome

**3. Colore:**

- Attributi:
  - Nome

**4. Materiale:**

- Attributi:
  - Nome
- Relazioni:
  - Filato (N - M) [Con le percentuali di composizione]

**5. Fornitore:**

- Attributi:
  - Partita IVA
  - Denominazione sociale
  - Telefono
  - Indirizzo
  - Email
  - Sito web

**6. Fattura:**

- Attributi:
  - Numero
  - Data
  - Importo totale
- Relazioni:
  - Fornitore (1 - N)

**7. Modello:**

- Attributi:
  - Codice
  - Pattern



- Note

- Relazioni:

- Categoria (N - 1)
- Sottocategoria (N - 1)

8. **Categoria:**

- Attributi:

- Nome

9. **Sottocategoria:**

- Attributi:

- Nome

10. **Prodotto:**

- Attributi:

- Tipo (Finito o Kit)

- Relazioni:

- Modello (1 - N)

11. **Versione Prodotto:**

- Attributi:

- Colore
- Taglia

- Relazioni:

- Prodotto (1 - N)

12. **Ordine:**

- Attributi:

- Data

- Relazioni:

- Utente (1 - N)
- Versione Prodotto (N - M)

13. **Utente:**

- Attributi:

- Nome
- Cognome

- Indirizzo
- Email
- Telefono
- Password
- Punti fedeltà
- Relazioni:
  - Ordine (1 - N)
  - Buono Sconto (1 - N)

14. **Buono Sconto:**

- Attributi:
  - Codice
  - Tipo (Sconto quantità o Buono)
  - Valore

15. **Sconto Quantità:**

- Attributi:
  - Soglia minima
  - Sconto percentuale

16. **Punto Fedeltà:**

- Attributi:
  - Quantità di acquisti
  - Punti accumulati

17. **Scaglioni Punto Fedeltà:**

- Attributi:
  - Punti necessari
  - Valore del buono

Questa struttura ERD cattura i concetti chiave del sistema e le relazioni tra di essi. Ogni entità è associata ad attributi e relazioni che riflettono i requisiti e le interazioni specificate nella descrizione.

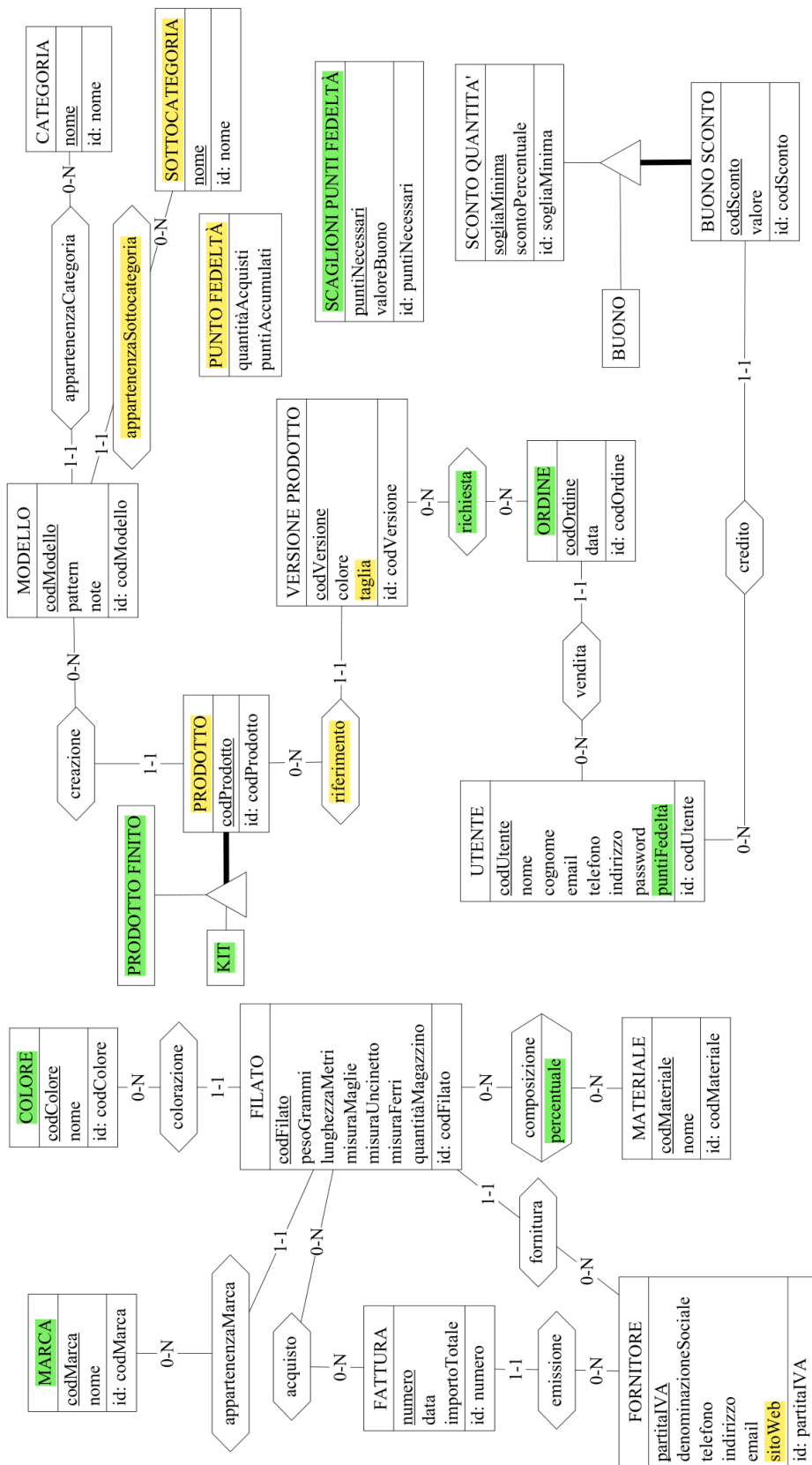


Figura 2.5: ERD completo iniziale seconda conversazione GPT 3.5

Nel raffigurare l'output (figura 2.5), per interpretare le tipologie di prodotti e sconti sono state usate delle gerarchie, mentre le chiavi primarie per semplicità sono state scelte arbitrariamente, in quanto non definite nella risposta.

Il diagramma ottenuto, sebbene ancora suddiviso in due sottodiagrammi, è molto più connesso di prima, introduce entità non individuate precedentemente e le particolarità non individuate e/o modellate sono in minor numero.

Alcune differenze positive rispetto all'output precedente:

- Per quanto riguarda i filati, sono state aggiunte le entità marca e colore, ma soprattutto la composizione è espressa nel modo più corretto: ponendo nella relazione l'attributo percentuale, è possibile rappresentare i diversi abbinamenti.
- Gli utenti non sono più scollegati, ma effettuano ordini contenenti una o più versioni prodotto
- Gli ordini delle vendite sono rappresentati
- I punti fedeltà accumulati sono accorpati nell'utente e vengono distinti dai scaglioni punti fedeltà che esprimono le tipologie di sconti in base ai punti posseduti
- I prodotti comprendono sia kit che prodotti finiti

Si potrebbe quindi affermare che sintetizzando ed eliminando buona parte delle parole non utili alla risoluzione del problema, il risultato è decisamente più calzante rispetto all'output desiderato.

Tuttavia vi è ancora qualche concetto mancante, per citarne alcuni:

- L'associazione che collega il filato utilizzato per la produzione di una determinata versione di prodotto non esiste
- Non viene rappresentata la riscossione eventuale dei buoni posseduti
- Kit e prodotti finiti fanno riferimento a versioni di prodotto specifiche e non a prodotti, in quanto creati con filati specifici a seconda della versione.

- Lo sconto quantità è strettamente correlato alla versione di prodotto: è la quantità di una certa versione venduta che determina lo sconto
- Il prodotto (che sia kit o lavorato), viene creato con un unico modello di riferimento, perciò la cardinalità sarebbe 1:1 e si potrebbe accorpare il prodotto nel modello stesso togliendo un'entità che risulterebbe ridondante
- Solo gli utenti registrati possiedono dei buoni e la password per accedere
- Il costo unitario e la quantità di filato acquistato, considerando che il costo può variare ad ogni transazione
- Il prezzo di vendita e la quantità venduta

Si continua la conversazione chiedendo di correggere in base a quest'elenco dei concetti mancanti senza accennare a tecnicismi relativi agli ERD.

Di tutta la lista sembra aver colto solo due aspetti:

- è stata aggiunta l'eventuale riscossione dei buoni, insieme agli attributi prezzoVendita e quantitàVenduta all'interno dell'ordine (figura 2.6)

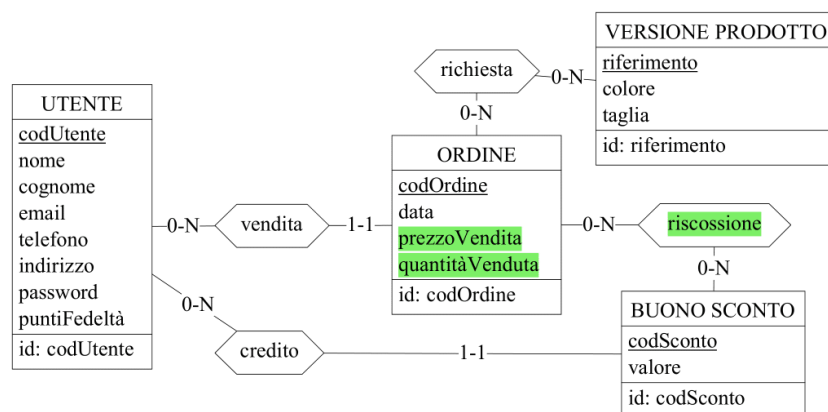


Figura 2.6: Sezione di ERD per le vendite

Essendo i due attributi riferiti strettamente alla versione di prodotto, si sarebbero dovuti rappresentare all'interno dell'associazione 'richiesta'.

- risulta presente l'associazione produzione con la quantità di filato usata (figura 2.7)

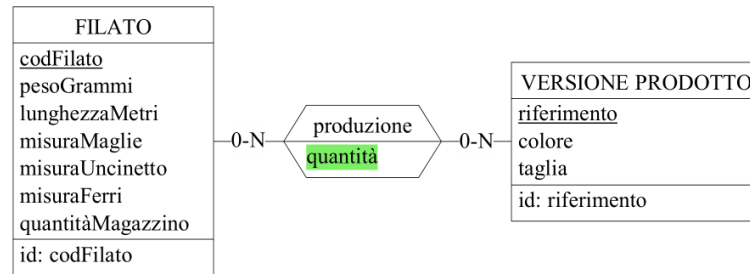


Figura 2.7: Sezione di ERD con associazione produzione

Si prova ad utilizzare un prompt di autocritica prendendo in esame solo uno dei punti richiesti non implementati:

Come hai modellato il costo unitario e la quantità di filato acquistato?

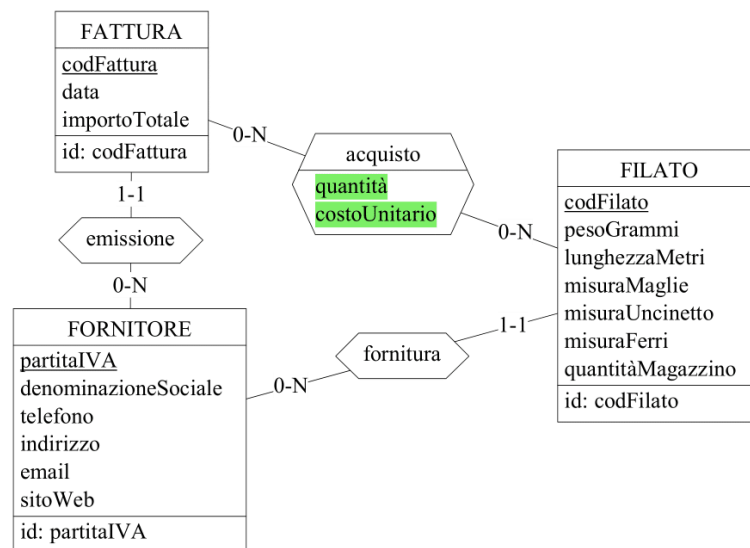


Figura 2.8: Sezione di ERD acquisto di filato

Andando a riesaminare il suo stesso output, ChatGPT è stato in grado di effettuare un'autocorrezione appropriata, aggiungendo gli attributi all'associazione acquisto (figura 2.8).

Si reitera lo stesso procedimento per ogni punto non modellato.

Per quanto riguarda la distinzione di utenti registrati e non, GPT sceglie di aggiungere un'entità e, quando richiesto di collocarle all'interno dello schema prova a disegnare l'ER in esame in questo formato:

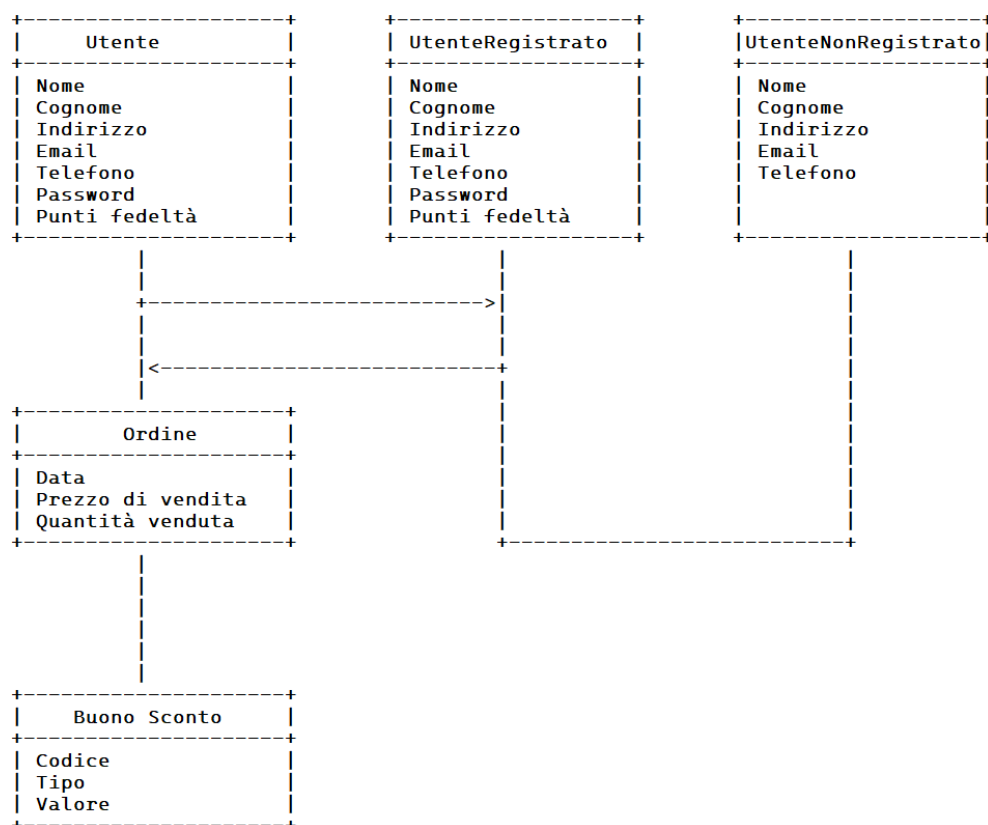


Figura 2.9: Disegno di GPT 3.5 del diagramma per gli utenti

Non sembra quindi conoscere le gerarchie o considerarne l'utilizzo, ma pare intuire che l'utente registrato e non sono due sottoinsiemi di utente generico.

Dopo ogni iterazione completata si chiede sempre di riscrivere l'elenco completo dei concetti prima di procedere con nuove informazioni, per evitare un'eccessiva perdita di memoria. È interessante notare come, da questo momento in poi, ri-proponga più volte la versione disegnata dello schema piuttosto che l'elenco dei concetti; si è scelto comunque di guidarlo verso la produzione di elenchi in quanto più leggibili e dall'interpretazione più chiara dei processi logici che affronta nell'elaborare le risposte.

Nel modellare il fatto che kit e prodotti finiti sono due tipi versioni di prodotto specifiche e non legate ai prodotti, sembra aver riscontrato nuovamente delle difficoltà. Per questo motivo, oltre alla domanda di autoanalisi, è stato anche offerto un esempio di caso d'uso come segue: "due filati di lana gialla servono sia per fare il cappello giallo che per fare il kit che l'utente userà per produrselo da solo".

La modifica della cardinalità tra prodotto e modello è stata eseguita, gli sconti quantità sono stati collegati alle versioni prodotto ed è stato aggiunto un campo tipo (kit o finito) anche per le versioni prodotto. Da quest'ultimo punto si deduce che continua a dimostrare una certa difficoltà a gestire tipologie di entità molto simili, equiparabili o raggruppabili.

Considerando questo risultato nel complesso quasi accettabile, si prova a chiedere in modo generico, sempre sfruttando l'autoanalisi, cosa si potrebbe fare per ottimizzare lo schema.

La prima risposta consiglia strategie in maniera molto generale:

Per ottimizzare lo schema, possiamo prendere in considerazione diverse strategie:

**1. Normalizzazione del Database:**

- Assicuriamoci che lo schema sia normalizzato per eliminare la ridondanza dei dati e migliorare l'integrità dei dati.

**2. Indicizzazione:**

- Aggiungiamo indici alle colonne frequentemente utilizzate nelle operazioni di ricerca, ordinamento e filtraggio per migliorare le prestazioni delle query.

**3. Denormalizzazione Selettiva:**

- In alcuni casi, è possibile denormalizzare lo schema per migliorare le prestazioni delle query, soprattutto se ci sono query complesse e pesanti in termini di utilizzo delle risorse. Tuttavia, bisogna bilanciare questo con la necessità di mantenere l'integrità dei dati.

**4. Utilizzo di Vincoli di Integrità Referenziale:**

- Assicuriamoci che siano applicati vincoli di integrità referenziale in modo appropriato per mantenere la coerenza dei dati tra le tabelle.

**5. Partizionamento dei Dati:**

- Se il volume dei dati è molto elevato, possiamo considerare il partizionamento delle tabelle per migliorare le prestazioni e la gestibilità del database.



**6. Ottimizzazione delle Query:**

- Esaminiamo le query più frequenti e ottimizziamole per ridurre i tempi di esecuzione. Possiamo aggiungere o modificare gli indici, riscrivere le query o ridisegnare lo schema per meglio supportare le operazioni richieste.

**7. Utilizzo di Viste Materializzate:**

- Se ci sono query complesse e pesanti che vengono eseguite frequentemente, possiamo creare viste materializzate per memorizzare i risultati di queste query e migliorare le prestazioni.

**8. Archiviazione Ottimizzata dei Dati:**

- Archiviamo i dati storici o meno frequentemente utilizzati in modo ottimizzato, ad esempio utilizzando partizionamento dei dati o archiviazione compressa.

**9. Cache delle Query:**

- Utilizziamo una cache delle query per memorizzare i risultati delle query frequentemente eseguite e ridurre il carico sul database.

**10. Monitoraggio delle Prestazioni:**

- Monitoriamo regolarmente le prestazioni del database e apportiamo le modifiche necessarie per ottimizzarle.

Rivedendo lo schema ERD, potremmo valutare l'opportunità di eventuali denormalizzazioni selettive o di ottimizzazioni delle tabelle, a seconda del carico previsto e delle esigenze di accesso ai dati. Ad esempio, potremmo valutare se l'introduzione di viste materializzate o l'ottimizzazione delle query potrebbero migliorare le prestazioni del sistema.

Se hai qualche preferenza specifica su quale aspetto ottimizzare o se hai bisogno di ulteriori dettagli su una particolare strategia, fammelo sapere!

Anche provando a specificare di ottimizzare lo schema di cui si è trattato i risultati continuano ad essere proposte di strategie, ma sono più focalizzate sul contesto. In particolare, GPT si è accorto che prodotto e modello rappresentano lo stesso concetto e ha proposto quindi di mapparli in un'unica entità, ovvero prodotto.

Ecco alcune possibili ottimizzazioni per migliorare lo schema:

**1. Denormalizzazione selettiva:**

- Considerando che ogni prodotto è associato esattamente a un modello, possiamo denormalizzare l'entità 'Prodotto' e incorporare gli attributi del modello direttamente all'interno dell'entità 'Prodotto'. Questo ridurrebbe la complessità dello schema e semplificherebbe le query, evitando la necessità di unire 'Prodotto' e 'Modello' ogni volta che è necessaria una visualizzazione congiunta.

**2. Partizionamento dei Dati:**

- Se il volume dei dati diventa considerevole, possiamo considerare di partizionare le tabelle, ad esempio 'Fattura' o 'Ordine', per migliorare le prestazioni e la gestibilità del database. Le tabelle possono essere partizionate in base a criteri temporali, come ad esempio archiviare i dati più vecchi in partizioni separate.

**3. Indicizzazione:**

- Aggiungiamo indici alle colonne frequentemente utilizzate nelle operazioni di ricerca, ordinamento e filtraggio. Ad esempio, possiamo aggiungere un indice sulla colonna 'Data' della tabella 'Ordine' per accelerare le ricerche basate sulla data.

**4. Ottimizzazione delle Query:**

- Analizziamo le query più frequenti e ottimizziamole per ridurre i tempi di esecuzione. Potremmo riscrivere le query o aggiungere/ottimizzare gli indici in modo da rendere le query più efficienti.

**5. Utilizzo di Viste Materializzate:**

- Se ci sono query complesse e pesanti che vengono eseguite frequentemente, possiamo creare viste materializzate per memorizzare i risultati di queste query e migliorare le prestazioni.

**6. Cache delle Query:**

- Utilizziamo una cache delle query per memorizzare i risultati delle query frequentemente eseguite e ridurre il carico sul database.

**7. Normalizzazione aggiuntiva:**

- Potremmo esaminare la possibilità di normalizzare ulteriormente lo schema, soprattutto se ci sono ridondanze di dati che possono essere eliminate.

**8. Archiviazione Ottimizzata dei Dati:**

- Archiviamo i dati storici o meno frequentemente utilizzati in modo ottimizzato, ad esempio utilizzando partizionamento dei dati o archiviazione compressa.

### 9. Utilizzo di vincoli di integrità referenziale:

- Assicuriamoci che siano applicati vincoli di integrità referenziale in modo appropriato per mantenere la coerenza dei dati tra le tabelle.

Ecco una possibile implementazione delle ottimizzazioni suggerite:

- Incorporare gli attributi del modello direttamente nell'entità 'Prodotto'.
- Aggiungere indici alle colonne frequentemente utilizzate nelle operazioni di ricerca, ordinamento e filtraggio.
- Partizionare le tabelle 'Ordine' e 'Fattura' in base alla data.
- Ottimizzare le query più frequenti e utilizzate.
- Creare viste materializzate per le query complesse e pesanti.
- Utilizzare una cache delle query per memorizzare i risultati delle query frequentemente eseguite.

Con queste ottimizzazioni, lo schema sarà più efficiente in termini di prestazioni e manutenibilità.

Applicata quest'ultima ridefinizione, a seguire lo schema finalizzato (figura 2.10):

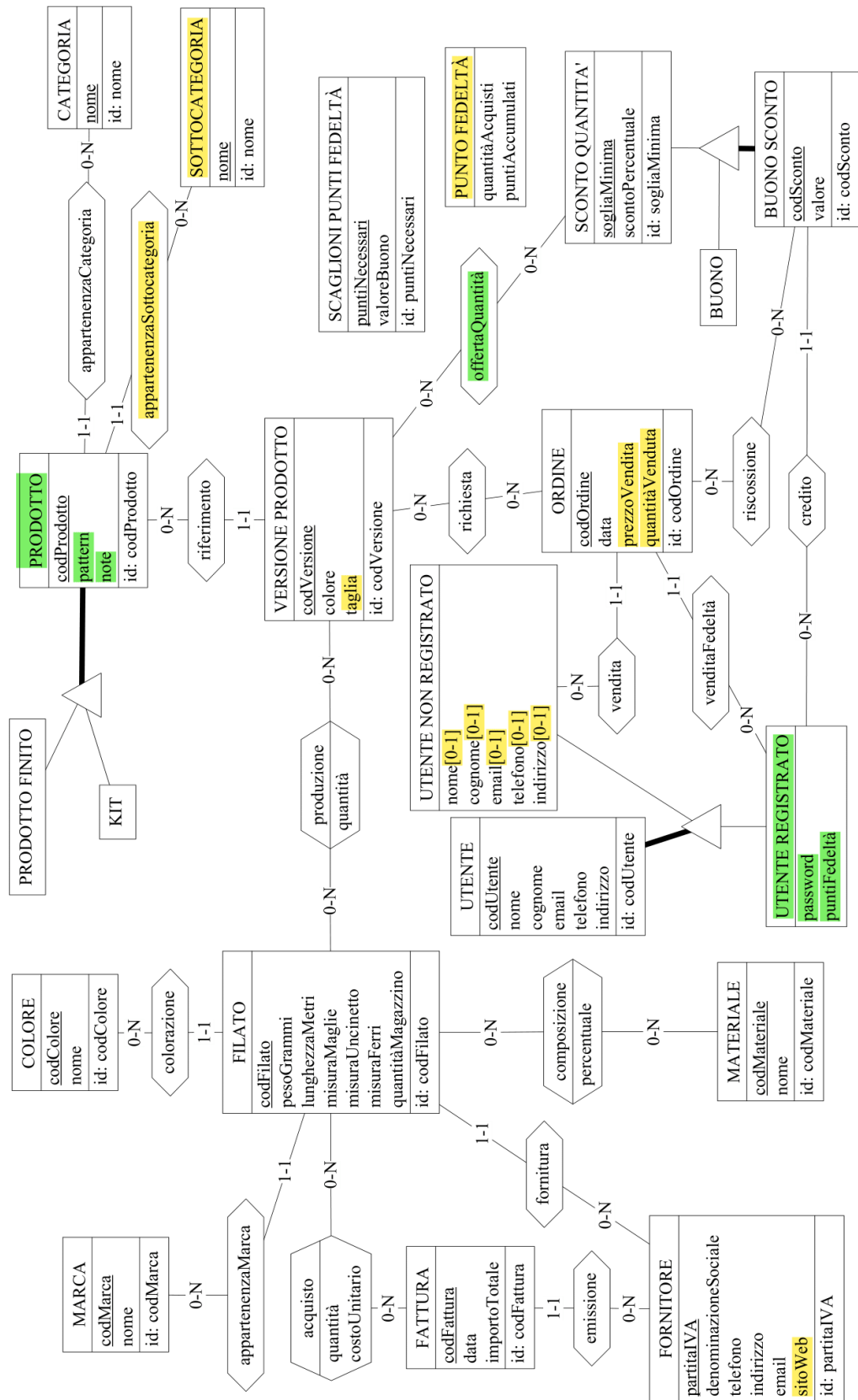


Figura 2.10: ERD completo finale seconda conversazione GPT 3.5

### 2.2.1.2 Valutazione comprensione linguaggio tecnico: capacità di esecuzione diretta

Si prova ad affrontare il prompting usando un approccio molto più tecnico e diretto, per valutare le capacità di comprensione ed esecuzione in ambito di modellazione ERD.

Il testo iniziale sarà una descrizione esplicita del diagramma da parte di un esperto che ha già analizzato e rielaborato l'intervista.

Tutti gli aggiustamenti e le correzioni verranno richiesti usando il più possibile termini tecnici, i quali dovrebbero indirizzare maggiormente GPT verso il risultato desiderato.

#### Rilevamento ambiguità e ridefinizione

Con questa conversazione si vuole studiare la capacità di comprensione dell'intervista originale tramite step guidati.

Il primo approccio è stato richiedere di effettuare un'analisi delle parole incontrate, creando un glossario dei termini. Quest'ultimo ha una semplice struttura di cinque colonne come segue:

- Termine
- Descrizione: una breve descrizione del concetto
- Sinonimi: eventuali sinonimi trovati
- Omonimi: casi in cui è possibile che lo stesso termine venga usato per descrivere concetti differenti
- Collegamenti: relazioni con altri concetti del glossario stesso

Nonostante si sia provato più volte a rigenerare l'output e specificare di estrarre i sinonimi dal testo, fornendo anche alcuni esempi, il modello continua ad aggiungere dati nuovi. Per questo motivo si è scelto di cambiare approccio in partenza, ricominciando chiedendo un semplice raggruppamento delle frasi per concetto.

Dato che per eliminare le ambiguità non è stato possibile indirizzare il modello tramite i termini, una volta suddivise le frasi si richiede direttamente di utilizzare una strategia bottom-up per creare il diagramma. Si spiega molto brevemente anche in cosa consiste questa strategia prima di effettuarla richiesta.

Una strategia bottom-up suddivide in concetti il testo in modo da sviluppare semplici schemi parziali ma dettagliati, che poi vengono integrati tra loro. Dimostra gli step che svolgi usando i concetti individuati per creare un ERD con la strategia bottom-up.

La risposta ottenuta, sebbene non sia basata sui concetti in base ai quali sono state raggruppate le frasi in precedenza, è apparentemente più tecnica e completa rispetto al prompting basato su un linguaggio più inesperto.

Ecco come suddividerei il testo in concetti e come svilupperei semplici schemi parziali per ciascun concetto:

**Concetti individuati:**

1. Filati
2. Fornitori
3. Prodotti
4. Modelli
5. Versioni Prodotto
6. Programma Fedeltà
7. Buoni Sconto
8. Utenti
9. Ordini

1. **Filati:**

- Dettagli filato:
  - ID\_Filato (PK)
  - Peso (grammi)
  - Lunghezza (metri)
  - Misura maglie
  - Misura uncinetto

- Misura ferri
- Quantità disponibile
- Dettagli composizione:
  - ID\_Materiale (PK)
  - Nome
- Marca:
  - ID\_Marca (PK)
  - Nome
- Colore:
  - ID\_Colore (PK)
  - Nome
- Composizione filato:
  - ID\_Composizione (PK)
  - ID\_Filato (FK)
  - ID\_Materiale (FK)
  - Percentuale

## 2. Fornitori:

- Fornitore:
  - Partita IVA (PK)
  - Denominazione sociale
  - Telefono
  - Indirizzo
  - Email
  - Sito web
- Fatture:
  - ID\_Fattura (PK)
  - Numero
  - Data
  - Importo
  - Partita IVA Fornitore (FK)

## 3. Prodotti:

- Categorie:

- ID\_Categoria (PK)
- Nome
- Sottocategorie:
  - ID\_Sottocategoria (PK)
  - ID\_Categoria (FK)
  - Nome
- Prodotto:
  - ID\_Prodotto (PK)
  - ID\_Modello (FK)
  - ID\_Versione (FK)
- Versione Prodotto:
  - ID\_Versione (PK)
  - ID\_Filato (FK)
  - ID\_Colore (FK)
  - Taglia
- Kit:
  - ID\_Kit (PK)
  - ID\_Modello (FK)
  - Quantità

#### 4. Modelli:

- Modello:
  - ID\_Modello (PK)
  - Codice
  - Pattern
  - Note

#### 5. Versioni Prodotto:

- Versione Prodotto:
  - ID\_Versione (PK)
  - ID\_Filato (FK)
  - ID\_Colore (FK)
  - Taglia

#### 6. Programma Fedeltà:



- Utente:
  - Email (PK)
  - Password
  - Punti Fedeltà
- RegISTRAZIONI Acquisti:
  - ID\_Acquisto (PK)
  - Email Utente (FK)
  - Data Acquisto
  - Importo
- Punti Accumulati:
  - ID\_Punto (PK)
  - Email Utente (FK)
  - Data Accumulo
  - Punti

#### 7. Buoni Sconto:

- Buono Sconto:
  - Codice (PK)
  - Email Utente (FK)
  - Importo

#### 8. Utenti:

- Utente:
  - Email (PK)
  - Password

#### 9. Ordini:

- Ordine:
  - ID\_Ordine (PK)
  - Email Utente (FK)
  - Data Ordine
- Dettagli Ordine:
  - ID\_Dettaglio (PK)
  - ID\_Ordine (FK)
  - ID\_Prodotto (FK)

– Quantità

Ora possiamo integrare questi schemi parziali in un unico schema ERD:

```
/* ***** ENTITIES ***** */
/* Filati */
FILATI {
    ID_Filato (PK),
    Peso,
    Lunghezza,
    Misura_maglie,
    Misura_uncinetto,
    Misura_ferri,
    Quantita'_disponibile,
    ID_Marca (FK),
    ID_Colore (FK)
}

MATERIALE {
    ID_Materiale (PK),
    Nome
}

COMPOSIZIONE_FILATO {
    ID_Composizione (PK),
    ID_Filato (FK),
    ID_Materiale (FK),
    Percentuale
}

MARCHE {
    ID_Marca (PK),
    Nome
}

COLORI {
    ID_Colore (PK),
    Nome
}

/* Fornitori */
FORNITORI {
    Partita_IVA (PK),
    Denominazione_sociale,
    Telefono,
    Indirizzo,
    Email,
    Sito_web
}
```

```
}  
  
FATTURE {  
    ID_Fattura (PK),  
    Numero,  
    Data,  
    Importo,  
    Partita_IVA_Fornitore (FK)  
}  
  
/* Prodotti */  
CATEGORIE {  
    ID_Categoria (PK),  
    Nome  
}  
  
SOTTOCATEGORIE {  
    ID_Sottocategoria (PK),  
    ID_Categoria (FK),  
    Nome  
}  
  
MODELLI {  
    ID_Modello (PK),  
    Codice,  
    Pattern,  
    Note  
}  
  
PRODOTTI {  
    ID_Prodotto (PK),  
    ID_Modello (FK),  
    ID_Versione (FK)  
}  
  
VERSIONI_PRODOTTI {  
    ID_Versione (PK),  
    ID_Filato (FK),  
    ID_Colore (FK),  
    Taglia  
}  
  
KIT {  
    ID_Kit (PK),  
    ID_Modello (FK),  
    Quantita'  
}
```

```
/* Programma Fedelta' */
UTENTI {
    Email (PK),
    Password,
    Punti_Fedelta'
}

REGISTRAZIONI_ACQUISTI {
    ID_Acquisto (PK),
    Email_Utente (FK),
    Data_Acquisto,
    Importo
}

PUNTI_ACCUMULATI {
    ID_Punto (PK),
    Email_Utente (FK),
    Data_Accumulo,
    Punti
}

/* Buoni Sconto */
BUONI_SCONTO {
    Codice (PK),
    Email_Utente (FK),
    Importo
}

/* Utenti */
UTENTI {
    Email (PK),
    Password
}

/* Ordini */
ORDINI {
    ID_Ordine (PK),
    Email_Utente (FK),
    Data_Ordine
}

DETTAGLI_ORDINE {
    ID_Dettaglio (PK),
    ID_Ordine (FK),
    ID_Prodotto (FK),
    Quantita'
}
```

```

    }

/***** RELATIONSHIPS *****/
FILATI ||--|| MARCHE : ID_Marca
FILATI ||--|| COLORI : ID_Colore
FILATI ||--|| COMPOSIZIONE_FILATO : ID_Filato
COMPOSIZIONE_FILATO ||--|{ MATERIALE : ID_Materiale
FORNITORI ||--|| FATTURE : Partita_IVA
PRODOTTI ||--|{ VERSIONI_PRODOTTI : ID_Versione
PRODOTTI ||--|{ KIT : ID_Kit
PRODOTTI ||--|| MODELLI : ID_Modello
MODELLI ||--|| SOTTOCATEGORIE : ID_Categoria
CATEGORIE ||--|| SOTTOCATEGORIE : ID_Categoria
REGISTRAZIONI_ACQUISTI ||--|| UTENTI : Email
PUNTI_ACCUMULATI ||--|| UTENTI : Email
BUONI_SCONTO ||--|| UTENTI : Email
ORDINI ||--|| UTENTI : Email
DETTAGLI_ORDINE ||--|{ ORDINI : ID_Ordine
DETTAGLI_ORDINE ||--|{ PRODOTTI : ID_Prodotto

```

Questo schema rappresenta l'intero sistema informativo per l'azienda CraftyCrocheters, organizzando le entità e le relazioni in modo chiaro e coerente.

Si chiede di spiegare la cardinalità rappresentata dalla notazione usata e provando a rigenerare la risposta due volte si ottengono due spiegazioni diverse. Lo stesso simbolo `| |-|` se prima si riferisce a una relazione molti a molti (o n a n) tra due entità, nella seconda risposta è relativa ad una cardinalità "uno a uno".

Per questo si chiede una spiegazione a parole di ogni relazione individuata, in modo da poter disegnare lo schema proposto da ChatGPT. Anche in questo caso la risposta risulta confusa; ad esempio l'associazione tra filati e marche viene così definita:

1. FILATI `| |-|` MARCHE : ID\_Marca

- Relazione: Ogni filato è associato a una marca.
- Cardinalità: Uno a uno (1:1). Ogni filato è prodotto da una singola marca, e una marca può produrre molti tipi di filati.

È evidente che la cardinalità non è coerente con la spiegazione concettuale. Inoltre, spesso la spiegazione stessa non risulta veramente esplicativa, ad esempio quella definita tra prodotti e kit:

### 7. PRODOTTI | |-{ KIT : ID\_Kit

- Relazione: Un prodotto può essere venduto sia come prodotto finito sia come kit.
- Cardinalità: Uno a molti (1:N). Un prodotto può essere venduto come prodotto finito o come parte di uno o più kit, ma ogni kit è composto da un unico prodotto.

”Un prodotto può essere venduto come prodotto finito o come parte di uno o più kit” non è esattamente atomica come spiegazione, lascia molto spazio a interpretazione e non è chiara.

Inoltre, molti concetti non sono ben definiti, per citarne solo alcuni:

- Nuovamente non vengono accennate le gerarchie: esistono due entità utente quasi identiche con lo stesso nome e c'è molta confusione su prodotti, prodotti finiti, kit e modelli
- Non viene affrontata la questione dello sconto quantità
- L'associazione per la composizione del filato è quasi appropriata, ma altrettanto confusa nelle cardinalità
- I filati non vengono acquistati dai fornitori e nemmeno usati per la produzione
- Registrazione acquisti non si capisce bene che ruolo ha all'interno del database, essendo già presenti ordine e dettaglio ordine

Avendo quindi così tante incognite, nonostante i numerosi tentativi di rigenerazione e riformulazione della proposta, si sceglie di iniziare una conversazione differente.

### **Un approccio più esperto e telegrafico**

L'intenzione in questo caso è usare la tecnica bottom-up, anziché lasciando in autonomia tutta la gestione del processo, procedendo in maniera guidata. Suddividiamo il diagramma in concetti:

- Filato: caratteristiche e fornitura

- Prodotti in vendita
- Utenti e vendite
- Buoni sconto

Partendo dal primo sottoinsieme si fornisce un elenco più direttamente dettagliato:

Sei un database architect che deve progettare il diagramma ER per uno small business di uncinetto.

- Un filato ha come attributi: `codFilato(pk)`, `pesoGrammi`, `lunghezzaMetri`, `misuraMaglie`, `misuraUncinetto`, `misuraFerri`, `quantitàMagazzino`
- Ha un colore e una marca dotati rispettivamente di `codColore(pk)`, nome e descrizione e `codMarca(pk)`, nome e note
- Ogni filato viene acquistato da diversi fornitori con `partitaIVA(pk)`, denominazione sociale, dettagli di contatto (telefono, indirizzo,email) e sito web opzionale a prezzi diversi
- Per ogni acquisto esiste una fattura con `codFattura(pk)` e data
- Un filato è composto in una certa percentuale da diversi materiali (esempio 30% lana, 70% alpaca)

Nella risposta, dopo aver elencato le entità con i relativi attributi, indicando chiavi primarie e secondarie, prosegue con una breve spiegazione delle associazioni:

RELATIONSHIPS:

- Filato (1) <—> (N) ComposizioneFilato
- Colore (1) <—> (N) Filato
- Marca (1) <—> (N) Filato
- Fornitore (1) <—> (N) Acquisto
- Acquisto (1) <—> (1) Fattura

Spiegazione:

- Un filato può avere più colori e marche associati.
- Ogni filato può essere composto da diversi materiali con la relativa percentuale.

- Ogni filato può essere acquistato da uno o più fornitori a prezzi diversi, con ogni acquisto associato a una fattura.
- Un fornitore può essere associato a molti acquisti.
- Ogni acquisto ha una sola fattura associata.

Purtroppo anche riducendo gli elementi da prendere in considerazione, risulta questa incongruenza tra le chiavi primarie e secondarie indicate tra gli attributi delle entità e le descrizioni delle cardinalità relazioni, nonostante sia tutto parte dello stesso output elaborato.

Decidendo di basarsi solo sulle chiavi esterne indicate, dopo un ulteriore prompt di autoanalisi per correggere la mancanza dell'entità 'materiale' nonostante avesse considerato 'composizione materiale' che comprende codMateriale come chiave esterna, lo schema risulta il seguente (figura 2.11):

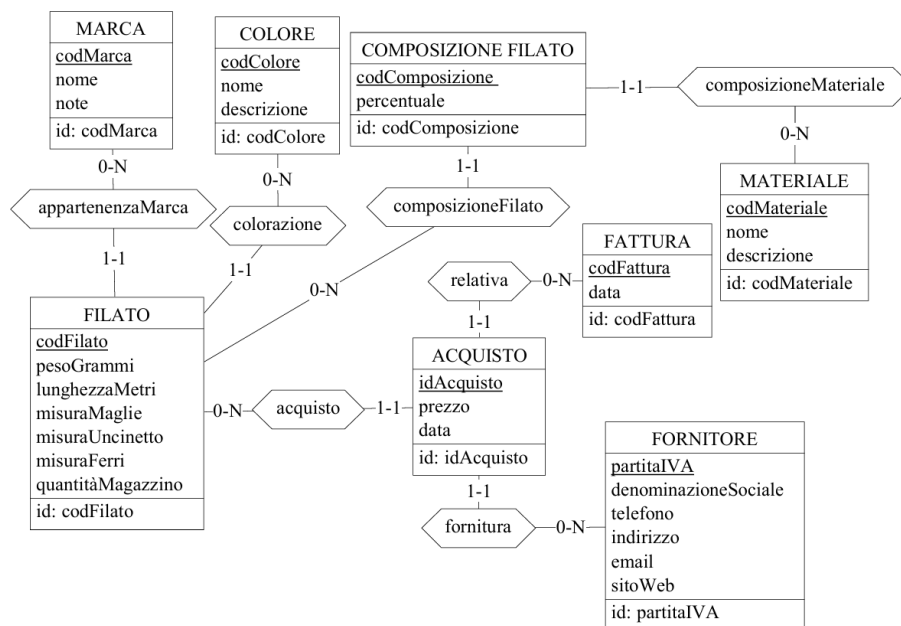


Figura 2.11: ERD bottom-up: filato

Dopodiché si procede col concetto di prodotto: nel modellare categorie e sottocategorie GPT crea nuovamente due entità separate. Senza fare riferimento alle



associazioni ricorsive, viene chiesto se una categoria può fare riferimento a un'altra categoria per creare sotto categorie. La risposta ottenuta è positiva: viene introdotta l'associazione ad anello e quando si chiede 'come semplifichereesti lo schema senza perdere informazioni?' rimuove l'entità sottocategoria che risulterebbe superflua (figura 2.12).

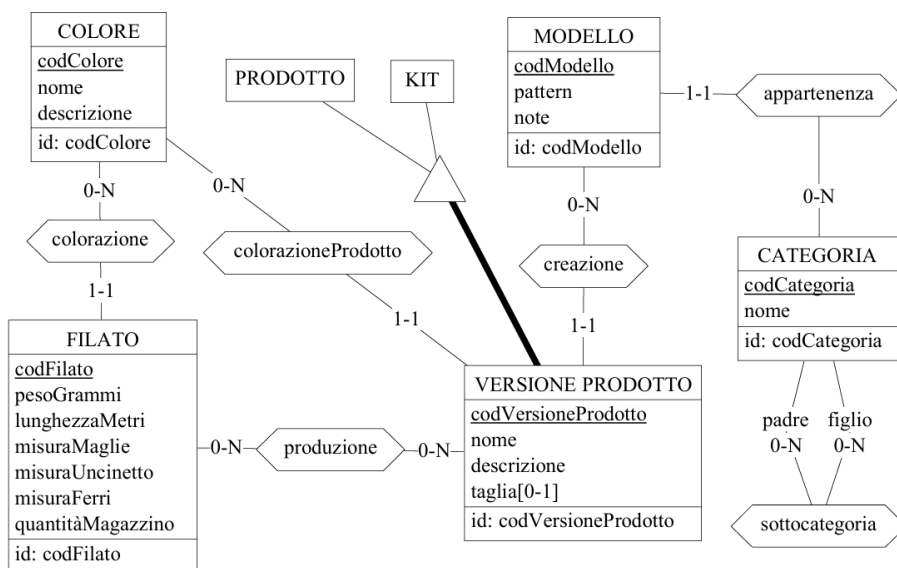


Figura 2.12: ERD bottom-up: prodotto

Per evitare perdite di dati si chiede di unire questa sezione con quella dei filati e si ottiene con successo la fusione dei 2 diagrammi.

Successivamente si definisce il dettaglio del diagramma sulla tipologia di utente (figura 2.13).

Si cerca di capire se GPT 3.5 riesce a rappresentare efficacemente le gerarchie, chiedendo se è possibile applicarla sull'entità utente: afferma di aver utilizzato una gerarchia con l'entità utente come entità madre e utente registrato come entità figlia. Si potrebbe commentare che il modo di rappresentare questo tipo di struttura non viene reso molto esplicito dal modello, al contrario va interpretato leggendo l'ERD che prova a spiegare a parole.

Inoltre, inizialmente aggiunge un'associazione 'dettaglio ordine - utente' che non è molto pertinente al testo: GPT nell'introdurre questa relazione spiega che

solo gli utenti registrati possono avere i dettagli del proprio ordine. Questo genere di assunzione è un po' troppo astratta e non esattamente pertinente, viene chiesto quindi di spiegare tutte le associazioni che concernono utenti, ordine e dettagli ordine. Risulta che viene effettuata un'auto correzione nel fornire spiegazioni e 'utente - dettaglio' ordine sparisce dal diagramma.

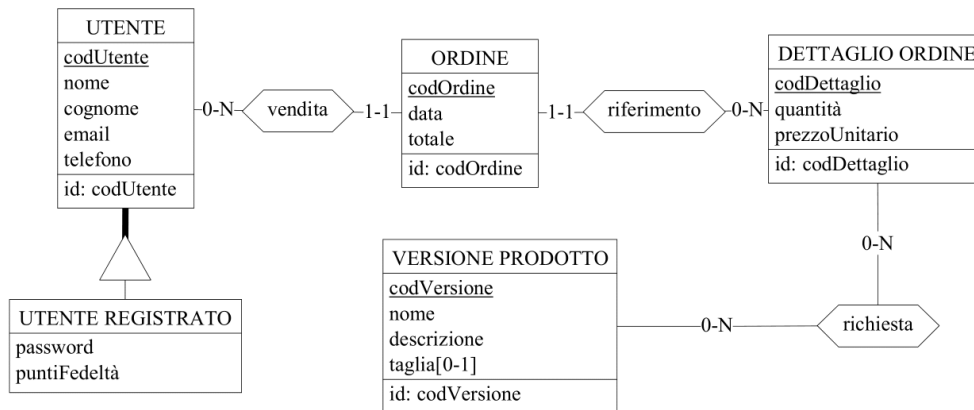


Figura 2.13: ERD bottom-up: vendite e utente

Per concludere si passa alla definizione degli sconti (figura 2.14).

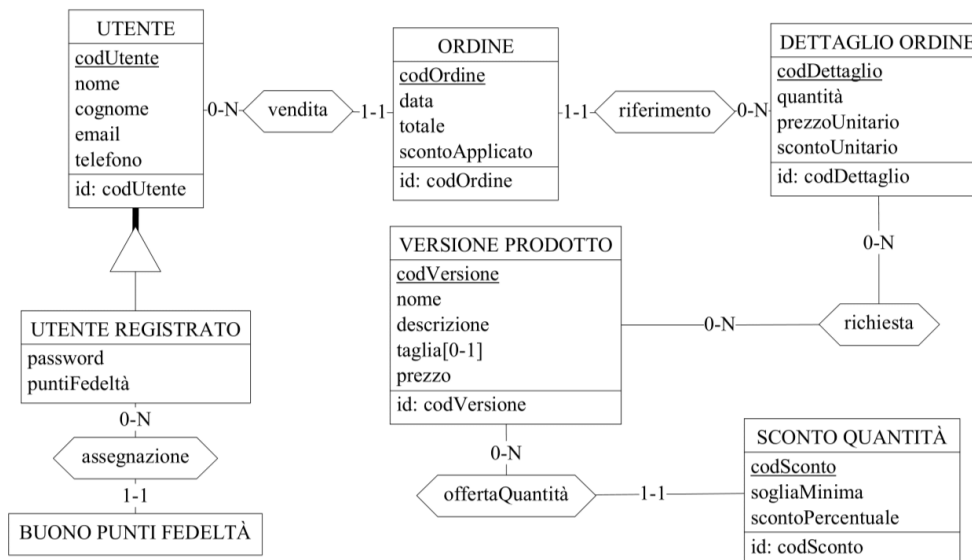


Figura 2.14: ERD bottom-up: sconti

Senza applicare particolari ridefinizioni sugli sconti si sceglie di procedere con la fusione totale degli schemi, che non avviene più così fedelmente; diverse associazioni ed entità sono state perse, per questo è stato necessario fare un controllo manuale degli elementi mancanti e chiedere al modello direttamente di reintegrarli.

Un dettaglio ordine dovrebbe essere relativo a ogni prodotto dell'ordine a cui si riferisce, quindi la cardinalità dovrebbe essere 1-N e non N-N. Si prova a indicare questa condizione, ma anziché correggersi sbaglia diversamente cambiando la cardinalità in 1-1.

Si prova allora a fare una semplice richiesta di tracciare la quantità di filato usata per la produzione di una versione di prodotto: il risultato è la sostituzione dell'associazione produzione tra filato e versione prodotto (figura 2.12) con un'entità consumo filato, avente per attributo la quantità usata. Il problema risiede nella cardinalità assegnata N-N per entrambe le entità collegate. Si può notare come questo non sia l'unico caso: in casi come acquisto dei filati e dettaglio ordine nelle vendite ha scelto di introdurre entità per rappresentare i dati specifici di certe relazioni N-N. Dall'altra parte composizione filato è un'associazione N-N correttamente impostata come entità.

Riguardo alla fornitura di filato, sarebbe stato più efficace collegare il fornitore alle fatture emesse anziché al dettaglio di acquisti presenti in una fattura.

Lo schema ottenuto (figura 2.15) risulta quindi anche in questo caso non solo differente, ma parzialmente incorretto rispetto all'originale progettato (figura 2.1).

Al termine di queste sessioni di prompting, si potrebbe affermare che GPT possiede le conoscenze potenziali a risolvere il problema di creazione di ERD, ma ha ancora molto progresso da fare per ottenere una maggiore comprensione del linguaggio naturale e un migliore capacità di rappresentazione dei concetti.

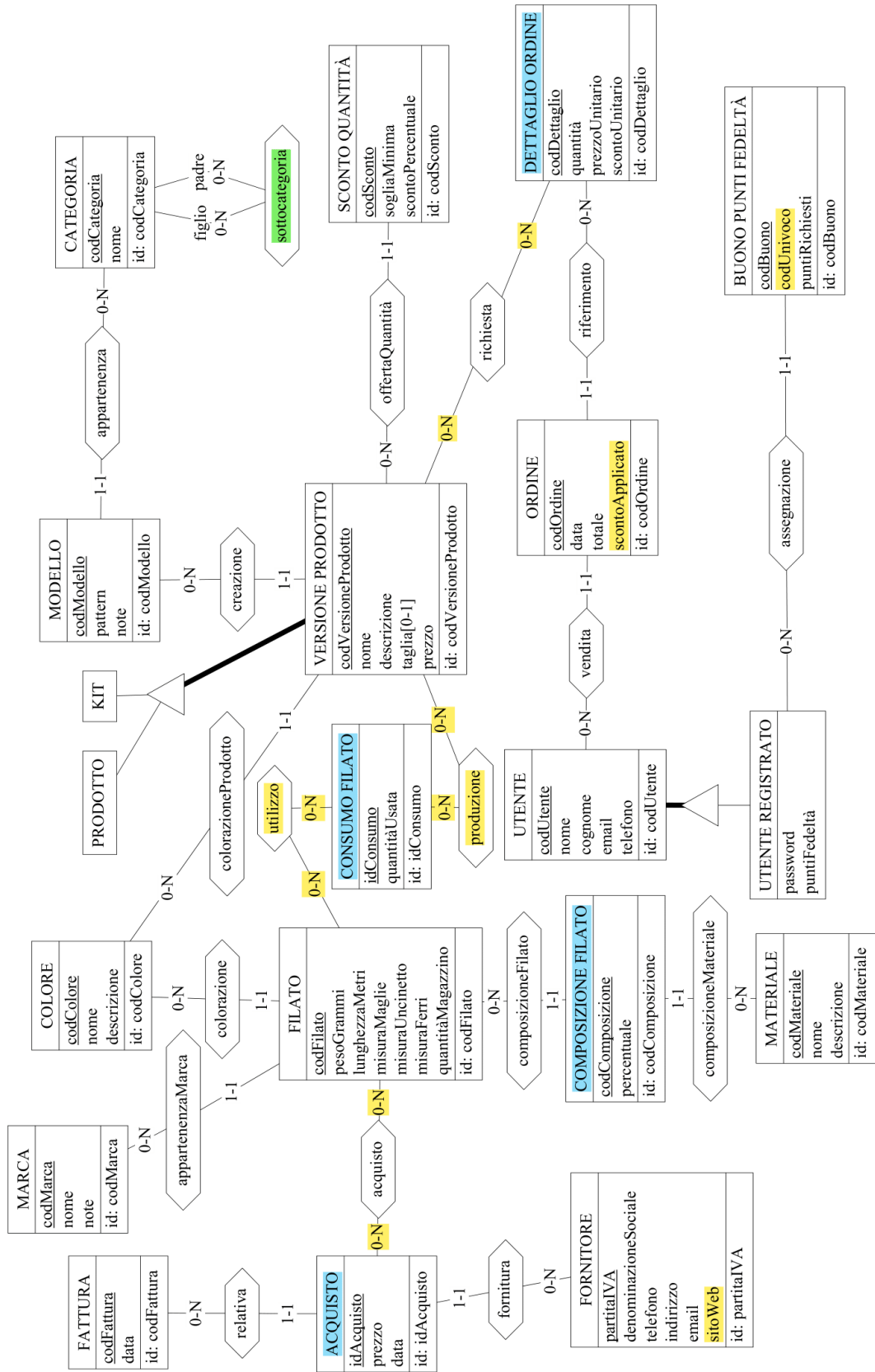


Figura 2.15: ERD finale approccio bottom-up GPT 3.5

## 2.3 Progettazione logica

### 2.3.1 Schema relazionale

Fornendo a ChatGPT un elenco dettagliato che descrive l'ERD progettato inizialmente (figura 2.1), si chiede di effettuare la progettazione logica del diagramma.

La risposta viene scritta direttamente sotto forma di query sql per la creazione delle tabelle. Risulta davvero molto coerente all'input; per gestire le associazioni N-N e 1-N, sono state aggiunte entità. Per quelle 1-N però la soluzione preferibile prevede l'importazione di foreign key, per questo si chiede una spiegazione a riguardo. Il modello si auto corregge e l'output risultante ora presenta solo pochi errori:

1. Non è rappresentata l'associazione che permette di avere sottocategorie
2. Sono state introdotte entità prodotto e kit che potevano invece essere accorpate come attributo di tipo dell'entità versione prodotto
3. Manca l'associazione tra versione prodotto e sconto quantità
4. L'entità riscossione introdotta per rappresentare le relazione 0-N, come anche l'entità credito tra utente registrato e buono sconto non sono necessarie

Per sistemare le sottocategorie decide di aggiungere una tabella che le rappresenta, mentre per quanto riguarda kit e prodotto questa è la soluzione che propone:

Per modellare il fatto che "Kit" e "Prodotto" sono due tipi di "VersioneProdotto", possiamo utilizzare una strategia di ereditarietà chiamata "Tutte le tabelle con sottoinsiemi concreti". In questa strategia, ogni sottoinsieme concreto (nel nostro caso, "Kit" e "Prodotto") ha una propria tabella e condivide la chiave primaria con la tabella genitore ("VersioneProdotto").

Ecco come possiamo modificare lo schema per riflettere questa relazione di ereditarietà:

```
CREATE TABLE VersioneProdotto (  
  codVersione INT PRIMARY KEY,  
  nome VARCHAR(255),  
  colore INT,  
  taglia VARCHAR(50),  
  prezzo DECIMAL(10, 2),
```

```
    descrizione TEXT,
    tipo ENUM('Kit', 'Prodotto')
);

CREATE TABLE Kit (
    codVersione INT PRIMARY KEY,
    -- Altri attributi specifici per i kit
    FOREIGN KEY (codVersione) REFERENCES VersioneProdotto(codVersione)
);

CREATE TABLE Prodotto (
    codVersione INT PRIMARY KEY,
    -- Altri attributi specifici per i prodotti
    FOREIGN KEY (codVersione) REFERENCES VersioneProdotto(codVersione)
);
```

In questo modo, ogni record nella tabella 'Kit' e 'Prodotto' condivide la stessa chiave primaria ('codVersione') con la tabella 'VersioneProdotto'. La colonna 'tipo' nella tabella 'VersioneProdotto' identifica se una particolare versione prodotto è un kit o un prodotto.

Inoltre, chiedendo se nonostante non ci sono altri attributi specifici per i prodotti e i kit, bisognerebbe mantenere le entità, lo schema viene ottimizzato eliminando questi elementi superflui. Viene però persa la relazione versione prodotto-modello, in quanto non viene riportata più la chiave esterna: per questo è necessario applicare una correzione diretta per ripristinare l'associazione.

Si chiede perché è stata usata l'entità riscossione per rappresentare l'associazione tra ordine e buono sconto, e nuovamente il modello si autocorregge: viene aggiunta una chiave esterna nell'ordine. Lo stesso processo si verifica nel caso dell'entità credito.

Inoltre, risulta che il saldo dei punti fedeltà è tenuto anche all'interno dell'entità 'ordine', perciò si chiede di rimuovere l'attributo ridondante e incorretto.

In questo modo si ottiene quindi, partendo da una descrizione molto chiara e ben definita del diagramma ER, esattamente lo schema desiderato.

Unica aggiunta rispetto al risultato con GPT, è l'attributo 'scontoQuantitàApplicato' all'interno di 'richiesta' (entità tra versione prodotto e ordine): si è reputato che una volta assegnato lo sconto quantità migliore sia più efficace tenerne traccia dato che nel tempo potrebbero esserne aggiunti altri cambiando il calcolo dello sconto.

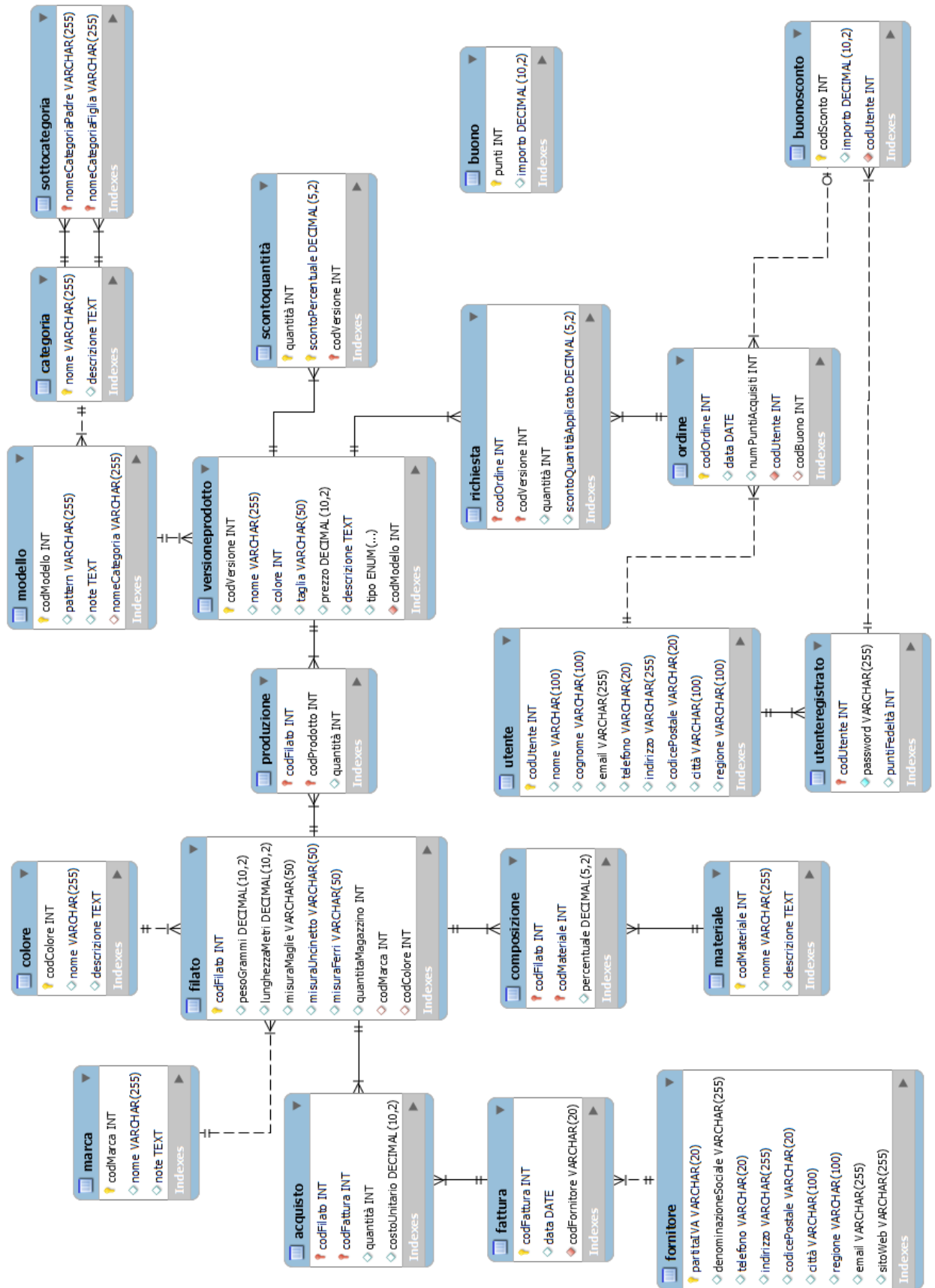


Figura 2.16: Schema relazionale finale

# Capitolo 3

## Interrogazione di database in linguaggio naturale

### 3.1 Elaborazione query SQL

#### 3.1.1 Schema relazionale completo: analisi cumulativa

Partendo dalle query SQL che generano lo schema relazionale ottenuto al termine del capitolo precedente (figura 2.16), si chiede a GPT di generare l'SQL per tutte le operazioni. Di seguito l'analisi della prima risposta suddivisa per query.

1.1 Visualizzare i filati composti da una certa percentuale di un determinato materiale

```
SELECT f.codFilato, f.pesoGrammi, f.lunghezzaMetri
FROM filato f
JOIN composizione c ON f.codFilato = c.codFilato
JOIN materiale m ON c.codMateriale = m.codMateriale
WHERE m.nome = 'NomeMateriale' AND c.percentuale >= 0.5;
```

Sostituisci 'NomeMateriale' con il nome del materiale desiderato e 0.5 con la percentuale desiderata.

Con questa prima query si può affermare che GPT è in grado di effettuare operazioni SQL semplici: sostituendo i valori si ottiene il risultato desiderato.



1.2 Visualizzare il fatturato vendite di questo mese (considerando sconti quantità e buoni sconto applicati)

```
SELECT SUM((r.quantita * vp.prezzo) * (1 - SQ.scontoPercentuale) * (1
  - B.importo / 100)) AS Fatturato
FROM richiesta r
JOIN ordine o ON r.codOrdine = o.codOrdine
JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
LEFT JOIN scontoquantita SQ ON r.quantita >= SQ.quantita AND SQ.
  codVersione = vp.codVersione
LEFT JOIN ordine B ON o.codBuono = B.codBuono
WHERE MONTH(o.data) = MONTH(CURDATE()) AND YEAR(o.data) = YEAR(CURDATE
  ());
```

In questo caso vediamo le prime difficoltà: ChatGPT riesce a comprendere la logica, ma anziché considerare lo sconto applicato per ogni prodotto in un ordine (all'interno dell'entità 'richiesta') risale alla tabella sconto quantità. Inoltre vuole fare riferimento all'importo con B.importo nella SELECT, quando la tabella ordine non presenta quest'attributo e chiamare ordine B rende la lettura ambigua.

1.3 Visualizzare le categorie padre di una determinata sottocategoria, ordinando in base al livello di gerarchia

```
SELECT c1.nome AS CategoriaPadre
FROM sottocategoria sc
JOIN categoria c1 ON sc.nomeCategoriaPadre = c1.nome
WHERE sc.nomeCategoriaFiglia = 'NomeSottocategoria'
ORDER BY c1.nome;
```

Sostituisci 'NomeSottocategoria' con il nome della sottocategoria desiderata.

Qui si riesce a ottenere la categoria immediatamente superiore a quella data, ma non si risale ricorsivamente per poter effettivamente ottenere un elenco di tutta la gerarchia superiore.

1.4 Visualizzare le vendite per ciascun modello con fatturato minore di 100 dell'ultimo mese, ordinando dal fatturato maggiore al minore

```
SELECT vp.codModello, COUNT(*) AS NumeroVendite, SUM(r.quantita * vp.
  prezzo * (1 - SQ.scontoPercentuale) * (1 - B.importo / 100)) AS
  Fatturato
```

```

FROM richiesta r
JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
JOIN ordine o ON r.codOrdine = o.codOrdine
LEFT JOIN scontoquantita SQ ON r.quantita >= SQ.quantita AND SQ.
    codVersione = vp.codVersione
LEFT JOIN ordine B ON o.codBuono = B.codBuono
WHERE o.data >= DATE_SUB(LAST_DAY(CURDATE()), INTERVAL 1 MONTH) +
    INTERVAL 1 DAY
GROUP BY vp.codModello
HAVING Fatturato < 100
ORDER BY Fatturato DESC;

```

Risulta mantenere in maniera coerente la logica della query 1.2 per quanto riguarda le vendite, proseguendo quindi con gli stessi errori.

#### 1.5 Visualizzare le versioni dei prodotti che non sono stati ordinati negli ultimi 6 mesi

```

SELECT vp.*
FROM versioneprodotto vp
LEFT JOIN richiesta r ON vp.codVersione = r.codVersione
WHERE r.codOrdine IS NULL OR r.data < DATE_SUB(CURDATE(), INTERVAL 6
    MONTH);

```

In questo caso cerca di recuperare il valore della data da richiesta, senza considerare che in realtà la tabella non presenta questo campo.

#### 1.6 Visualizzare gli utenti che hanno ordinato prodotti di tutti i modelli

```

SELECT u.codUtente, u.nome, u.cognome
FROM utente u
JOIN ordine o ON u.codUtente = o.codUtente
JOIN richiesta r ON o.codOrdine = r.codOrdine
JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
GROUP BY u.codUtente
HAVING COUNT(DISTINCT vp.codModello) = (SELECT COUNT(*) FROM modello);

```

#### 1.7 Visualizzare tutte le categorie, riportando anche il numero di prodotti che eventualmente vi appartengono

```

SELECT c.nome AS Categoria, COUNT(m.codFilato) AS NumeroProdotti
FROM categoria c
LEFT JOIN modello mo ON c.nome = mo.nomeCategoria

```

```

LEFT JOIN versioneprodotto vp ON mo.codModello = vp.codModello
LEFT JOIN richiesta r ON vp.codVersione = r.codVersione
LEFT JOIN filato m ON r.codVersione = m.codFilato
GROUP BY c.nome;

```

### 1.8 L'utente con il maggior numero di ordini per ogni anno

```

SELECT YEAR(o.data) AS Anno, u.codUtente, u.nome, u.cognome, COUNT(*)
      AS NumeroOrdini
FROM ordine o
JOIN utente u ON o.codUtente = u.codUtente
GROUP BY YEAR(o.data), u.codUtente
HAVING COUNT(*) = (
      SELECT MAX(NumeroOrdini)
      FROM (
            SELECT YEAR(data) AS Anno, codUtente, COUNT(*) AS NumeroOrdini
            FROM ordine
            GROUP BY YEAR(data), codUtente
          ) AS T
      WHERE T.Anno = YEAR(o.data)
);

```

Mentre le query 1.6 e 1.7 risultano corrette, la 1.8 non è eseguibile in quanto nella clausola WHERE viene utilizzato un attributo non definito quale o.data. Sarebbe anche da modificare l'incapsulamento della subquery effettuando il raggruppamento per anno all'interno ed estrapolare così il risultato.

### 2.1 Registrare la vendita a un utente non registrato (non c'è accumulo di punti) con controllo sconto quantità migliore

```

INSERT INTO ordine (data, codUtente, codBuono)
VALUES (CURDATE(), 1, (
      SELECT codSconto
      FROM buonosconto
      WHERE importo = (
            SELECT MAX(importo)
            FROM buonosconto
          )
      LIMIT 1
));

```

In questo caso si può notare una certa confusione nell'inserimento di una vendita all'interno del database:

- Si considera `codOrdine` come un valore automaticamente incrementale, non essendo specificato è un'assunzione presa in autonomia dal modello
- Viene applicato un buono sconto anziché controllare se c'è uno sconto quantità possibile da riscuotere

2.2 Registrare la vendita a un utente registrato (ogni 10 euro accumula un punto fedeltà) con controllo sconto quantità migliore

```
INSERT INTO ordine (data, codUtente, codBuono, numPuntiAcquisiti)
VALUES (CURDATE(), 'CodiceUtenteRegistrato', (
    SELECT codSconto
    FROM buonosconto
    WHERE importo = (
        SELECT MAX(importo)
        FROM buonosconto
    )
    LIMIT 1
),
(
    SELECT FLOOR(SUM(vp.prezzo * r.quantita) / 10)
    FROM richiesta r
    JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
    WHERE r.codOrdine = 'CodiceOrdineAppenaCreato'
));
```

Sostituisci `'CodiceUtenteRegistrato'` con il codice dell'utente registrato e `'CodiceOrdineAppenaCreato'` con il codice dell'ordine appena creato.

Nuovamente alla registrazione di una vendita rimane coerente alla sua logica precedente e applica buoni sconto anziché considerare gli sconti quantità.

Provando a chiedere un raffinamento cumulativo delle query incorrette, indicando direttamente le incongruenze appena commentate, l'unica che risulta funzionante è la 1.5.

- Nella 1.2 si ottiene un valore di fatturato negativo a causa del modo incorretto di considerare gli sconti nel calcolo e del non utilizzo dell'attributo `'scontoQuantitàApplicato'` in `'richiesta'`.
- La 1.3 presenta una sintassi inesistente al termine della query quale `'ORDER BY LEVEL() DESC;'` per tentare di eseguire la ricorsività dell'operazione.

- La 1.4 non calcola bene lo sconto come anche la 1.2
- Nella 1.8 si ottiene lo stesso errore precedentemente riscontrato
- Sebbene la 2.1 adesso dia la possibilità di registrare il codice dell'ordine (nonostante sia posto come NULL nei valori di esempio che propone) e anche di registrare gli acquisti singoli in 'richiesta', continua a non considerare gli sconti quantità
- Anche nel caso della 2.2 è ora possibile inserire valori anche in richiesta con una seconda query, ma propone una terza per aggiornare il valore dei punti usando una variabile 'importo' non esistente e manca anche qui la gestione degli sconti quantità

Provando ad effettuare nuovamente la correzione delle query usando lo stesso formato, non si ottengono particolari miglioramenti, per questo si sceglie di cambiare approccio in una nuova conversazione.

### 3.1.2 Schema relazionale completo: analisi singola

Nel primo prompt anche in questo caso si riporta la totalità dello schema relazionale sottoforma di query SQL, ma si chiede l'esecuzione solo della prima query 1.1. La risposta ottenuta è decisamente più dettagliata e la scrittura stessa della query risulta più chiara del tentativo precedente, nonostante in entrambi i casi sia comunque funzionante.

Per visualizzare i filati composti da una certa percentuale di un determinato materiale, puoi utilizzare la tabella 'composizione' nel database 'gpt'. Ecco come puoi strutturare la query SQL:

```
SELECT f.*
FROM filato f
JOIN composizione c ON f.codFilato = c.codFilato
JOIN materiale m ON c.codMateriale = m.codMateriale
WHERE m.nome = 'nome_del_materiale'
      AND c.percentuale >= percentuale_minima;
```

Spieghiamo la query:

1. 'SELECT f.\*': Seleziona tutte le colonne dalla tabella 'filato' per i filati che soddisfano le condizioni specificate.
2. 'JOIN composizione c ON f.codFilato = c.codFilato': Esegue un join tra la tabella 'filato' e 'composizione' sulla chiave 'codFilato', per trovare le composizioni dei filati.
3. 'JOIN materiale m ON c.codMateriale = m.codMateriale': Esegue un secondo join con la tabella 'materiale' sulla chiave 'codMateriale', per associare il materiale alla composizione.
4. 'WHERE m.nome = 'nome\_del\_materiale''': Specifica il materiale desiderato cercando per il suo nome.
5. 'AND c.percentuale >= percentuale\_minima': Facoltativamente, puoi aggiungere questa condizione per filtrare i filati che hanno una percentuale di questo materiale uguale o superiore a un valore specifico.

Assicurati di sostituire 'nome\_del\_materiale' con il nome del materiale di interesse e 'percentuale\_minima' con il valore minimo di percentuale desiderato, se applicabile.

Questa query restituirà i filati che sono composti dal materiale specificato secondo i criteri definiti.

Come anticipato la query risulta ancora corretta, si prosegue con l'analisi della query 1.2. La risposta è molto dettagliata come nel caso precedente, ma per brevità da adesso in poi verrà riportato solo il processo svolto per ogni query.

```

SELECT SUM(vp.prezzo * r.quantita * (1 - IFNULL(sq.scontoPercentuale, 0)) * (1
    - IFNULL(b.importo, 0))) AS fatturato_vendite
FROM ordine o
JOIN richiesta r ON o.codOrdine = r.codOrdine
JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
LEFT JOIN scontoquantita sq ON r.quantita >= sq.quantita AND vp.codVersione =
    sq.codVersione
LEFT JOIN buonosconto b ON o.codBuono = b.codSconto
WHERE MONTH(o.data) = MONTH(CURDATE()) AND YEAR(o.data) = YEAR(CURDATE());

```

Inizialmente si può notare subito che lo sconto applicato nuovamente non è l'attributo presente in richiesta, ma viene preso dalla tabella 'sconto quantità'; si chiede quindi di riformulare ed effettivamente il modello si corregge. L'unico problema che permane è la sottrazione di sconti e buoni dall'importo finale; il calcolo risulta incorretto in quanto sconti e buoni vengono moltiplicati tra loro e

considerati per ogni richiesta. Nonostante si provi a spiegare l'errore GPT torna ciclicamente a ripetere la query sbagliata.

Si procede quindi con la 1.3:

```
SELECT c.nome AS categoria_padre
FROM sottocategoria sc
JOIN categoria c ON sc.nomeCategoriaPadre = c.nome
WHERE sc.nomeCategoriaFiglia = 'sottocategoria_specificata'
ORDER BY LENGTH(c.nome) ASC;
```

Nuovamente l'errore risiede nel mostrare solo la categoria immediatamente superiore a quella data, senza risalire la gerarchia completa. Si procede con una domanda di autoanalisi, ma non cambia nulla, si prova con una correzione diretta chiedendo di usare una query ricorsiva. Dopodiché sono serviti altri 2 prompt, con la descrizione diretta di errori relativi alla sintassi SQL, per riuscire a ottenere il risultato desiderato. A seguire la query 1.3 funzionante ottenuta:

```
-- Definisci la sottocategoria di partenza
SET @sottocategoria = 'sottocategoria_specificata';

-- Query per trovare le categorie padre
WITH RECURSIVE categorie_gerarchia AS (
    SELECT sc.nomeCategoriaPadre AS categoria_padre, sc.nomeCategoriaFiglia AS
        categoria_figlia, 1 AS livello
    FROM sottocategoria sc
    WHERE sc.nomeCategoriaFiglia = @sottocategoria
    UNION ALL
    SELECT sc.nomeCategoriaPadre, sc.nomeCategoriaFiglia, cg.livello + 1
    FROM sottocategoria sc
    JOIN categorie_gerarchia cg ON sc.nomeCategoriaFiglia = cg.categoria_padre
)
SELECT DISTINCT categoria_padre AS categoria
FROM categorie_gerarchia
ORDER BY categoria_padre ASC;
```

La definizione della variabile sottocategoria è superflua, in quanto il valore potrebbe essere inserito direttamente all'interno della query, ma comunque il risultato è valido e funzionante e si sceglie di proseguire con la 1.4.

```
-- Definisci la data di inizio e fine del mese scorso
SET @data_inizio = DATE_FORMAT(LAST_DAY(NOW() - INTERVAL 1 MONTH) + INTERVAL 1
    DAY, '%Y-%m-01');
SET @data_fine = LAST_DAY(NOW() - INTERVAL 1 MONTH);
```

```
-- Query per visualizzare le vendite per ciascun modello con fatturato minore
di 100 nell'ultimo mese
SELECT vp.codModello, m.nome AS nome_modello, COUNT(o.codOrdine) AS
numero_vendite, SUM(vp.prezzo) AS fatturato_totale
FROM ordine o
JOIN richiesta r ON o.codOrdine = r.codOrdine
JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
JOIN modello m ON vp.codModello = m.codModello
WHERE o.data >= @data_inizio AND o.data <= @data_fine
GROUP BY vp.codModello, m.nome
HAVING fatturato_totale < 100
ORDER BY fatturato_totale DESC;
```

Risalta subito un problema di perdita di memoria riguardo la struttura dello schema: si fa uso di attributi come nome per l'entità modello che non esistono nello schema iniziale. Inoltre, il calcolo stesso è incorretto: si considera la somma dei prezzi il fatturato totale e non vengono nuovamente presi in considerazione gli sconti applicati, che siano di quantità o buoni.

Si prova a raffinare puntualizzando che nome non esiste in modello, ma la risposta che si ottiene inizia così:

Chiedo scusa per l'inconveniente. Il campo 'nome' non esiste direttamente nella tabella 'modello', ma è presente nella tabella 'categoria', che è collegata tramite la chiave esterna 'nomeCategoria' nella tabella 'modello'.

Ecco come possiamo modificare la query per visualizzare le vendite per ciascun modello con fatturato minore di 100 nell'ultimo mese, ordinato dal fatturato maggiore al minore, prendendo in considerazione il nome della categoria associata al modello: ...

Per quanto vero che 'nome' esiste in 'categoria' non significa che questo sia correlato alla query in oggetto.

Dato che l'output risulta già così allucinato, anziché proseguire con ulteriori prompt, si sceglie di optare per un altro approccio diverso nella seguente sezione.

### 3.1.3 Schema parziale

Anziché pretendere una scrittura cumulativa di molte query basandosi su uno schema generale fornito solo nel primo prompt, si prova a fornire la sezione di



schema interessata e a richiedere solo una query relativa a quelle determinate entità.

### 3.1.3.1 Composizione filato

#### Query 1.1: visualizzare i filati composti da una certa percentuale di un determinato materiale

Per la prima query si forniscono solo le 3 tabelle necessarie: filato, composizione e materiale. La risposta segue un po' più dettagliata delle precedenti:

Per visualizzare i filati composti da una certa percentuale di un determinato materiale, puoi utilizzare la seguente query SQL. Questa query selezionerà i filati che hanno un'associazione nella tabella 'composizione' con il codice del materiale desiderato e una percentuale specificata.

Assumeremo di voler trovare i filati che sono composti per almeno il 20% di un particolare materiale, identificato dal suo 'codMateriale'.

```
SELECT f.*
FROM filato f
JOIN composizione c ON f.codFilato = c.codFilato
WHERE c.codMateriale = <codice_materiale_desiderato>
AND c.percentuale >= 20; -- Modifica questo valore percentuale a seconda delle
tue esigenze
```

Dove:

- 'codice\_materiale\_desiderato' è il codice del materiale di interesse.
- '20' è la percentuale minima desiderata. Modifica questo valore a seconda della percentuale specifica che stai cercando.

Questa query effettua una join tra le tabelle 'filato' e 'composizione' basata sul campo 'codFilato', filtrando i risultati in base al 'codMateriale' desiderato e alla percentuale specificata nella tabella 'composizione'.

Se hai bisogno di trovare filati che contengono un materiale specifico con una qualsiasi percentuale (senza specificare un valore minimo), puoi semplificare la condizione del 'WHERE' eliminando il filtro sulla percentuale:

```
SELECT f.*
FROM filato f
JOIN composizione c ON f.codFilato = c.codFilato
WHERE c.codMateriale = <codice_materiale_desiderato>;
```

Questa seconda query restituirà tutti i filati che includono il materiale specificato, indipendentemente dalla percentuale esatta.

Il risultato è soddisfacente, offre ben 2 opzioni, sebbene la prima sia quella più coerente alla richiesta. Rispetto ai tentativi precedenti effettua la ricerca tramite il codice del materiale, ma non essendo esplicitamente specificato se sia tramite nome o codice nell'operazione è comunque considerato accettabile.

### 3.1.3.2 Categorie

**Query 1.3: visualizzare le categorie padre di una determinata sottocategoria, ordinando in base al livello di gerarchia**

La query ricorsiva relativa alle categorie viene nuovamente svolta senza ottenere un elenco gerarchico, ma solo quella immediatamente superiore:

```
SELECT c.nome AS categoria_padre
FROM categoria c
JOIN sottocategoria sc ON c.nome = sc.nomeCategoriaPadre
WHERE sc.nomeCategoriaFiglia = '<nome_sottocategoria_desiderata>'
ORDER BY LENGTH(c.nome) DESC;
```

Si prova quindi a richiedere il motivo per cui non vengono mostrate le altre e si corregge utilizzando una combinazione di Common Table Expressions (CTE) e una query ricorsiva. L'unico problema che permane è l'ordinamento: anziché per gerarchia va in base alla lunghezza del nome della categoria. Si chiede quindi di sistemare la visualizzazione in base a ciò, ma in risposta crea procedure, che non corrispondono esattamente all'output desiderato. Rigenerando la risposta si ottiene la query SQL precedentemente creata, ma corretta, con l'unico difetto di un conflitto tra l'uso del DISTINCT e di ORDER BY. Una volta indicato l'errore si ottiene una query funzionante:

```

WITH RECURSIVE CategoryHierarchy AS (
  SELECT
    nomeCategoriaPadre AS categoria_padre,
    nomeCategoriaFiglia AS categoria_figlia,
    1 AS livello
  FROM sottocategoria
  WHERE nomeCategoriaFiglia = '<nome_sottocategoria_desiderata>'

  UNION ALL

  SELECT
    sc.nomeCategoriaPadre,
    ch.categoria_figlia,
    ch.livello + 1
  FROM CategoryHierarchy ch
  JOIN sottocategoria sc ON sc.nomeCategoriaFiglia = ch.categoria_padre
)
SELECT categoria_padre
FROM (
  SELECT DISTINCT categoria_padre, livello
  FROM CategoryHierarchy
  ORDER BY livello DESC
) AS ordered_hierarchy;

```

**Query 1.7: visualizzare tutte le categorie, riportando anche il numero di prodotti che eventualmente vi appartengono**

La query 1.7, sempre fornendo il contesto relativo, viene immediatamente eseguita correttamente, come si può notare a seguire:

```

SELECT c.nome AS categoria, COUNT(vp.codVersione) AS num_prodotti
FROM categoria c
LEFT JOIN modello m ON c.nome = m.nomeCategoria
LEFT JOIN versioneprodotto vp ON m.codModello = vp.codModello
GROUP BY c.nome
ORDER BY c.nome;

```

### 3.1.3.3 Ordini

Si forniscono le tabelle buonosconto, modello, ordine, richiesta, scontoquantità, versioneprodotto, utente e utenteregistrato per ognuna delle query qui affrontata.

### Query 1.2: visualizzare fatturato vendite di questo mese (al netto di sconti quantità e buoni eventualmente applicati nelle vendite)

Nuovamente il primo approccio è risalire alla tabella sconto quantità per calcolare lo sconto: dopo aver specificato di utilizzare l'attributo in richiesta si passa a correggere il calcolo del fatturato.

Si chiede di spiegare il calcolo effettuato per capire la logica utilizzata e si riscontra che lo sconto quantità come anche i buoni vengono considerati con la metrica sbagliata. Si specifica quindi che gli sconti sono registrati già in percentuale (ad esempio 10 equivale a 10%), mentre il buono è direttamente un valore intero. Il primo output considera entrambi percentuali, ma nemmeno proseguendo correggendo anche direttamente il modello spiegando che il buono va semplicemente sottratto, si riesce a ottenere l'output voluto.

Nonostante i numerosi tentativi di rigenerazione e prompting vario, non si riesce ad ottenere un esito positivo. Ecco quindi la query risultante:

```
SELECT SUM(
    r.quantita * vp.prezzo *
    (1 - IFNULL(r.scontoQuantitaApplicato, 0.00)) - IFNULL(bs.importo,
    0.00)
) AS fatturato_vendite
FROM ordine ord
JOIN richiesta r ON ord.codOrdine = r.codOrdine
JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
LEFT JOIN buonosconto bs ON ord.codBuono = bs.codSconto
WHERE MONTH(ord.data) = MONTH(CURRENT_DATE()) AND YEAR(ord.data) = YEAR(
CURRENT_DATE());
```

Mentre la query possibilmente desiderata:

```
SELECT SUM(fatturato_ordine) AS totale_fatturato_mese
FROM (
    SELECT
        o.codOrdine,
        SUM(r.quantita * vp.prezzo)
        - IFNULL(SUM(r.quantita * vp.prezzo * (r.scontoQuantitaApplicato /
        100)), 0)
        - IFNULL(b.importo, 0) AS fatturato_ordine
    FROM ordine o
    JOIN richiesta r ON o.codOrdine = r.codOrdine
    JOIN versioneprodotto vp ON r.codVersione = vp.codVersione
```

```

LEFT JOIN buonosconto b ON o.codBuono = b.codSconto
WHERE MONTH(o.data) = MONTH(CURDATE()) AND YEAR(o.data) = YEAR(CURDATE())
GROUP BY o.codOrdine
) AS fatturati_mese;

```

**Query 1.4: visualizzare le vendite per ciascun modello con fatturato minore di 100 dell'ultimo mese, ordinando dal fatturato maggiore al minore (al netto degli sconti quantità)**

Anche in questo caso è necessario specificare di utilizzare l'attributo in richiesta per il calcolo dello sconto, inoltre viene identificato l'attributo pattern come nome del modello, per questo se ne chiede la rimozione, ottenendo il seguente SQL:

```

SELECT m.codModello, SUM(r.quantita * vp.prezzo * (1 - IFNULL(r.
    scontoQuantitaApplicato, 0))) AS fatturatoNetto
FROM modello m
JOIN versioneprodotto vp ON m.codModello = vp.codModello
JOIN richiesta r ON vp.codVersione = r.codVersione
JOIN ordine o ON r.codOrdine = o.codOrdine
WHERE MONTH(o.data) = MONTH(CURRENT_DATE()) - 1 AND YEAR(o.data) = YEAR(
    CURRENT_DATE())
GROUP BY m.codModello
HAVING fatturatoNetto < 100
ORDER BY fatturatoNetto DESC;

```

Non solo il calcolo risulta incorretto similmente al caso della 1.2, ma ha assunto in maniera errata che l'ultimo mese sia il precedente a quello corrente: nonostante i numerosi tentativi di raffinamento, continua a compiere ciclicamente gli stessi errori. Offrendo esempi e la query 1.2 come analogia i risultati sono ancora più confusi. A seguire la query ideata senza ChatGPT:

```

SELECT v.codModello, SUM(r.quantita * v.prezzo) - IFNULL(SUM(r.quantita * v.
    prezzo * (r.scontoQuantitaApplicato / 100)), 0) AS fatturato_totale
FROM versioneprodotto v
JOIN richiesta r ON v.codVersione = r.codVersione
JOIN modello m ON v.codModello = m.codModello
JOIN ordine o ON r.codOrdine = o.codOrdine
WHERE o.data >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
GROUP BY v.codModello
HAVING fatturato_totale < 100
ORDER BY fatturato_totale DESC;

```

### Query 1.5: visualizzare le versioni dei prodotti che non sono stati ordinati negli ultimi 6 mesi

La query risulta immediatamente corretta:

```
SELECT vp.codVersione, vp.nome
FROM versioneprodotto vp
LEFT JOIN richiesta r ON vp.codVersione = r.codVersione
LEFT JOIN ordine o ON r.codOrdine = o.codOrdine
WHERE o.data IS NULL OR o.data < DATE_SUB(CURRENT_DATE(), INTERVAL 6 MONTH)
GROUP BY vp.codVersione, vp.nome;
```

### Query 1.6: visualizzare gli utenti che hanno ordinato prodotti di tutti i modelli

In questo caso l'output è inizialmente sbagliato, ma specificando di nuovo che si vuole visualizzare solo gli utenti che hanno acquistato tutti i modelli corregge un WHERE EXISTS in WHERE NOT EXISTS e risulta funzionante come segue:

```
SELECT u.codUtente, u.nome, u.cognome, u.email
FROM utente u
WHERE NOT EXISTS (
  SELECT m.codModello
  FROM modello m
  WHERE NOT EXISTS (
    SELECT vp.codVersione
    FROM versioneprodotto vp
    WHERE vp.codModello = m.codModello
    AND EXISTS (
      SELECT 1
      FROM ordine o
      JOIN richiesta r ON o.codOrdine = r.codOrdine
      JOIN versioneprodotto vp2 ON r.codVersione = vp2.codVersione
      WHERE vp2.codModello = m.codModello
      AND o.codUtente = u.codUtente
    )
  )
)
ORDER BY u.codUtente;
```

**Query 1.8: visualizzare l'utente con il maggior numero di ordini per ogni anno**

La query in oggetto risulta immediatamente corretta:

```
WITH OrdiniPerAnno AS (  
  SELECT  
    YEAR(data) AS Anno,  
    o.codUtente,  
    COUNT(*) AS NumeroOrdini,  
    ROW_NUMBER() OVER (PARTITION BY YEAR(data) ORDER BY COUNT(*) DESC) AS  
      RankPerAnno  
  FROM ordine o  
  GROUP BY YEAR(data), o.codUtente  
)  
SELECT u.codUtente, u.nome, u.cognome, u.email, opa.Anno, opa.NumeroOrdini  
FROM OrdiniPerAnno opa  
JOIN utente u ON opa.codUtente = u.codUtente  
WHERE opa.RankPerAnno = 1  
ORDER BY opa.Anno, opa.NumeroOrdini DESC;
```

**Query 2.1: registra una vendita a un utente non registrato (non c'è accumulo di punti) con controllo sconto quantità migliore**

Dal primo output si può subito notare un'ottima comprensione della logica secondo cui creare più query in diverse tabelle per effettuare una registrazione completa della vendita.

Le prime precisazioni da effettuare sono che il codice dell'ordine viene inserito manualmente e non è un valore incrementale e che nella colonna sconto quantità applicato in richiesta non si vuole registrare ad esempio 'sconto 5.00', ma solo il valore percentuale '5.00'.

Dopodiché si ha un valore NULL anziché il codice utente all'interno della registrazione dell'ordine, perché GPT ha interpretato 'utente non registrato' come un utente non presente nel database, invece di considerarlo un'istanza di 'utente'. Chiedendo una correzione esplicita di quest'assunzione, spiegando che il codice utente esiste in 'utente' e correggendo quando senza una motivazione valida lo in-

serisce anche dentro la tabella 'richiesta', si ottiene una sequenza di query coerente e funzionale:

```
-- Identifica il miglior sconto quantita per il prodotto ordinato
SET @quantitaOrdinata := 20; -- Esempio di quantita ordinata
SET @codVersioneProdotto := 1; -- Esempio di codice versione prodotto

SELECT sq.scontoPercentuale
FROM scontoquantita sq
WHERE sq.codVersione = @codVersioneProdotto
      AND sq.quantita <= @quantitaOrdinata
ORDER BY sq.scontoPercentuale DESC
LIMIT 1 INTO @migliorSconto;

-- Specifica il codice ordine da inserire manualmente
SET @codOrdine := 1001; -- Esempio di codice ordine scelto manualmente

-- Codice utente che rappresenta l'utente non registrato
SET @codUtenteNonRegistrato := 4; -- Esempio di codice utente non registrato

-- Inserisci l'ordine nella tabella ordine (utente non registrato, senza punti
  fedelta)
INSERT INTO ordine (codOrdine, data, codUtente, codBuono)
VALUES (@codOrdine, CURDATE(), @codUtenteNonRegistrato, NULL);

-- Visualizza i dettagli dell'ordine appena inserito
SELECT *
FROM ordine
WHERE codOrdine = @codOrdine;
```

### Query 2.2: registra una vendita a un utente registrato (ogni 10 euro accumula un punto fedeltà) con controllo sconto quantità migliore

Anche in questo caso la logica di registrazione risulta molto completa e coerente alla logica dello schema relazionale.

Il prezzo del prodotto non ha notato che si poteva risalire a versione prodotto per ottenerlo, inoltre, il calcolo dei punti lo basa sul prezzo unitario, perciò si spiega la logica da usare per queste condizioni. Nell'effettuare queste correzioni, modifica l'inserimento di 'codOrdine' usando una sua formula anziché il dato manualmente inserito che prima aveva invece considerato. Si chiede di ritornare a usare un valore manuale e, con questi pochi aggiustamenti, si ottiene la query corretta:



```
-- Variabili di esempio per il calcolo
SET @codOrdine := 1002; -- Codice ordine scelto manualmente
SET @quantitaOrdinata := 10; -- Quantita ordinata
SET @codVersioneProdotto := 7; -- Codice versione prodotto
SET @codUtenteRegistrato := 1; -- Codice utente registrato

-- Trova il miglior sconto percentuale per la quantita ordinata
SELECT COALESCE(MAX(sq.scontoPercentuale), 0)
INTO @migliorSconto
FROM scontoquantita sq
WHERE sq.codVersione = @codVersioneProdotto
AND sq.quantita <= @quantitaOrdinata;

-- Calcola l'importo totale prima dello sconto
SELECT vp.prezzo * @quantitaOrdinata
INTO @importoTotale
FROM versioneprodotto vp
WHERE vp.codVersione = @codVersioneProdotto;

-- Calcola l'importo scontato con lo sconto migliore
SELECT
  CASE WHEN @migliorSconto > 0 THEN
    @importoTotale * (1 - @migliorSconto / 100)
  ELSE
    @importoTotale
  END
INTO @importoScontato;

-- Calcola i punti fedelta in base all'importo scontato
SELECT FLOOR(@importoScontato / 10)
INTO @puntiFedelta;

-- Inserisci l'ordine nella tabella ordine
INSERT INTO ordine (codOrdine, data, numPuntiAcquisiti, codUtente, codBuono)
VALUES (
  @codOrdine,
  CURDATE(),
  @puntiFedelta,
  @codUtenteRegistrato,
  NULL
);

-- Visualizza i dettagli dell'ordine appena inserito
SELECT *
FROM ordine
WHERE codOrdine = @codOrdine;
```

## 3.2 Automatizzazione query SQL: dialogare con i dati

Lo scopo di questa sezione è studiare la possibilità di dialogare con i propri dati, compiendo uno step successivo rispetto al prompting e all'implementazione diretta di codice SQL.

Si propone quindi un applicativo Python che consente di interagire con un database MySQL tramite ChatGPT. Questo è reso possibile dall'utilizzo del framework LangChain, che funge da ponte tra i dati presenti nel database e il modello GPT, facilitando la generazione automatica di query SQL e la produzione di risposte in linguaggio naturale.

Uno dei punti di forza nell'utilizzo di questo strumento è proprio la possibilità di offrire al modello il contesto reale dei dati che si hanno a disposizione: senza questo, come si è dimostrato in precedenza, il modello è soggetto ad allucinazioni ed errori molto più facilmente.

Si vuole quindi studiare la differenza che si ottiene con questo tipo di approccio.

### 3.2.1 Langchain

LangChain è un framework open source rilasciato nell'ottobre 2022 da Harrison Chase[25], mentre lavorava presso la startup di machine learning Robust Intelligence. Il tempismo, considerando il boom che ha seguito l'uscita di ChatGPT a novembre del 2022, è stato ottimo: il progetto riceve interesse crescente e guadagna molta popolarità, ottenendo centinaia di contributori su GitHub.

LangChain semplifica lo sviluppo di applicazioni che utilizzano modelli LLM, ad esempio per la creazione di chatbot, per la automatic summarization (creazione di subset rappresentativi di dataset più grandi), come anche per procedimenti quali il retrieval-augmented generation, un processo in due fasi che prevede il recupero dei documenti e la formulazione delle risposte da parte di un LLM, e altro ancora.

In particolare, le chain in LangChain permettono di combinare vari componenti, come input, trasformazioni, modelli LLM, e output, gestendo così il flusso delle

operazioni da svolgere. Questo rende più facile gestire e automatizzare processi complessi che coinvolgono più passaggi di elaborazione, tra i vantaggi:

- **Modularità:** Ogni passo della catena è un componente modulare, rendendo il codice più leggibile e manutenibile.
- **Riutilizzabilità:** I componenti delle chain possono essere riutilizzati in contesti diversi.
- **Composizione:** Combinare diverse operazioni in una sequenza logica è più semplice.
- **Automazione:** Riduce l'intervento manuale e aumenta l'efficienza automatizzando l'intero flusso di lavoro.

Gli strumenti e le API di LangChain sono disponibili nelle librerie basate su JavaScript e Python. A seguire ne vedremo l'applicazione in quest'ultimo linguaggio, connettendo ChatGPT al codice tramite l'API di OpenAI.

## 3.2.2 Applicativo proposto

### 3.2.2.1 Descrizione dell'applicativo

Si descrivono a seguire quelli che sono stati i punti fondamentali nella creazione di un prototipo funzionante con questo framework.

Il codice è diviso in due chain principali:

- **SQL Chain:** Genera una query SQL basata sulla domanda dell'utente e lo schema della tabella, ed è così composta:
  - **Input:** Schema della tabella e domanda dell'utente.
  - **Trasformazione:** Generazione della query SQL usando un prompt e ChatGPT.
  - **Output:** Stringa della query SQL generata.
- **Full Chain:** Esegue la query SQL generata, ottiene il risultato e fornisce una risposta in linguaggio naturale, ed è così composta:

- Input: Domanda dell'utente.
- Trasformazione: Generazione della query SQL con `sql_chain`.
- Esecuzione: Esecuzione della query SQL sul database e ottenimento del risultato.
- Trasformazione Finale: Generazione di una risposta in linguaggio naturale usando un prompt e ChatGPT.
- Output: Risposta finale in linguaggio naturale.

Particolarmente determinante è stato definire dei prompt standard da utilizzare durante il corso delle analisi: in questa maniera, fornendo sempre lo stesso contesto descritto al dettaglio, si ottengono forti differenze positive in termini di comprensione delle richieste.

- Prompt per la generazione del codice SQL

```
template = """In base allo schema della tabella seguente, scrivi una
query SQL che risponda alla domanda dell'utente:
{schema}

Domanda: {question}
SQL Query: """
prompt = ChatPromptTemplate.from_template(template)
```

- Prompt per la risposta in linguaggio naturale

```
template = """In base allo schema della tabella seguente, domanda,
query SQL e risposta SQL, scrivi una risposta in linguaggio
naturale:
{schema}

Domanda: {question}
SQL Query: {query}
SQL Output: {response}"""
prompt_response = ChatPromptTemplate.from_template(template)
```

### 3.2.2.2 Analisi delle query

#### Query in oggetto

Le query prese in considerazione per testare l'applicativo sono solo read, quindi di visualizzazione di informazioni estratte dai dati presenti nel database.

Questo poiché eseguire query di inserimento tramite un LLM come ChatGPT utilizzando LangChain, non è una pratica consigliabile, in quanto presenta diversi rischi e limitazioni, tra cui:

- **Sicurezza:** Strutturare del codice con questo scopo comporta prevenire possibili SQL injections, ovvero attacchi in cui l'aggressore manipola le query SQL per eseguire comandi non autorizzati sul database. Inoltre, gli LLM non sono stati progettati per gestire dati sensibili, perciò questi rischierebbero di essere potenzialmente esposti o gestiti in maniera scorretta. In generale sono quindi possibili perdita e/o corruzione di dati e accesso non autorizzato alle informazioni sensibili.
- **Affidabilità:** Sia per eventuale imprecisione dei prompt, sia per l'interpretazione imprevedibile degli LLM, si può giungere a inserimenti di dati incorretti o incompleti.

### I risultati ottenuti

Le query di visualizzazione eseguite hanno ottenuto quasi tutte esito positivo, spesso al primo tentativo di prompting. Delle 5 ottenute immediatamente, solamente la 1.3 che richiede un'operazione ricorsiva ha necessitato di una piccola riformulazione; anziché chiedere le categorie 'padre' di una sottocategoria sono state chieste le categorie 'sopra' una certa sottocategoria. Questo è bastato per ottenere non solo la categoria padre immediatamente superiore, ma tutto l'elenco. Dimostra di aver anche compreso l'ordine gerarchico, perché quando richiesto di usare un ordine per gerarchia discendente modifica l'output correttamente. È interessante notare come quando si tenta di forzare una correzione diretta chiedendo di usare una query ricorsiva dia un errore probabilmente relativo alla connessione con il database, senza quindi andare a buon fine.

Non si può affermare lo stesso per le query che richiedono di calcolare il fatturato di vendita (1.2 e 1.4) e per quella che vuole visualizzare gli utenti che hanno effettuato più ordini per ogni anno (1.8).

Mentre è comprensibile che possa essere complesso per il modello capire la logica con cui si vuole calcolare il totale delle vendite considerando tutti gli sconti

relativi, nonostante i numerosi tentativi, è comunque particolare che una query statistica come la 1.8 non riesca a ottenere un output completo. Si ottengono solo due tipi di output: l'utente che ha effettuato più ordini nell'anno corrente, oppure un risultato allucinato dove elenca per ogni anno l'utente con più ordini aggiungendo dati relativamente ad anni non registrati.



# Conclusioni sulle prestazioni di ChatGPT per RDBMS

Al termine dell'analisi delle prestazioni del modello GPT 3.5 tramite ChatGPT e LangChain, si può affermare che questo modello risulta essere un potenziale supporto che può rivoluzionare il settore.

Inizialmente è emerso che l'uso di ChatGPT per la progettazione di schemi ER può presentare alcuni svantaggi. In particolare, il sistema ha dimostrato di avere difficoltà nel comprendere alcuni concetti in modo corretto e coerente rispetto alle richieste e ha spesso prodotto risultati imprecisi, contraddittori o sbagliati. Tuttavia, è importante sottolineare che ChatGPT può comunque essere considerato un valido strumento per ottenere spunti e idee preliminari nella fase iniziale della progettazione.

D'altro canto, partendo da uno schema ER ben definito, ChatGPT ha dimostrato di offrire ottime possibilità nella generazione di uno schema relazionale. Sebbene siano emerse alcune imprecisioni, il modello, a seguito di qualche raffinamento, è stato in grado di creare uno schema relazionale coerente e funzionale, che può costituire una solida base per lo sviluppo del database.

Dal punto di vista delle query SQL, fornendo in ogni prompt le porzioni di schema relazionale relative alla richiesta si riesce a ottenere una buona formulazione del codice e con qualche affinamento esperto si può raggiungere un risultato utile. L'applicativo, impostando il prompt sempre per dare il miglior contesto possibile, risulta ancora più immediato nell'offrire un output coerente.

La difficoltà maggiore si è rilevata nell'esecuzione di calcoli che sfruttano dati



presenti in più tabelle, sebbene si siano esplicitati anche i collegamenti logici e persino l'operazione che restituisce il calcolo desiderato. Per le query 1.2 e 1.4 relative ai fatturati con sconti applicati non è stato infatti possibile affidarsi al modello in alcuna forma.

Per quanto riguarda l'esecuzione di query tramite un LLM all'interno di un applicativo ci sono diversi fattori da considerare: in termini di prestazioni potrebbe essere eccessivamente costoso sia in termini di tempo che di risorse e anche la manutenibilità risulta più complessa, rendendo difficile anche effettuare debugging sugli output di un LLM.

Inoltre, si può affermare per garantire un'implementazione robusta e sicura, è ancora fondamentale utilizzare tecniche e strumenti specificamente progettati per la gestione dei database. Nonostante ciò, può essere comunque un potenziale supporto per un esperto del settore.

In conclusione ChatGPT può essere considerato uno strumento ausiliare piuttosto valido per lo sviluppo di RDBMS, ma è comunque sempre necessario un occhio esperto per poter sfruttare criticamente l'output ottenuto, individuando errori e perfezionando la comunicazione col modello. Inoltre, essendo questa una tecnologia in continua evoluzione, si prospetta comunque un forte miglioramento da parte del modello stesso negli anni a venire. Si può quindi sperare in una collaborazione sempre più efficace e raffinata tra uomo e AI.

# Bibliografia

- [1] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.
- [2] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning*. Packt Publishing Ltd., Livery Place 35 Livery Street Birmingham B3 2PB, UK, second edition, September 2017.
- [3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [4] Partha Pratim Ray. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 2023.
- [5] Definizione di intelligenza dall'enciclopedia treccani. URL <https://www.treccani.it/vocabolario/intelligenza/>.
- [6] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005. URL <https://www.taylorfrancis.com/books/mono/10.4324/9781410612403/organization-behavior-hebb>.
- [7] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423, 14602113. URL <http://www.jstor.org/stable/2251299>.

- 
- [8] N. Boldrini. *AI Artificial Intelligence: Come è nata, come funziona e come l'Intelligenza Artificiale sta per cambiare il mondo, la vostra vita e il vostro lavoro*. Class Editori, 2018. ISBN 9788831991100.
- [9] Tom M. Mitchell. *Machine learning*. MacGraw-Hill, 1997.
- [10] Bin Liu. "weak ai" is likely to never become "strong ai", so what is its greatest value for us?, 2021.
- [11] Commissione europea e Direzione generale delle Reti di comunicazione, dei contenuti e delle tecnologie. Proposta di regolamento del parlamento europeo e del consiglio relativo alla governance europea dei dati (atto sulla governance dei dati) - art. 2, n. 1. URL <https://eur-lex.europa.eu/legal-content/IT/TXT/?uri=CELEX%3A52020PC0767&qid=1713963224043>.
- [12] Xiaojin Zhu. Semi-supervised learning tutorial. In *International conference on machine learning (ICML)*, pages 1–135, 2007.
- [13] Yuxi Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018. URL <http://arxiv.org/abs/1810.06339>.
- [14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [15] Minh-Thang Luong. *Neural machine translation*. PhD thesis, Stanford University, 2016.
- [16] Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38, 1994. URL <https://api.semanticscholar.org/CorpusID:59804030>.
- [17] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

- 
- [18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [19] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [20] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- [21] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [22] Konstantinos I Roumeliotis and Nikolaos D Tselikas. Chatgpt and open-ai models: A preliminary review. *Future Internet*, 15(6):192, 2023.
- [23] Anis Koubaa. Gpt-4 vs. gpt-3.5: A concise showdown, 04 2023.
- [24] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [25] Harrison Chase. Langchain, 10 2022. URL <https://github.com/langchain-ai/langchain>".



# Ringraziamenti

Giunta al termine di questo percorso, ci tengo particolarmente a ringraziare alcune persone che hanno svolto un ruolo fondamentale nel sostenermi verso il raggiungimento di quest'obiettivo.

Prima di tutto vorrei ringraziare i miei genitori, senza i quali non sarei qua oggi; grazie per la fiducia incondizionata che avete riposto in me e per avermi supportata nelle mie decisioni con ogni mezzo a disposizione. A mio fratello Ivan, grazie per esserci in qualsiasi momento e per ascoltarmi sempre. Sono molto grata anche ai miei familiari che mi sono stati vicino nonostante la lontananza e mi hanno sempre incoraggiata e consigliata, grazie per la comprensione e il supporto.

Ci tengo davvero molto a ringraziare la mia relatrice Annalisa Franco, che si è resa disponibile a guidarmi nello svolgimento di questa tesi con grande umanità e dedizione, oltre ad avermi formata durante il suo corso. La ringrazio per avermi affidato questo studio e aver accolto le mie proposte con professionalità e interesse, fornendo indicazioni mirate durante la stesura di questo elaborato.

Un ringraziamento va di dovere anche a tutti i compagni di corso che non solo hanno contribuito alla mia crescita professionale e personale ognuno in modo diverso, ma hanno reso questo percorso migliore di quanto potessi immaginare. Grazie per tutti i giorni di studio e non trascorsi insieme, avete reso tutto più bello e non lo dimenticherò mai.

Vorrei inoltre ringraziare il mio fidanzato, con cui ho avuto la fortuna di condividere buona parte di questo percorso. Grazie per la tua presenza, sono felice di saperti al mio fianco e di poterti sostenere a mia volta.

Grazie anche a tutti gli amici di una vita che da sempre mi ascoltano e mi accompagnano; Sara, Alessandro, Irene, Vittoria, Francesca e Sofia, siete una seconda famiglia per me.

Ringrazio anche tutti coloro che, anche solo di passaggio e nelle forme più piccole, hanno comunque contribuito a questo successo.

Infine vorrei ringraziare me stessa per aver ponderato il mio futuro e aver compiuto quel primo passo. Grazie per aver superato ostacoli, dubbi e incertezze in vista di quest'obiettivo e per aver mantenuto la determinazione ferrea che ti ha sempre contraddistinta nonostante tutto.

A tutti voi dedico questo traguardo con grande affetto.

*Sara Romeo*  
*18 Luglio 2024*