Progetto Basi di Dati

Studenti:
PAOLO ADDIS
SAMUELE POZ
TRISTANO MUNINI

ANNO ACCADEMICO 2018-2019

Indice

1 Progettazione Concettuale

1.1 Raccolta ed Analisi dei Requisiti

Si vuole modellare il seguente insieme di informazioni riguardanti un sistema per la gestione delle diagnosi e delle terapie dei pazienti ricoverati in un ospedale.

- Di ogni ricovero, il sistema deve memorizzare il codice univoco, il nome della divisione ospedaliera (Cardiologia, Reumatologia, Ortopedia, ...), il paziente ricoverato, le date di inizio e fine del ricovero ed il motivo principale del ricovero.
- Di ogni paziente, il sistema deve memorizzare il codice sanitario (univoco), il cognome, il nome, la data di nascita, il luogo di nascita e la provincia di residenza. Per i pazienti residenti fuori regione, vengono memorizzati anche il nome della ULSS e la regione di appartenenza. Inoltre, per ogni paziente, è necessario memorizzare il totale delle durate dei ricoveri.
- Di ogni diagnosi effettuata durante il ricovero, sono memorizzati la patologia diagnosticata, col suo codice ICD10 (classificazione internazionale delle patologie) e l'indicazione della sua gravità (grave: si/no), la data ed il nome ed il cognome del medico che ha effettuato la diagnosi.
- Nella base di dati si tiene traccia delle terapie prescritte ai pazienti durante il ricovero. Di ogni terapia, si memorizzano il farmaco prescritto, la dose giornaliera, le date di inizio e di fine della prescrizione, la modalità di somministrazione ed il medico che ha prescritto la terapia.
- Di ogni farmaco sono memorizzati il nome commerciale (univoco), l'azienda produttrice, il nome e la quantità dei principi attivi contenuti e la dose giornaliera raccomandata.
- Si tiene, infine, traccia delle diagnosi che hanno motivato le terapie. In particolare, ogni terapia è prescritta al fine di curare una o più patologie diagnosticate. Può capitare anche che una nuova patologia (registrata come nuova diagnosi) sia causata, come effetto collaterale, da una terapia precedentemente prescritta. Tale legame causa-effetto va registrato nella base di dati.
- L'ospedale, in una previsione di 10 anni, prevede di gestire un volume pari a 10000 pazienti, 30000 ricoveri, 120000 diagnosi, 30000 terapia e 110000 prescrizioni. Inoltre, l'esigenza dell'ospedale è effettuare efficacemente l'accesso alle diagnosi dei pazienti per revisione e/o modifica.

Tramite un portale web dovrà essere possibile accedere a tale base di dati ed aggiungere o rimuovere pazienti, ricoveri, diagnosi con relative terapie e farmaci.

1.2 Progettazione del Modello E-R

La progettazione dello schema Entità-Relazioni è avvenuta in tre fasi. Nella prima fase, la prima entità modellata è stata RICOVERO che memorizza tutte le informazioni relative ad un ricovero e possiede come attributi: *Codice Ricovero*, *Divisione Ospedaliera*, *Paziente Ricoverato*, *Data Inizio*, *Data Fine*, *Motivo*. In particolare, l'attributo *Codice Ricovero* è stato scelto come chiave primaria a causa della sua proprietà di unicità. Successivamente è stata modellata l'entità PAZIENTE che possiede i seguenti attributi: *Codice Fiscale*, *Cognome*, *Nome*, *Data di Nascita*, *Luogo di Nascita* e *Provincia di residenza*. L'attributo *Codice Fiscale* è stato scelto come chiave primaria in quanto unico per ogni paziente. Inoltre, è richiesto di memorizzare informazioni aggiuntive per i pazienti fuori regione. Per far ciò è stata creata una specializzazione PAZIENTE_FUORI_REGIONE di PAZIENTE i cui attributi sono: *Regione di Appartenenza* e *ULSS*. Nei requisiti sul paziente è richiesto, inoltre, di tener traccia del tempo totale trascorso all'interno dell'ospedale durante i ricoveri. Quest'informazione è modellata attraverso *Totale Giorni Ricoveri*, un attributo derivato su paziente che è la somma di tutte le durate dei ricoveri riguardanti il paziente. La relazione di afferenza tra RICOVERO e PAZIENTE prende il nome di VIENE_RICOVERATO. L'informazione memorizzata nell'attributo *Paziente Ricoverato* di RICOVERO viene salvata anche attraverso la nuova relazione, di conseguenza, per evitare ridondanze, è possibile rimuovere l'attributo dall'entità. Un nuovo paziente viene inserito nella base di dati la prima volta in cui viene ricoverato, quindi la partecipazione di PAZIENTE alla relazione VIENE_RICOVERATO è totale. Anche RICOVERO partecipa in modo totale alla relazione, in quanto sarebbe insensato inserire i dati di un ricovero senza specificare a chi si riferiscano. Ogni paziente può essere ricoverato più volte ma ogni ricovero riguarda un solo paziente, dunque la relazione VIENE_RICOVERATO è della forma Uno-a-Molti.
Successivamente è stata modellata l'entità DIAGNOSI che possiede come attributi: *Data*, *Medico* e *Patologia*, un attributo composto formato da *Codice* e *Gravità*. La chiave primaria è composta dall'attributo *Data* e dalla relazione EFFETTUATA_DURANTE. Quest'ultima, del tipo Uno-a-Molti, lega l'entità RICOVERO a DIAGNOSI. In particolare una diagnosi verrà inserita solo se nella base di dati è già presente il ricovero a cui afferisce, quindi la partecipazione di DIAGNOSI alla relazione è totale. Sarebbe perfettamente lecito, invece, avere un ricovero durante il quale non è stata effettuata alcuna diagnosi, questo stabilisce la partecipazione parziale di RICOVERO in EFFETTUATA_DURANTE. Ora è importante introdurre due vincoli di integrità:

- il primo deve assicurare che nessun ricovero possa iniziare prima della data di nascita del paziente a cui si riferisce;
- il secondo riguarda la data in cui viene effettuata la diagnosi, in quanto deve essere compresa tra la data di inizio e quella di fine del ricovero.

Il Risultato della prima fase della progettazione è illustrata in Figura 1.

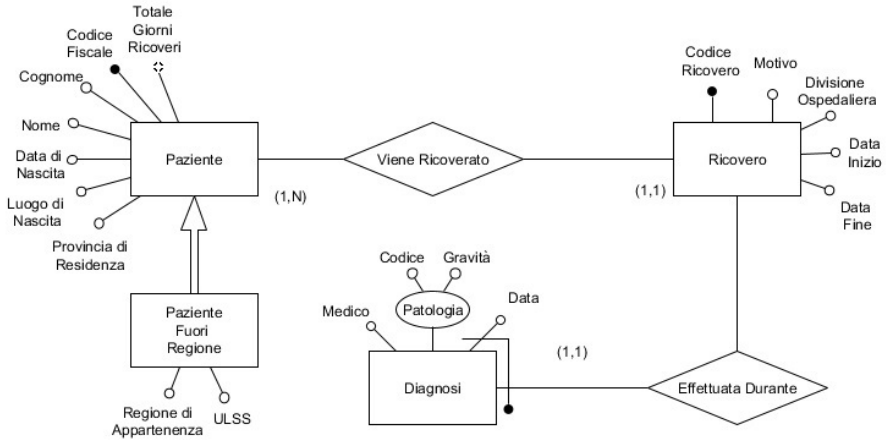

Figura 1: Prima parte

La seconda fase considera il quarto, il quinto e parte del sesto punto dell'analisi dei requisiti, nei quali si chiede di tener traccia delle terapie e dei farmaci utilizzati. Per questi ultimi è stata creata l'entità FARMACO con attributi: *Dose Giornaliera Raccomandata*, *Nome Commerciale*, *Azienda Produttrice* e due attributi multivalore *Nome Principi Attivi* e *Quantità Principi Attivi*. La chiave primaria è data dall'attributo *Nome Commerciale* in quanto univoco. Un farmaco deve contenere uno o più principi attivi.
I requisiti richiedono che una terapia possa curare più patologie, di conseguenza una terapia deve rappresentare un metodo standard con cui affrontare varie patologie. Quindi l'entità TERAPIA non può contenere informazioni come data di inizio, data di fine oppure medico prescrivente, altrimenti non sarebbe più generale. Per questi motivi all'entità sono stati attribuiti i campi: *Dose Giornaliera* e *Modalità Somministrazione*. Le richieste dei requisiti quali data di inizio e di fine e medico prescrivente della terapia verrano trattate nella terza fase di progettazione. La chiave di TERAPIA è composta dagli attributi *Dose Giornaliera* e *Modalità Somministrazione* e dalla relazione SOMMINISTRATO_DURANTE che lega FARMACO a TERAPIA. Questa relazione è della forma Uno-a-Molti in quanto in ogni terapia si somministra un solo farmaco mentre un farmaco può essere somministrato in più terapie. È importate considerare il fatto che una terapia non può esistere senza un farmaco da somministrare, quindi la partecipazione di TERAPIA alla relazione SOMMINISTRATO_DURANTE è totale. D'altro canto un farmaco può essere memorizzato nella base di dati pur non essendo associato ad una particolare terapia, quindi la partecipazione di FARMACO alla precedente relazione è parziale. Il risultato della seconda fase di progettazione è illustrata nella Figura 2.

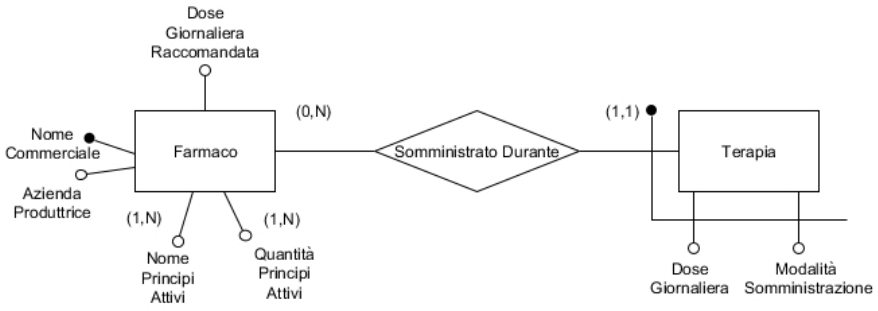

Figura 2: Seconda parte

Nella terza fase della progettazione è stato modellato l'ultimo punto dell'analisi dei requisiti attraverso la relazione MOTIVA_TERAPIA. Particolare attenzione è stata fatta alla parte in cui si specifica che una terapia può essere associata ad una diagnosi in due modi differenti: quello che la definisce come cura per una patologia e quello che la definisce come causa di una nuova patologia. Va sottolineato che in entrambi i casi, nella base di dati, la patologia è salvata attraverso DIAGNOSI.

In un primo momento è stato modellato il sottoschema illustrato in Figura 3. Il modo in cui è stata definita l'entità TERAPIA permette di modellare il requisito *"ogni terapia può curare più patologie"*. Un'informazione come data di inizio, o data di fine, o medico prescrivente, non può essere memorizzata in TERAPIA perché, altrimenti, non sarebbe possibile applicare questa stessa terapia in uno spazio temporale differente, pena la perdita delle informazioni precedenti. D'altra parte non è nemmeno accettabile includere nella chiave primaria le informazioni relative alle date ed al medico, perché così facendo si differenzierebbero le entità, andando contro il requisito che si vuole modellare. Da qui la necessità di omettere attributi data inizio, data fine e medico prescrivente in TERAPIA, assegnandoli alla relazione MOTIVA_TERAPIA. Intuitivamente questi attributi riguardano contemporaneamente la diagnosi, quindi la patologia che si vuole curare, e la terapia applicata, quindi il modo in cui si vuole affrontare la patologia, indicando il periodo in cui la terapia viene applicata e chi ha definito questo periodo. Nei requisiti è specificato che per alcune patologie mancano le cure, per questo motivo la partecipazione a MOTIVA_TERAPIA di DIAGNOSI è parziale. Invece è sensato pensare che, se una terapia esiste, può curare almeno una patologia, quindi la partecipazione di TERAPIA alla relazione è totale. Nel complesso MOTIVA_TERAPIA, per ora, risulta essere una relazione Uno-A-Molti: nel testo è specificato che ad ogni diagnosi può essere associata al più una terapia. Mentre TERAPIA è stata definita appositamente per poter esser associata a più DIAGNOSI.

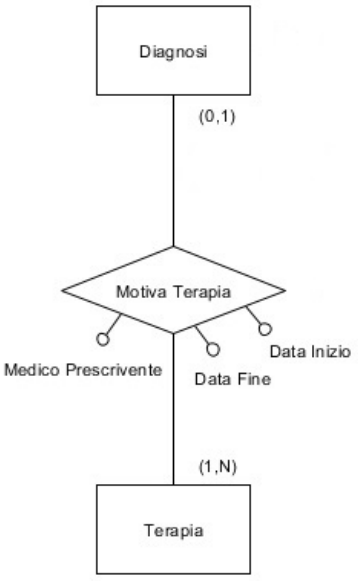

Figura 3: Prima modellazione

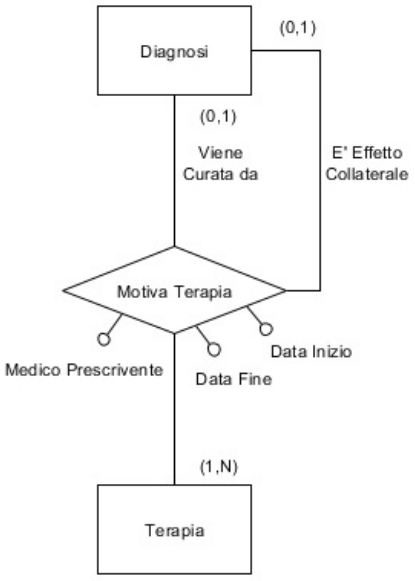

Figura 4: Relazione completa

Successivamente è stato necessario modellare la situazione in cui una terapia prescritta è causa di una nuova patologia. L'informazione che si vuole mantenere non dipende esclusivamente dall'entità TERAPIA ma dipende anche dal periodo di applicazione, quindi non è sufficiente creare una nuova relazione tra TERAPIA e DIAGNOSI, in quanto le informazioni aggiuntive sulla terapia sono memorizzate sulla relazione che le lega. La soluzione migliore è quella di trasformare MOTIVA_TERAPIA in una relazione ternaria: con DIAGNOSI che partecipa da una parte come patologia in cura e dall'altra come rilevazione di una nuova patologia. Per comodità, il risultato di tale trasformazione è riportato in Figura 4. Ricordando che non tutte le applicazioni di una terapia causano un effetto collaterale e che ne possono causare al massimo uno, la partecipazione di DIAGNOSI come effetto collaterale è parziale con cardinalità 1.

Al termine di questa terza fase di progettazione è necessario introdurre dei vincoli d'integrità per la relazione ternaria:

- in nessuna tripletta di MOTIVA_TERAPIA può comparire due volte la stessa DIAGNOSI;
- le DIAGNOSI devono essere state effettuate durante lo stesso ricovero;
- la DIAGNOSI che partecipa come effetto collaterale deve avere una data successiva alla data della prima diagnosi, ma comunque precedente alla data di fine ricovero.

Nella fase logica verrà mostrato che questa relazione può essere trasformata, senza perdita d'informazione, con l'aggiunta di una nuova entità, in tre relazioni binarie.

Nei requisiti è richiesto di memorizzare il medico che ha effettuato la diagnosi ed il medico che ha prescritto la terapia. Ciò è stato fatto riportando rispettivamente le due informazioni nell'attributo *Medico* in DIAGNOSI e nell'attributo *Medico Prescrivente* in MOTIVA_TERAPIA. Non è stata introdotta un'entità MEDICO perché avrebbe contenuto soltanto l'informazione riguardante nome e cognome del medico. Inoltre non è richiesto di tener traccia dello storico delle prescrizioni o delle diagnosi rispetto ai medici. In conclusione lo schema Entità-Relazioni si presenta come in Figura 5.

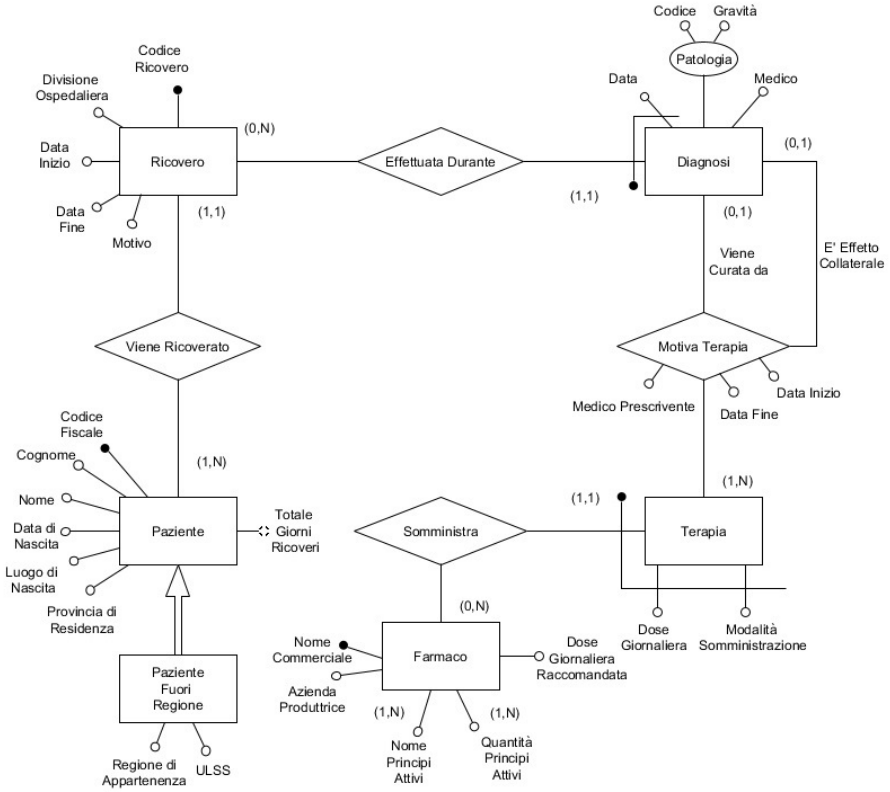

Figura 5: Lo schema Entità-Relazioni completo

2 Progettazione Logica

2.1 Ristrutturazione del Modello E-R

È utile semplificare strutture come specializzazioni, attributi composti e attributi multivalore perché renderà più facile la traduzione del modello Entità-Relazioni in quello Relazionale.

2.1.1 Semplificazione dei Concetti

Il modello relazionale non permette di rappresentare generalizzazioni dunque sarà necessario rimuovere tutte le specializzazioni dal nostro schema E-R. Si può notare che la specializzazione PAZIENTE_FUORI_REGIONE può essere facilmente rappresentata aggiungendo due attributi all'entità PAZIENTE. I nuovi attributi, *ULSS* e *Regione di Appartenenza*, in PAZIENTE saranno facoltativi poiché la specializzazione è parziale. La rimozione delle generalizzazioni non è sempre così immediata: se PAZIENTE_FUORI_REGIONE avesse avuto una relazione aggiuntiva rispetto al genitore si sarebbero dovute valutare varie opzioni. Per esempio la creazione di una nuova relazione tra genitore e figlio per mantenere il legame con gli attributi facoltativi. Oppure la duplicazione nel figlio di tutti gli attributi e di tutte le relazioni del genitore.

Anche nel caso degli attributi composti la semplificazione è intuitiva, infatti nello schema l'unico attributo composto è *Patologia* in DIAGNOSI. L'attributo *Patologia* può essere trasformato in due nuovi attributi *Codice Patologia* e *Gravità Patologia* contenenti esattamente la stessa informazione.

Bisogna prestare più attenzione agli attributi multivalore: in FARMACO sono presenti *Nome Principi Attivi* e *Quantità Principi Attivi* entrambi obbligatori e multivalore. L'unica semplificazione attuabile è creare una nuova entità chiamata PRINCIPIO_ATTIVO e metterla in relazione Molti-A-Molti con FARMACO attraverso la nuova relazione CONTIENE. Poiché i nomi dei principi attivi possono essere molto lunghi e non hanno un formato standard, in PRINCIPIO_ATTIVO, sono salvati nella variabile *Nome*, mentre una sequenza numerica chiamata *Codice Principio Attivo* viene usata come chiave primaria. Alla relazione CONTIENE è stato aggiunto l'attributo *Quantità*. In questo modo saranno modellate esattamente le stesse informazioni precedentemente modellate dagli attributi multivalore. In realtà ora le informazioni sono salvate in un formato meno equivoco: prima non era chiaro come si mantenessero in relazione il nome del principio attivo e la sua quantità.

Un'ulteriore trasformazione da effettuare riguarda la relazione ternaria. Infatti MOTIVA_TERAPIA, come è stato accennato in precedenza, può essere espressa tramite relazioni binarie. In Figura 6 si può vedere il risultato di tale trasformazione. Si nota subito la nuova entità TERAPIA_PRESCRITTA, crea-

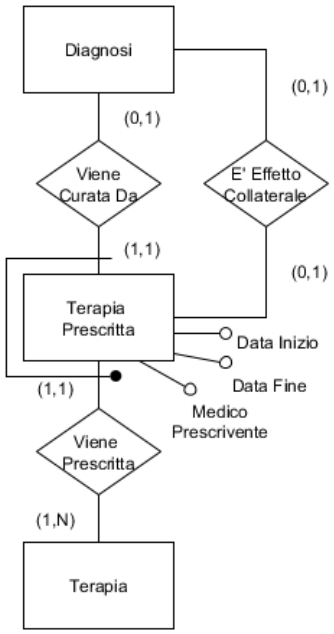

Figura 6: Il risultato della trasformazione di MOTIVA_TERAPIA

ta proprio per rappresentare una particolare applicazione di una terapia al fine di curare una patologia di una specifica diagnosi. La nuova entità, quindi, è caratterizzata da: *Data Inizio* che indica il giorno in cui il paziente dovrà iniziare la terapia; *Data Fine* che indica il giorno in cui il paziente terminerà la terapia; *Medico Prescrivente* che indica il medico che ha prescritto la terapia e deciso il periodo di applicazione. Osservando che la relazione MOTIVA_TERAPIA non può esistere senza la partecipazione simultanea di DIAGNOSI (come *Viene Curata da*) e di TERAPIA si è deciso di rendere TERAPIA_PRESCRITTA un'entità debole. La chiave primaria è composta dalle nuove relazioni binare VIENE_CURATA_DA e VIENE_PRESCRITTA, per entrambe la partecipazione di TERAPIA_PRESCRITTA è totale, mentre la cardinalità è al massimo uno. DIAGNOSI e TERAPIA interagiscono con le nuove relazioni, rispettivamente, nello stesso modo con cui interagivano con la relazione ternaria. Per quanto riguarda È_EFFETTO_COLLATERALE si può dire che cattura la possibilità che una terapia prescritta possa causare una nuova patologia. Notare che le partecipazioni sono appunto parziali. Bisognerà modificare oppure aggiungere alcuni vincoli d'integrità:

- TERAPIA_PRESCRITTA non può essere in relazione È_EFFETTO_COL-

LATERALE con la DIAGNOSI con cui definisce la sua chiave primaria;

- TERAPIA_PRESCRITTA non può essere in relazione È_EFFETTO_COLLATERALE con una DIAGNOSI avente una data che precede la data della DIAGNOSI con cui definisce la sua chiave primaria;
- la data d'inizio della TERAPIA_PRESCRITTA deve essere compresa tra la data della DIAGNOSI che la definisce e la data della DIAGNOSI che identifica l'effetto collaterale;
- la data di fine della TERAPIA_PRESCRITTA non può precedere la sua data di inizio.

A questo punto, osservando nuovamente i sottoschemi in Figura 4 ed in Figura 6, è chiaro che modellino ogni possibile situazione:

- una DIAGNOSI a cui non è ancora stata assegnata una terapia;
- la prescrizione di una specifica terapia;
- la scoperta di una nuova patologia (attraverso una nuova diagnosi) causata dall'applicazione di una terapia;
- la possibilità che una TERAPIA possa curare più patologie (registrate attraverso DIAGNOSI differenti).

Durante la fase di semplificazione dei concetti vengono analizzate anche le chiavi primarie delle entità. Quando una chiave è troppo complessa, cioè composta da molti elementi, è bene sostituirla, così facendo si faciliteranno tutte le operazioni sulla base di dati. Si ricorda che un'entità può disporre di chiavi candidate, cioè insiemi di attributi che, per loro natura, rispettano la proprietà di chiave primaria ma che, per qualche motivo, non sono stati scelti come tale. Ora è naturale verificare se tra queste chiavi opzionali ce n'è una che, dopo tutte le semplificazioni effettuate allo schema E-R, è migliore della chiave primaria. Purtroppo questa strada non è sempre percorribile, infatti non è raro che le entità siano tra loro distinguibili soltanto per mezzo di un preciso insieme di attributi. Nello schema E-R considerato fino ad ora non sono presenti chiavi candidate migliori delle chiavi primarie attuali, infatti la chiave primaria della maggior parte delle entità è composta da un solo attributo.

Rimangono da analizzare DIAGNOSI, TERAPIA e la nuova TERAPIA_PRESCRITTA. La chiave primaria di quest'ultima, per il modo in cui l'entità è stata costruita, non può essere modificata a meno di introdurre pesanti vincoli di integrità. Ora sorge un problema: TERAPIA_PRESCRITTA è un'entità debole che dipende da due entità a loro volta deboli. Ciò significa che per identificare una terapia prescritta si deve risalire al ricovero da una parte ed al farmaco dall'altra, nel mentre si deve anche ricordare *Data* in DIAGNOSI

e *Dose Giornaliera* e *Modalità Somministrazione* in TERAPIA. Intuitivamente questo equivale ad avere una chiave primaria in TERAPIA_PRESCRITTA composta da 5 elementi. Quindi si propone di creare sia in DIAGNOSI che in TERAPIA un nuovo attributo chiamato, rispettivamente, *Codice Diagnosi* e *Codice Terapia* che conterrà un valore numerico simile a quello contenuto in *Codice Ricovero*.

Applicando tutte le semplificazioni sopra descritte si ottiene lo schema E-R illustrato In Figura 7.

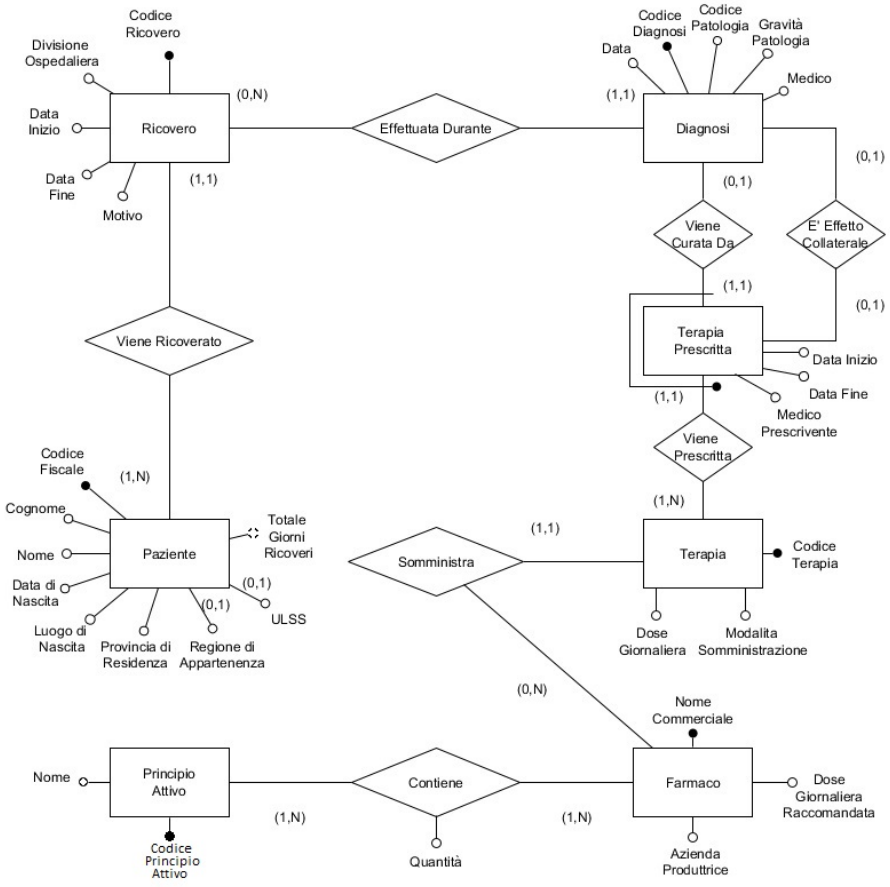

Figura 7: Lo schema Entità-Relazioni ristrutturato

2.1.2 Analisi delle Ridondanze

Gli attributi derivati, come può suggerire il nome, conservano dati già presenti nella base di dati, quindi introducono ridondanza. Si ricorda che quest'ultima, quando si progetta una base di dati, è da evitare, non tanto perché l'informazione ripetuta occupa spazio doppio, ma piuttosto perché si rischia di introdurre pericolose inconsistenze. Detto ciò, gli attributi derivati sono utili se mantengono informazione che altrimenti sarebbe dispendioso ottenere, oppure informazione richiesta molto spesso. In entrambi i casi un attributo derivato velocizza il recupero del dato, introducendo però del carico aggiuntivo in fase di inserimento od aggiornamento dello stesso.

Sfruttando i volumi stimati durante l'analisi dei requisiti si valuterà se conviene o meno mantenere l'attributo derivato *Totale Giorni Ricoveri* dell'entità PAZIENTE. In Figura 8 è stata riportata una sezione dello schema relazionale.

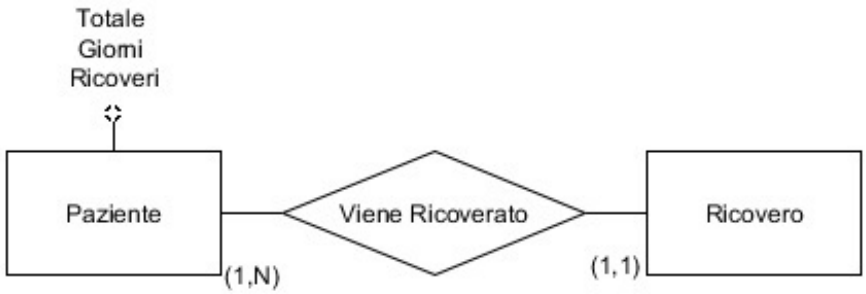

Figura 8: Attributo derivato Totale Giorni Ricoveri

I volumi stimati sono riportati in Tabella 1 e la frequenza delle operazioni interessate in Tabella 2.

Concept	**Type**	**Volume**
Paziente	E	100000
Ricovero	E	300000
Viene Ricoverato	R	300000

Tabella 1: Tabella dei volumi ridotta

Operation	**Type**	**Frequency**
Registrazione Ricovero	I	175/giorno
Controllo totale giorni di ricoveri	I	175/giorno

Tabella 2: Tabella delle operazioni ridotta

In presenza dell'attributo derivato, per l'operazione di registrazione di un nuovo ricovero è necessario 1 accesso in scrittura per l'entità RICOVERO, 1 accesso in scrittura per la relazione VIENE_RICOVERATO, 1 accesso in lettura ed 1 accesso in scrittura per l'entità PAZIENTE per leggere e aggiornare l'attributo *Totale Giorni Ricoveri*. Per l'operazione di verifica del totale dei giorni di ricoveri è sufficiente 1 accesso in lettura all'entità PAZIENTE.

Concept	Type	Access	Type
Registrazione Ricovero			
Ricovero	E	1	W
Viene Ricoverato	R	1	W
Paziente	E	1	R
Paziente	E	1	W
Controllo totale giorni di ricoveri			
Paziente	E	1	R

Tabella 3: Tabella dei costi, caso con ridondanza

Considerando il costo di 1 accesso in scrittura come 2 accessi in lettura, si calcola il numero di accessi complessivo giornaliero per le 2 operazioni nel caso con ridondanza:

$$(3W + 1R) \times 175 + 1R \times 175 \approx 1400R \tag{1}$$

Rimuovendo l'attributo, per l'operazione di registrazione di un nuovo ricovero è sufficiente 1 accesso in scrittura per l'entità RICOVERO e 1 accesso in scrittura per la relazione VIENE_RICOVERATO. Per l'operazione di verifica del totale dei giorni di ricoveri, considerando la tabella dei volumi, sono necessari in media 3 accessi in lettura per la relazione VIENE_RICOVERATO e 3 accessi in lettura per l'entità RICOVERO.

Concept	Type	Access	Type
Inserimento Diagnosi			
Viene Ricoverato	R	1	W
Ricovero	R	1	W
Stampa Storico Diagnosi			
Viene Ricoverato	R	3	R
Ricovero	E	3	R

Tabella 4: Tabella dei costi, caso senza ridondanza

Considerando il costo di 1 accesso in scrittura come 2 accessi in lettura, si calcola il numero di accessi complessivo giornaliero per le 2 operazioni nel caso senza ridondanza:

$$2W \times 175 + 6R \times 175 \approx 1750R \tag{2}$$

In conclusione, risulta conveniente mantenere l'attributo *Totale Giorni Ricoveri*.

Lo schema Entità-Relazioni in Figura 7 non presenta ulteriori ridondanze. Il ciclo formato dalle entità DIAGNOSI, TERAPIA_PRESCRITTA e DIAGNOSI

non crea ridondanze perché ogni elemento fornisce un'informazione unica: la prima diagnosi giustifica la cura, la terapia prescritta spiega quando tale cura è stata applicata, mentre la seconda diagnosi riporta eventuali effetti collaterali.

Nei requisiti è specificato che l'ospedale visualizza molto spesso le diagnosi dei pazienti, senza richiedere necessariamente i dati relativi ai ricoveri. Per questo motivo si è considerata la possibilità di aggiungere una nuova relazione:

- EFFETTUATA_A: una relazione Uno-A-Molti tra PAZIENTE e DIAGNOSI. La partecipazione di PAZIENTE è parziale mentre quella di DIAGNOSI è totale.

Viene creato quindi il ciclo in Figura 9 PAZIENTE, RICOVERO, DIAGNOSI dato dalle relazioni VIENE_RICOVERATO, EFFETTUATA_DURANTE, EFFETTUATA_A.

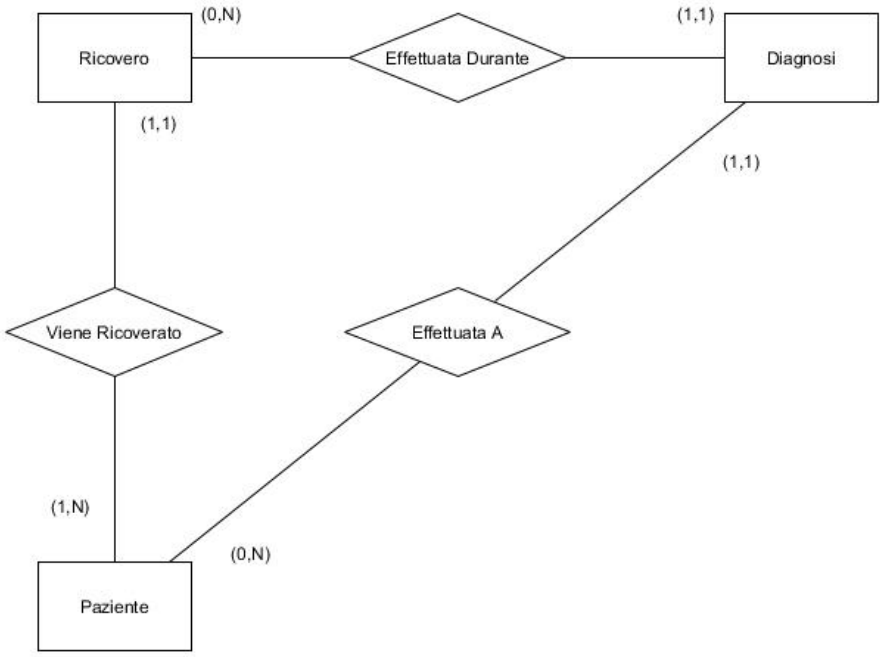

Figura 9: Ciclo con ridondanza

Si procede con il valutare se è conveniente mantenere la relazione EFFETTUATA_A tra PAZIENTE e DIAGNOSI in funzione della frequenza delle operazioni di inserimento nuove diagnosi e stampa dello storico diagnosi di un paziente.

I volumi stimati sono riportati in Tabella 6 e la frequenza delle operazioni interessate in Tabella 7

Concept	**Type**	**Volume**
Paziente	E	100000
Ricovero	E	300000
Diagnosi	E	1200000
Terapia	E	30000
Istanza di Terapia	E	1100000
Farmaco	E	10000
Viene Ricoverato	R	300000
Effettuata Durante	R	1200000
Effettuta A	R	1200000
Curata Da	R	1100000
Istanza Di	R	1100000
Effetto Collaterale	R	50000
Somministrato Durante	R	10000

Tabella 5: Tabella dei volumi

Concept	**Type**	**Volume**
Paziente	E	100000
Ricovero	E	300000
Diagnosi	E	1200000
Viene Ricoverato	R	300000
Effettuata Durante	R	1200000
Effettuata A	R	1200000

Tabella 6: Tabella dei volumi ridotta

Operation	**Type**	**Frequency**
Inserimento Diagnosi	I	700/giorno
Stampa Storico Diagnosi	I	1400/giorno

Tabella 7: Tabella delle operazioni ridotta

Nel caso con ridondanza per l'operazione di inserimento di una nuova diagnosi sono necessari 1 accesso in scrittura per l'entità DIAGNOSI e 1 accesso in scrittura per entrambe le relazioni EFFETTUATA_DURANTE e EFFETTUATA_A. Per l'operazione di stampa delle diagnosi di un paziente sono necessari 12 accessi in lettura per la relazione EFFETTUATA_A e 12 accessi in lettura per l'entità DIAGNOSI.

Considerando il costo di 1 accesso in scrittura come 2 accessi in lettura, si calcola il numero di accessi complessivo giornaliero per le 2 operazioni nel caso con ridondanza:

$$3W \times 700 + 24R \times 1400 \approx 37800R \tag{3}$$

Concept	**Type**	**Access**	**Type**
Inserimento Diagnosi			
Diagnosi	E	1	W
Effettutata Durante	R	1	W
Effettuata A	R	1	W
Stampa Storico Diagnosi			
Effettuata A	R	12	R
Diagnosi	E	12	R

Tabella 8: Tabella dei costi, caso con ridondanza

Nel caso senza ridondanza per l'operazione di inserimento di una nuova diagnosi sono necessari 1 accesso in scrittura per l'entità DIAGNOSI e 1 accesso in scrittura per la relazione EFFETTUATA_DURANTE. Per l'operazione di stampa delle diagnosi di un paziente sono necessari 3 accessi in lettura per la relazione VIENE_RICOVERATO, 12 accessi in lettura per la relazione EFFETTUATA_DURANTE e 12 accessi in lettura per l'entità DIAGNOSI.

Considerando il costo di 1 accesso in scrittura come 2 accessi in lettura, si calcola il numero di accessi complessivo giornaliero per le 2 operazioni nel caso senza ridondanza:

$$2W \times 700 + 27R \times 1400 \approx 40600R \tag{4}$$

Concept	**Type**	**Access**	**Type**
Inserimento Diagnosi			
Diagnosi	E	1	W
Effettutata Durante	R	1	W
Stampa Storico Diagnosi			
Viene Ricoverato	R	3	R
Effettutata Durante	R	12	R
Diagnosi	E	12	R

Tabella 9: Tabella dei costi, caso senza ridondanza

Concludiamo che è conveniente mantenere la ridondanza visto il costo minore delle operazioni giornaliere.

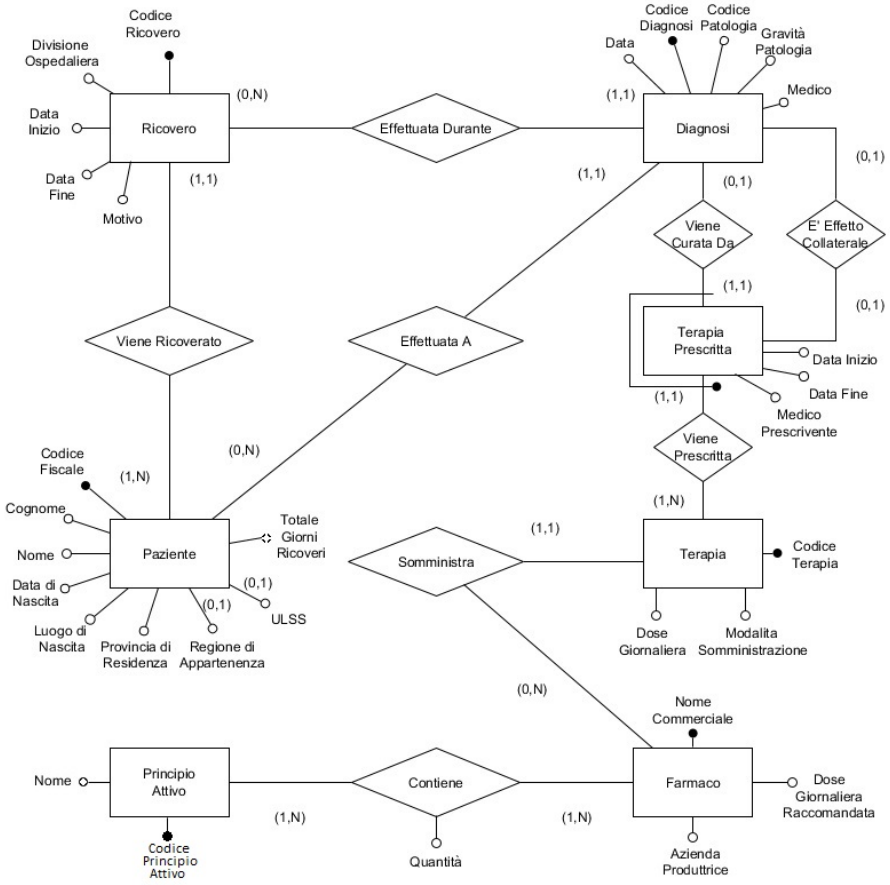

Figura 10: Lo schema Entità-Relazioni finale

2.2 Traduzione da ER a Relazionale

In questa fase della progettazione logica è possibile iniziare la traduzione del modello E-R in quello Relazionale. Come prima cosa, in quanto immediato, ogni entità verrà rappresentata sotto forma di tupla. Queste tuple verranno poi modificate per poter mantenere anche le informazioni relative alle relazioni, può capitare che ad una relazione venga associata un nuova tupla.

In Figura 11 si può vedere una prima modellazione dello schema Relazionale, con le seguenti tuple:

- PAZIENTE(*cf, cognome, nome, data_nasc, luogo_nasc, prov_res, reg_app, ulss, tot_gg_ric*) con *cf* come chiave primaria.
- RICOVERO(*cod_ric, data_i, data_f, motivo, div_osp*) con *cod_ric* come chiave primaria;
- DIAGNOSI(*cod_dia, data_dia, cod_pat, medico*) con *cod_dia* come chiave primaria;
- TERAPIA(*cod_ter, dose_gio, mod_somm, farmaco*) con *cod_ter* come chiave primaria;
- FARMACO(*nome_comm, azienda_prod, dose_gg_racc*) con *nome_comm* come chiave primaria;
- PRINCIPIO_ATTIVO(*cod_pa*, *nome*) con *cod_pa* come chiave primaria.

Bisogna prestare più attenzione a TERAPIA_PRESCRITTA perché, in quanto entità debole, la sua chiave primaria dipende dalle relazioni VIENE_CURATA_DA e VIENE_PRESCRITTA, o meglio, comprende degli identificatori esterni. Nel dettaglio un identificatore sarà dato dalla chiave primaria di DIAGNOSI mentre l'altro dalla chiave primaria di TERAPIA. Si ricorda che le entità identificate esternamente partecipano alle associazioni identificanti sempre con una cardinalità minima e massima pari a uno. Quindi, nonostante le due relazioni siano di tipi differenti (Uno-A-Uno ed Uno-A-Molti), possono essere trattate allo stesso modo, cioè aggiungendo alla tupla

TERAPIA_PRESCRITTA(*data_i, data_f, med_presc*)

i nuovi valori referenziali *diagnosi* e *terapia*. Questi, rispettivamente, conterranno il valore *cod_dia* e *cod_ter* delle tuple dalle quali TERAPIA_PRESCRITTA dipende. La tupla risultante è la seguente:

TERAPIA_PRESCRITTA(*diagnosi, terapia, data_i, data_f, med_presc*)

la chiave primaria è composta dalla coppia (*diagnosi, terapia*).

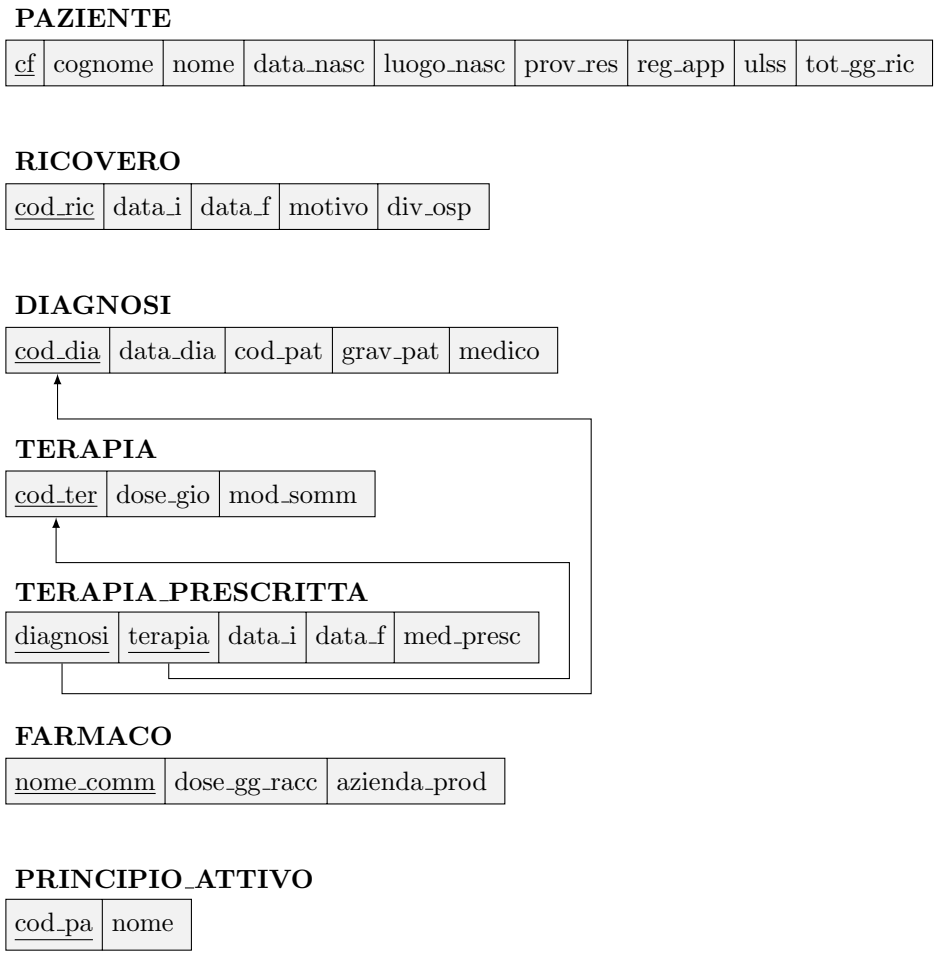

Figura 11: Primo modello relazionale

In questi passaggi, in realtà, è stato fatto qualcosa di più di una semplice traduzione di una entità dal modello E-R a quello Relazione, infatti sono state tradotte anche due relazioni.

La prima è VIENE_CURATA_DA che fornisce l'informazione: una patologia, trovata per mezzo di una diagnosi, può essere curata con l'applicazione di una particolare terapia. Quest'informazione, nel modello relazionale, può essere trovata cercando, tra tutte le tuple per le terapie prescritte, una che abbia il campo *diagnosi* uguale al campo *cod_dia* della DIAGNOSI. Notare che non esiste nessuna certezza che tale tupla esista. Questa possibilità è consistente con quanto modellato nello schema E-R (la partecipazione di DIAGNOSI è parziale). Bisogna, invece, introdurre un vincolo che vieti che due tuple del tipo TERAPIA_PRESCRITTA referenzino la stessa diagnosi, in altre parole bisogna assicurare la proprietà di chiave al campo *diagnosi*.

La seconda relazione è VIENE_PRESCRITTA. Anche in questo caso il campo *Terapia* permette di trovare tutte le particolari applicazioni di una data terapia. In questo caso, poiché la relazione è del tipo Uno-A-Molti, la possibilità che lo stesso *Codice_Terapia* venga referenziato da tuple differenti non è un problema. Invece, deve essere introdotto un vincolo d'integrità per assicurare che ogni TERAPIA salvata sia stata applicata almeno una volta (partecipazione totale).

Ora che tutte le entità sono state tradotte si può procedere alla traduzione delle relazioni. Il risultato finale è rappresentato in Figura 12.

VIENE_RICOVERATO è del tipo Uno-A-Molti con entrambe le partecipazioni totali. Come visto in precedenza non c'è modo di assicurare la partecipazione se non con dei vincoli esterni, in questo caso da applicare su RICOVERO. La tupla che rappresenta i ricoveri deve essere modificata aggiungendo un campo per una chiave esterna chiamata *Paziente*.

Anche le relazioni EFFETTUATA_DURANTE, EFFETTUATA_A e SOMMINISTRATA devono essere affrontate allo stesso modo, tutte del tipo Uno-A-Molti. DIAGNOSI partecipa alle prime due con cardinalità pari a uno bisognerà quindi aggiungere due nuovi campi: il primo punterà alla chiave primaria di RICOVERO, mentre il secondo a quella di PAZIENTE. Poiché una diagnosi deve essere necessariamente collegata ad un ricovero, dunque anche ad un paziente, i campi appena introdotti non possono essere lasciati vuoti. A TERAPIA verrà aggiunta una chiave esterna contenente l'identificatore di FARMACO, anch'essa deve obbligatoriamente contenere un valore.

È_EFFETTO_COLLATERALE si differenzia dai casi precedenti non solo perché è del tipo Uno-A-Uno ma soprattutto perché non necessita di vincoli esterni, in quanto le due entità partecipano in modo parziale. Data la simmetria della relazione, si può scegliere se modificare la tupla relativa a DIAGNOSI o quella di TERAPIA_PRESCRITTA.

Si è preferito usare quest'ultima perché le terapie prescritte sono in numero minore e ciò significa che ci saranno meno campi vuoti nella base di dati.

L'ultima relazione da analizzare è CONTIENE, del tipo Molti-A-Molti. L'unico modo per trasformare questa relazione è creare una nuova tupla, che è stata chiamata CONTIENE ed è composta da tre valori: le chiavi esterne *Farmaco* e *Principio_Attivo* seguite da *Quantità*. La chiave primaria della tupla è formata dai primi due valori in modo che un qualsiasi FARMACO possa essere messo in relazione con tutti i principi attivi necessari e, viceversa, che un PRINCIPIO_ATTIVO possa essere contenuto in più farmaci. Notare che è obbligatoria l'introduzione di due vincoli d'integrità, uno deve assicurare che un PRINCIPIO_ATTIVO compaia in almeno un farmaco, l'altro che un farmaco contenga almeno un PRINCIPIO_ATTIVO.

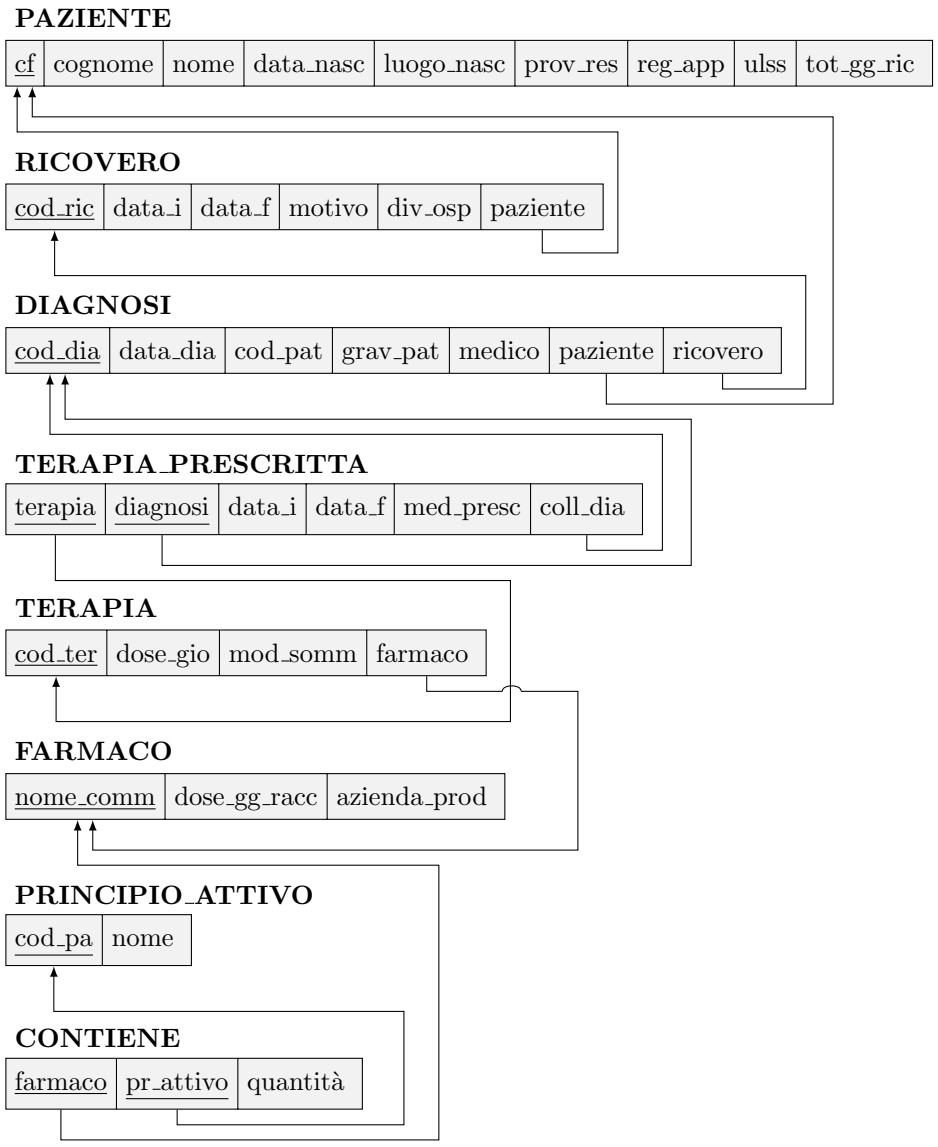

Figura 12: Modello relazionale finale

3 Definizione della Base di Dati in SQL

Il risultato della sezione precedente è lo schema Relazionale che verrà effettivamente usato per creare la base di dati. Per prima cosa dovrà essere definita una tabella per ogni tupla dello schema poi si procederà alla creazione di funzioni, constraint e trigger per assicurare tutti i vincoli d'integrità. Si suppone che ciascuno dei seguenti comandi SQL venga eseguito all'interno dello schema *ospedale*, cioè dopo l'esecuzione del seguente blocco:

```
create schema ospedale;
set search_path to ospedale;
```

3.1 Definizione delle Tabelle

Prima di iniziare a creare le tabelle bisogna definire alcuni domini:

```
create domain dom_cf as varchar
  check ( value ~ '^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}
                  [A-Z][0-9]{3}[A-Z]$' );

create sequence dom_ric_seq;
create domain dom_ric as int default
  nextval('dom_ric_seq');

create sequence dom_dia_seq;
create domain dom_dia as int default
  nextval('dom_dia_seq');

create sequence dom_ter_seq;
create domain dom_ter as int default
  nextval('dom_ter_seq');

create sequence dom_pa_seq;
create domain dom_pa as int default
  nextval('dom_pa_seq');

create domain ICD10 as varchar
    check ( value ~ '^[A-Z]([A-Z]|[0-9]){2}
                  ((\.)([0-9])*)?$' );
```

Il primo e l'ultimo rispettano le normali convenzioni mentre gli altri sono rappresenti da delle sequenze crescenti distinte di interi.

Si può ora procedere alla creazione delle tabelle, sotto ne sono riportate tre a titolo d'esempio:

```
create table paziente (
    cf dom_cf primary key,
    cognome varchar not null,
    nome varchar not null,
    data_nasc date not null,
    luogo_nasc varchar not null,
    prov_res varchar(2) not null,
    reg_app varchar,
    ulss varchar,
    tot_gg_ric int default 0
);

create table ricovero (
    cod_ric dom_ric primary key,
    data_i date not null,
    data_f date,
    motivo varchar not null,
    div_osp varchar not null,
    paziente dom_cf not null references
                    paziente(cf)
                    on update cascade
                    on delete cascade
);

create table terapia_prescritta (
    data_i date not null,
    data_f date not null,
    med_presc varchar not null,
    diagnosi dom_dia unique references
                    diagnosi(cod_dia)
                    on update cascade
                    on delete no action,
    terapia dom_ter references
                    terapia(cod_ter)
                    on update cascade
                    on delete no action,
    coll_dia dom_dia unique references
                    diagnosi(cod_dia)
                    on update cascade
                    on delete no action,
    primary key (diagnosi, terapia)
);
```

Questa tripletta è stata scelta in quanto illustra ogni possibile casistica:

- una tabella semplice, con un solo attributo come chiave ed un campo con valore di default;
- una tabella leggermente più complessa che presenta anche un campo chiave esterna;
- ed infine una tabella che rappresenta un'entità debole, con una coppia di valori come chiave primaria;

Le rimanenti tabelle sono di complessità minore od uguale a quelle mostrate.

A questo punto si può procedere con la creazione dei constraint. Questi possono essere sfruttati per assicurare i vincoli tra le varie date di inizio e fine presenti nella base di dati, nello specifico:

- in generale una tabella con dei campi che indicano una data d'inizio ed una di fine deve rispettare la condizione $data\_i \leq data\_f$;
- un RICOVERO non può avere *data_i* minore di *data_nasc* del suo PAZIENTE;
- una DIAGNOSI deve avere *data_dia* compresa tra *data_i* e *data_f* del RICOVERO a cui appartiene;
- una TERAPIA_PRESCRITTA non può avere una *data_i* anteriore a quella della DIAGNOSI in cui è stata prescritta;

Il primo punto può essere facilmente soddisfatto dal seguente blocco, notare che la parte finale del codice può essere applicata a tutte le tabelle con due campi del tipo *date*.

```
create or replace function
check_date_valide(data_i date, data_f date)
returns bool as
$$
  begin
    return data_i < data_f;
  end;
$$ language plpgsql;

alter table ricovero add
  constraint check_date_valide_ric
    check(check_date_valide(data_i, data_f));
```

Il terzo punto richiede una funzione più articolata: a partire dal codice identificativo di un RICOVERO si devono ottenere le sue date, per poi verificare la condizione, ricordando che alcuni ricoveri non presentato una data di fine.

```
create or replace function
check_data_valida(data_dia date, ric dom_ric)
returns bool as
$$
  begin
    perform *
    from ricovero
    where ric = cod_ric and
          data_i <= data_dia  and
          (data_f >= data_dia or
            data_f is null);
    return found;
  end;
$$ language plpgsql;

alter table diagnosi add
  constraint check_data_valida_dia
    check(check_data_valida(data_dia, ricovero));
```

Non verrà riportati i blocchi per la seconda e per la terza condizione perché sono semplicemente una rivisitazione del codice appena illustrato, ad esempio: ottenuta *data_dia* si verifica che sia minore o uguale alla data della TERAPIA_PRESCRITTA.

Di seguito sono riportati i vincoli di integrità ancora da implementare:

1. ogni paziente deve essere stato ricoverato almeno una volta;
2. tutte le diagnosi di un dato ricovero devono avere il campo *paziente* uguale a quello del ricovero;
3. ad ogni diagnosi può corrispondere al più una terapia prescritta;
4. ogni diagnosi può essere effetto collaterale di al più una terapia_prescritta;
5. ogni terapia deve essere stata prescritta almeno una volta;
6. ogni farmaco deve contenere almeno un principio attivo;
7. ogni principio attivo deve essere contenuto in almeno un farmaco;

I vincoli 3 e 4 possono essere assicurati impostando ad unique i campi *diagnosi* e *coll_dia* in TERAPIA_PRESCRITTA.

Il vincolo 2 è facile da assicurare. Infatti basta sfruttare una funzione (in questo caso *get_paziente*) che, a partire dal codice di un ricovero, torni il paziente di quel ricovero, tale output sarà il valore del campo *paziente* della diagnosi. Sono stati creati due trigger: il primo scatta alla creazione di una nuova diagnosi,

mentre il secondo assicura che *paziente* rimanga coerente anche dopo l'aggiornamento del campo *ricovero*. Il secondo trigger protegge anche da eventuali aggiornamenti errati del paziente, impostando sempre il valore corretto.

```
create or replace function init_paziente()
returns trigger as
$$
  begin
    new.paziente := get_paziente(new.ricovero);
    return new;
  end;
$$ language plpgsql;

create trigger init_paziente_dia before insert
  on diagnosi for each row
  execute procedure init_paziente();

create trigger update_paziente_dia before update
  on diagnosi
  for each row
  when ( new.paziente <> old.paziente or
         new.ricovero <> old.ricovero )
  execute procedure init_paziente();
```

Il vincolo 1 non può essere modellato né come constraint né come trigger perché introdurrebbe una dipendenza ciclica con la condizione *not null* del campo *paziente* in RICOVERO, in quanto non sarebbe accettato un paziente senza un ricovero, ma allo stesso tempo un ricovero non potrebbe essere inserito nella base di dati poiché il suo campo *paziente* sarebbe necessariamente *null*, non esistendo ancora il paziente. Il constraint *not null* in RICOVERO è necessario perché deve essere sempre possibile risalire alle informazioni del paziente di un dato ricovero. Diversamente la presenza nella base di dati di un PAZIENTE che non è mai stato ricoverato, situazione comunque anomala, non ha effetti disastrosi: in qualsiasi momento si può rimuovere tale paziente oppure gli si può attribuire un ricovero. Uno stesso ragionamento può essere fatto anche per 5. Ricordando che una TERAPIA_PRESCRITTA, essendo un'entità debole, deve necessariamente essere inserita dopo la TERAPIA da cui dipende.

Si può notare che anche i vincoli 6 e 7 creano una dipendenza ciclica. A livello intuitivo sarà possibile implementarne soltanto uno. Similmente a quanto detto prima è più grave che, dato una farmaco, non si sappiano quali principi attivi lo compongano, piuttosto che aver memorizzato principi attivi che non compaiono in alcun farmaco. Nel blocco sottostante viene illustrata questa soluzione:

```
create or replace function almeno_un_pa_cont()
returns trigger as
$$
```

```
  begin
    perform *
    from contiene
    where old.farmaco == farmaco and
          old.pr_attivo <> pr_attivo;
    if not found
    then
      raise exception
        'Rimarrebbe senza principi attivi';
      return null;
    else
      return new;
    end if;
  end;
$$ language plpgsql;

create trigger controlla_almeno_pa_del before delete
  on contiene for each row
    execute procedure almeno_un_pa_cont();

create trigger controlla_almeno_pa_up before update
  on contiene for each row
    when ( new.farmaco <> old.farmaco or
           new.pr_attivo <> old.pr_attivo)
    execute procedure almeno_un_pa_cont();
```

Non c'è modo, invece, di assicurare che all'inserimento un farmaco non rimanga senza principi attivi; spetterà all'utente inizializzare correttamente i dati. Il trigger assicura che ad un farmaco non vengano mai rimossi tutti i principi attivi.

Osservando lo schema in Figura 10 si può notare la presenza di un attributo derivato in PAZIENTE. SQL non offre un costrutto per la definizione di tali attributi, è necessario creare dei triggers:

```
create or replace function aggiorna_gg_update()
returns trigger as
$$
  begin
    update paziente
    set tot_gg_ric = ricalcolo_gg(new.paziente)
    where paziente.cf = new.paziente;

    if new.paziente <> old.paziente
      then
      update paziente
```

```
      set tot_gg_ric = ricalcolo_gg(old.paziente)
      where paziente.cf = old.paziente;
    end if;
    return new;
  end;
$$ language plpgsql;

create trigger aggiornamento_gg_insert after insert
  on ricovero for each row
  execute procedure aggiorna_gg_insert();

create trigger aggiornamento_gg_update after update
  on ricovero for each row
    when ( new.data_i <> old.data_i or
           new.data_f <> old.data_f or
           old.data_i is null       or
           old.data_f is null       or
           new.paziente <> old.paziente )
      execute procedure aggiorna_gg_update();
```

La funzione *ricalcolo_gg* calcola la somma totale dei giorni che un paziente ha passato da ricoverato.

3.2 Definizione di Query Significative

Segue un elenco di query significative, ciascuna con un breve commento.

3.2.1 Query 1

Per ogni provincia il paziente con il maggior numero di diagnosi.

```
CREATE OR REPLACE VIEW COUNT_DIA AS
SELECT cf, nome, cognome, prov_res,
       count(cod_dia) AS num_dia
FROM PAZIENTE
  JOIN DIAGNOSI ON paziente = cf
GROUP BY cf;

SELECT *
FROM COUNT_DIA cd1
WHERE
  NOT EXISTS
  (SELECT *
   FROM COUNT_DIA cd2
   WHERE cd2.prov_res = cd1.prov_res
     AND cd2.num_dia > cd1.num_dia);
```

La creazione della vista risulta molto utile per due motivi: la query è più leggibile; il JOIN ed il COUNT vengono eseguiti una sola volta, quindi il risultato sarà fornito prima. Va fatto notare che il numero di righe nella tabella risultato può essere maggiore del numero di province presenti nel database. Infatti, considerando solo i pazienti di una data provincia, è possibile che una coppia di pazienti abbia lo stesso numero di diagnosi e che non esista nessun paziente che ne abbia in maggior numero: è corretto che entrambi i pazienti compaiano.

3.2.2 Query 2

Si vogliono trovare tutti i pazienti che sono stati ricoverati esattamente due volte.

```
SELECT p.cf, p.nome, p.cognome
FROM PAZIENTE p
WHERE
  EXISTS
  (SELECT *
   FROM RICOVERO r1
   JOIN RICOVERO r2 ON
     r1.cod_ric < r2.cod_ric AND
     r1.paziente = r2.paziente
   WHERE r1.paziente = p.cf
     AND
     NOT EXISTS
     (SELECT *
      FROM RICOVERO r3
      WHERE r3.paziente = r1.paziente AND
            r3.cod_ric <> r1.cod_ric AND
            r3.cod_ric <> r2.cod_ric));
```

I due EXISTS annidiati permettono, prima di rimuovere tutti i pazienti che non sono stati ricoverati almeno due volte, poi di considerare soltanto quelli per cui non esiste un terzo ricovero.

3.2.3 Query 3

Si vuole trovare il mese nel quale sono iniziati il maggior numero di ricoveri.

```
CREATE OR REPLACE VIEW MONTH_COUNT AS
SELECT month, COUNT(*) AS c
FROM (SELECT cod_ric,
             date_part('month', data_i) AS month
      FROM RICOVERO) AS ric_month
GROUP BY month;

SELECT month, c
FROM MONTH_COUNT mc
WHERE
  NOT EXISTS
  (SELECT *
   FROM MONTH_COUNT
   WHERE mc.c < c);
```

Come si è visto nelle query precedenti, la vista permette di non ricostruire tabelle durante l'esecuzione della query, favorendo la rapida esecuzione della stessa.

3.2.4 Query 4

Si vogliono ottenere tutti i farmaci che hanno causato effetti collaterali, ordinati in ordine decrescente per numero di effetti collaterali.

```
SELECT farmaco, tc.n
FROM
  (SELECT terapia, COUNT(*) AS n
   FROM TERAPIA_PRESCRITTA
   WHERE coll_dia IS NOT NULL
   GROUP BY terapia) AS tc
  JOIN TERAPIA ON cod_ter = tc.terapia
ORDER BY tc.n DESC;
```

Il JOIN è stato usato perché l'informazione riguardo gli effetti collaterali risiede nella tabella TERAPIA_PRESCRITTA mentre quella riguardo al farmaco nella tabella TERAPIA.

3.2.5 Query 5

Si vogliono ottenere i pazienti che sono stati curati almeno due volte consecutivamente con lo stesso farmaco.

```
CREATE OR REPLACE VIEW PAZ_TER AS
SELECT cf, nome, cognome, cod_ter, data_i, farmaco
FROM PAZIENTE
  JOIN DIAGNOSI ON paziente = cf
  JOIN TERAPIA_PRESCRITTA ON coll_dia = cod_dia
  JOIN TERAPIA ON cod_ter = terapia;

SELECT DISTINCT pt1.cf, pt1.nome,
                pt1.cognome, pt1.farmaco
FROM PAZ_TER pt1
WHERE
  EXISTS
  (SELECT *
   FROM PAZ_TER pt2
   WHERE pt2.cf = pt1.cf
     AND pt2.cod_ter = pt1.cod_ter
     AND pt2.farmaco = pt1.farmaco
     AND
     NOT EXISTS
     (SELECT *
      FROM PAZ_TER pt3
      WHERE pt3.cf = pt1.cf
        AND pt1.data_i < pt3.data_i
        AND pt3.data_i < pt2.data_i
        AND pt3.farmaco <> pt1.farmaco
        ));
```

Lo EXISTS permette di verficiare che il paziente abbia assunto almeno due volte lo stesso farmaco. Con il NOT EXISTS vengono rimossi tutti quei pazienti che, durante il periodo tra le due assunzioni del farmaco sopra, hanno assunto un altro farmaco.

3.2.6 Query 6

Si vogliono ottenere tutte le coppie di pazienti in cui il primo paziente ha contratto le stesse (tutte e sole) malattie del secondo.

```
SELECT p1.cf, p2.cf
FROM PAZIENTE p1
  JOIN PAZIENTE p2 ON p1.cf < p2.cf
WHERE
  NOT EXISTS
  (SELECT *
   FROM DIAGNOSI pat2
   WHERE pat2.paziente = p2.cf
     AND
     NOT EXISTS
     (SELECT *
      FROM DIAGNOSI pat1
      WHERE pat1.paziente = p1.cf
        AND pat1.cod_pat = pat2.cod_pat))
  AND
  NOT EXISTS
  (SELECT *
   FROM DIAGNOSI pat1
   WHERE pat1.paziente = p1.cf
     AND
     NOT EXISTS
     (SELECT *
      FROM DIAGNOSI pat2
      WHERE pat2.paziente = p2.cf
        AND pat2.cod_pat = pat1.cod_pat));
```

La prima parte della query verifica che il paziente p1 abbia contratto almeno tutte le malattie del paziente p2. Viene sfruttata l'implicazione logica: se p2 ha contratto la malattia allora anche p1 deve averla contratta In altre parole non è possibile che p1 non abbia contratto una malattia contratta da p2. Nella seconda parte si verifica che p1 non abbia mai contratto malattie che p2 non ha mai contratto, cioè: se p2 non ha mai contratto la malattia allora anche p1 non l'ha mai contratta. Che può essere tradotto in: osservando l'elenco di malattia di p1 non ne compare nessuna che non sia stata contratta anche da p2.

4 Progettazione Fisica

4.1 Nuovi Indici

La creazione di strutture ausiliarie, quali gli indici, permette di migliorare le performance di un database. In particolare, gli indici vengono sfruttati per rendere più efficiente la ricerca di record in una determinata tabella. Per le chiavi primarie o attributi che devono soddisfare i vincoli di unicità viene creato automaticamente un indice dal gestore del database, di conseguenza la creazione manuale di indici fa riferimento, generalmente, solo a colonne che non rispettano queste condizioni. Non sempre un indice risulta utile ai fini del miglioramento delle performance, infatti, se esso viene creato su una colonna sulla quale si effettuano frequenti operazioni di aggiornamento, allora anche l'indice deve essere manipolato frequentemente per rispettare le modifiche. Viceversa, se il database viene utilizzato staticamente, solo per scopi di report, si possono sfruttare anche più indici. Un indice ottimale possiede alcune caratteristiche peculiari:

- Non è troppo lungo in quanto, in caso contrario, questo comportebbe un maggior numero di operazioni di lettura sul disco.
- I record dell'indice vengono spesso utilizzati nei confronti quindi entry più corte migliorano l'efficenza. Ad esempio, una colonna di interi è la scelta migliore per un indice mentre il confronto tra stringhe implica l'esecuzione del test di uguaglianza carattere per carattere.
- Ha pochi valori duplicati in quanto avere tante entry ripetute vanifica il lavoro dell'indice, perché nell'esecuzione di una ricerca bisogna procedere in modo sequenziale sui valori ripetuti.

Gli indici possono essere costruiti basandosi su strutture dati differenti a seconda di come verranno sfruttati nelle query. Ad esempio, se una colonna viene utilizzata spesso in test di uguaglianza, allora una struttura con tabella di hash risulta più efficiente di un btree che invece può essere utilizzato per ottimizzare i confronti generici. Inoltre, la caratteristica intrinseca degli indici di ordinare i record consente di ottimizzare quelle query in cui si richiede, tramite l'istruzione ORDER BY, di ordinare il risultato secondo una determinata colonna. Con un indice, in tempo di esecuzione, non dovrebbe esser svolto alcun lavoro, infatti i dati risiedono in memoria già ordinati.

Per la valutazione sull'inserimento o meno di alcuni indici nel caso specifico del progetto è stata utilizzata l'istruzione EXPLAIN ANALYZE che, invece di restituire il risultato della query, gestisce il piano dettagliato con cui il database procede nell'esecuzione dei comandi dell'interrogazione. In particolare consente di analizzare i costi, forniti tramite un valore numerico, di una specifica sequenza di operazioni nonché il tempo impiegato nella stessa. È interessante notare come

nell'output dell'analisi si faccia riferimento al tempo di pianificazione della query oltre al tempo della sua esecuzione.

Nel blocco di codice SQL sottostante viene mostrato il comando per creare un indice, seguito da una delle query su cui sono stati effettuati i calcoli sui tempi.

```
CREATE INDEX nome_idx ON paziente (nome,cognome);

EXPLAIN ANALYZE
SELECT cf, data_nasc,
       cod_ric, data_i, data_f,
       motivo, div_osp
FROM paziente
JOIN ricovero ON paziente = cf
WHERE nome = 'tino'AND cognome = 'labella';
```

Si è considerato il fatto che spesso si utilizzano il nome e il cognome del paziente per ricavare alcune informazioni dal database senza l'inserimento del codice fiscale. La query sopra, che fornisce i dati dei ricoveri di tutti i pazienti con nome e cognome inseriti, riesce a sfruttare l'indice creato ottenendo così un notevole miglioramento nei tempi di risposta. È stato interessante notare come il tempo di pianificazione aumenta in presenza dell'indice mentre le effettive operazioni risultano più efficienti e veloci in termini di costo e tempo.

Bisogna però verificare che l'aggiornamento dell'indice non causi rallentamenti in fase di inserimento[1] di un nuovo paziente. Osservando i tempi ed i costi di alcuni inserimenti prima e dopo la creazione dell'indice ci si accorge che questo non influenza negativamente i tempi di risposta delle query.

Sono state considerate anche altre query, ad esempio la query nel blocco sottostante, che permette di ottenere tutte le coppie di pazienti con lo stesso nome e cognome. Queste però non vengono influenzate, né positivamente né negativamente, dall'indice. Si suppone sia dovuto alla creazione di ulteriori strutture ausiliare in supporto alle operazioni di JOIN.

```
explain analyze select p1.nome, p1.cognome, count(*)
from paziente p1
join paziente p2 on p1.cf < p2.cf and
              p1.nome = p2.nome and
              p1.cognome = p2.cognome
group by p1.nome, p1.cognome;
```

[1] i casi DELETE ed UPDATE non sono stati citati in quanto si suppone che i pazienti non vengano eliminati dalla base di dati e che vengano aggiornati raramente. È stato comunque verificato che il loro comportamento non cambia significativamente in presenza dell'indice.

5 Analisi Dati

5.1 Popolamento della Base di Dati

La base di dati è stata testata su un insieme di dati generato tramite uno script R. Nello specifico lo script genera, ispirandosi ai volumi indicati nei requisiti, dei dati verosimili, ipotizzando che l'ospedale sia attivo da circa un decennio:

- sono presenti diecimila pazienti in totale,
- ciascun paziente in media è stato ricoverato tre volte,
- per ogni paziente esistono in media dodici diagnosi, cioè circa quattro diagnosi per ricovero,
- in totale sono presenti mille farmaci e ciascuno contiene un insieme di principi attivi scelti da un totale di novecento possibili,
- soltanto il dieci percento delle diagnosi non ha una cura,
- il cinque percento delle terapie prescritte causa un effetto collaterale;

Segue una breve descrizione della struttura dello script:

1. i pazienti vengono generati combinando nomi, cognomi, province di residenza, luogo di nascita e ULSS, ciascuno prelevato casualmente dal rispettivo file mentre codice fiscale e la data di nascita vengono creati attraverso funzioni apposite;
2. per ogni paziente vengono creati alcuni ricoveri ed alcune diagnosi, assicurando che le date di quest'ultime non precedano mai quelle d'inizio dei ricoveri;
3. principi attivi e farmaci vengono caricati dai rispettivi file ed associati tra loro;
4. in modo simile vengono create anche le terapie;
5. durante la creazione delle terapie prescritte si presta attenzione alla data del ricovero;
6. una diagnosi come effetto collaterale viene aggiunta ad una determinata quantità di terapie prescritte;

Ora che tutti i dati sono stati generati si può procedere con il loro inserimento nella base di dati. Nel blocco sottostante sono riportati alcuni dei comandi principali:

```
require("RPostgreSQL")

drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv,
                 dbname="bd_18_paolo_addis",
                 host="158.110.145.186",
                 port="5432",
                 user="bd_18_paolo_addis",
                 password="******")

dbWriteTable(con, name="paziente",
             value=pazienti_df, row.names=FALSE,
             append=TRUE)
dbWriteTable(con, name="diagnosi",
             value=diagnosi_df, row.names=FALSE,
             append=TRUE)

res <- dbGetQuery(con,
                  "set search_path to ospedale;
                   select nome, data_nasc, tot_gg_ric
                   from paziente; ")

dbDisconnect(con)
```

Nella prima parte viene creata la connessione con il database. Poco sotto viene mostrato come sono stati inseriti oggetti del tipo DataFrame nella base di dati. Infine, prima di chiudere la connessione, viene effettuata una semplice query.

5.2 Analisi Dati in R

Seguono una serie di grafici con breve descrizione.

Richiesta: Si vuole osservare come siano cambiate, col passare degli anni, la quantità di nuovi pazienti inseriti mensilmente rispetto al numero di nuovi ricoveri registrati mensilmente.

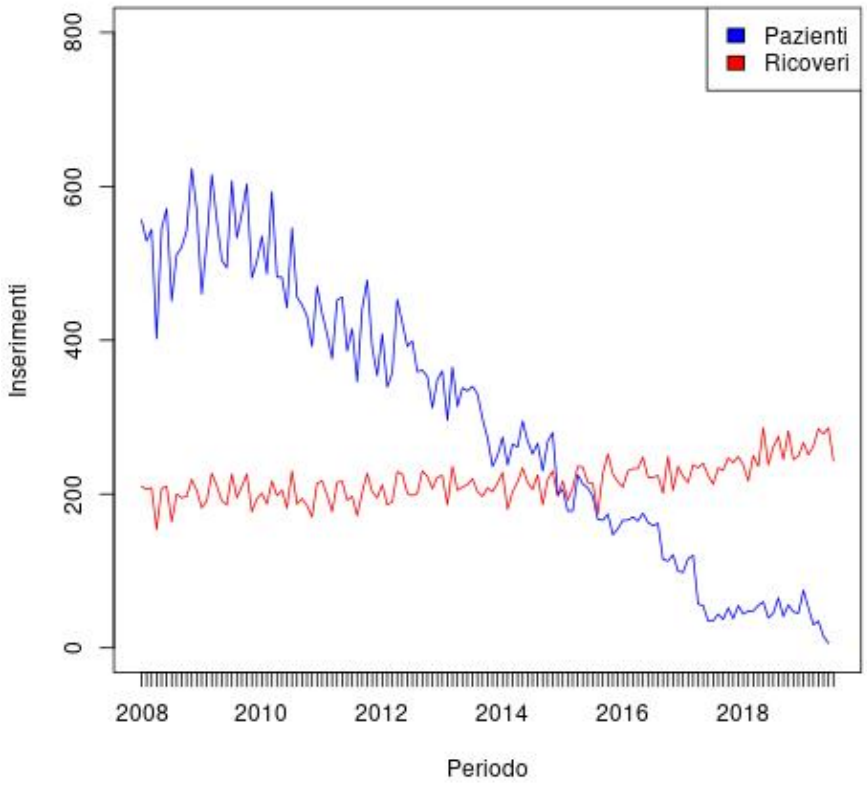

Figura 13: Andamento mensile inserimenti di Pazienti e di Ricoveri

Dal grafico si può intuire che: il numero di ricoveri inseriti mensilmente è soggetto ad una lenta crescita; il numero di nuove registrazioni, invece, sta calando notevolmente. Mettendo assieme le due informazioni si conclude che, in generale, un paziente che è già stato ricoverato almeno una volta, ha una probabilità maggiore, rispetto a qualche anno fa, di essere ricoverato nuovamente.

Richiesta: Annualmente quanti ricoveri riguardano un anziano (sopra i 60 anni)? Quanti riguardano un giovane (sotto i 30)?

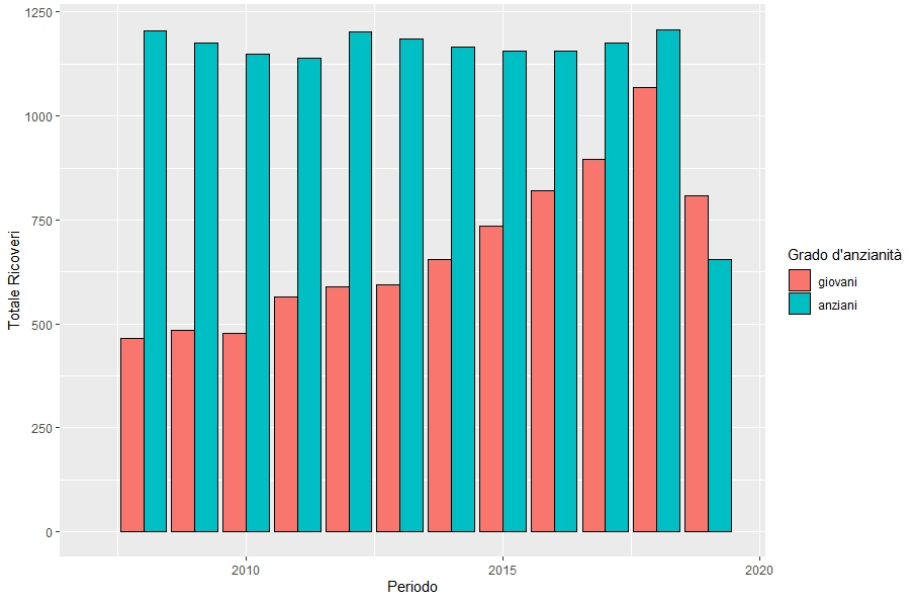

Figura 14: Ricoveri medi annui pazienti anziani vs giovani

Dal grafico si può notare che il numero di ricoveri di pazienti giovani è cresciuto notevolmente, mentre quello degli anziani è rimasto pressoché invariato. Nel 2008 il numero di anziani ricoverati era oltre il doppio del numero di giovani, invece nel 2018 il numero di giovani, seppur di poco, ha addirittura superato il numero di anziani. Questa tendenza potrebbe indicare che i giovani sono più cagionevoli di un tempo. Con il prossimo grafico si vuole analizzare se la tendenza di cui sopra sia in qualche modo correlata alla crescente probabilità di sviluppare tumori in giovane età.

Richiesta: Annualmente a quanti giovani viene diagnosticato un tumore? A quanti anziani? A quanti nella fascia tra i 30 ed i 60 anni?

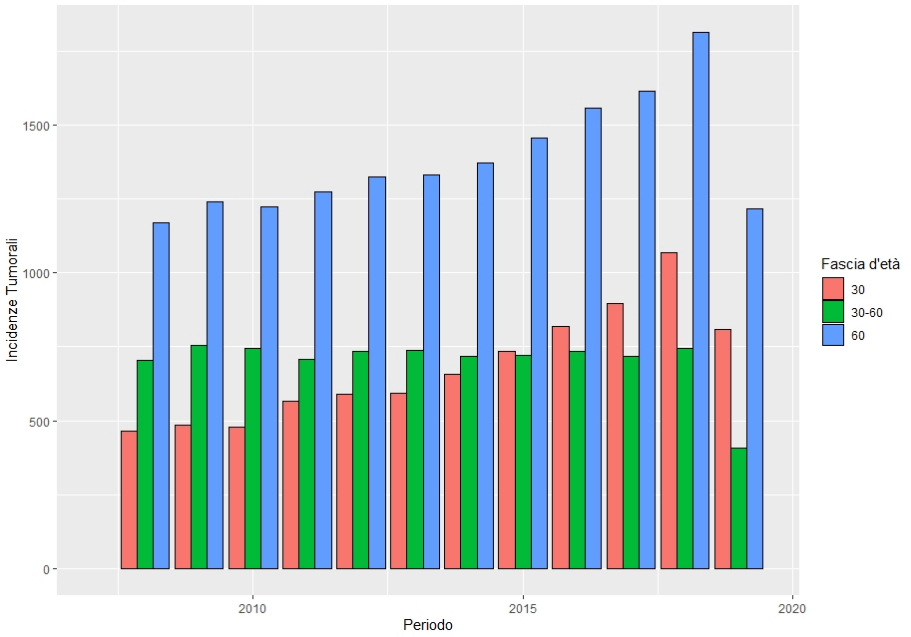

Figura 15: Ricoveri per tumore annui al variare delle età

Il grafico mostra chiaramente come i ricoveri per tumore sono sempre più frequenti per i giovani. Anche la fascia d'età sopra i 60 segue un andamento simile. Invece la fascia intermedia sembra avere una tendenza calante.

Richiesta: In quale periodo dell'anno ci sono più ricoveri per influenza?

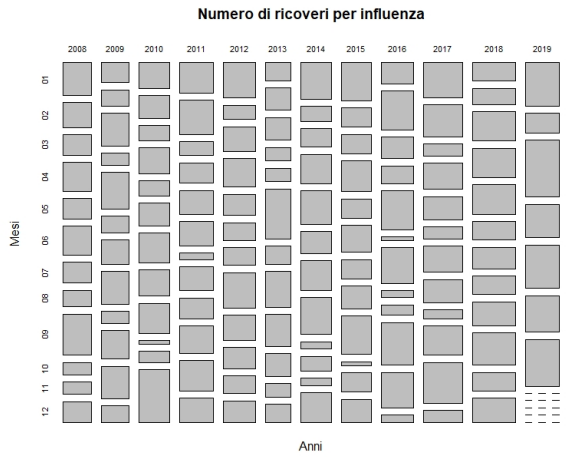

Figura 16: Ricoveri per influenza mensili al variare degli anni

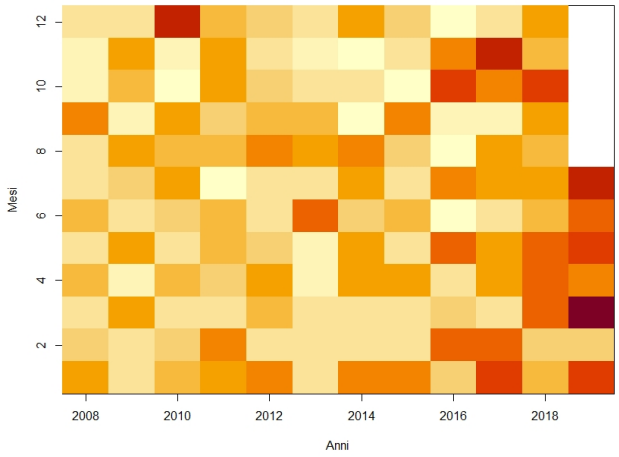

Figura 17: Ricoveri per influenza mensili al variare degli anni

I due grafici rappresentano la stessa informazione. Dal grafico a mosaico risulta più facile osservare l'andamento mensile di uno specifico anno, mentre il grafico *heatmap* facilita il confronto di un mese al variare degli anni. Ad esempio nel mese di dicembre del 2010 ci sono stati molti più ricoveri per influenza rispetto agli anni tra il 2011 ed il 2018.

Richiesta: Rispetto alle durate dei ricoveri in cui è stata diagnosticata una o più patologie tra le più frequenti, come si collocano le durate dei ricoveri legati a diagnosi per tumore?

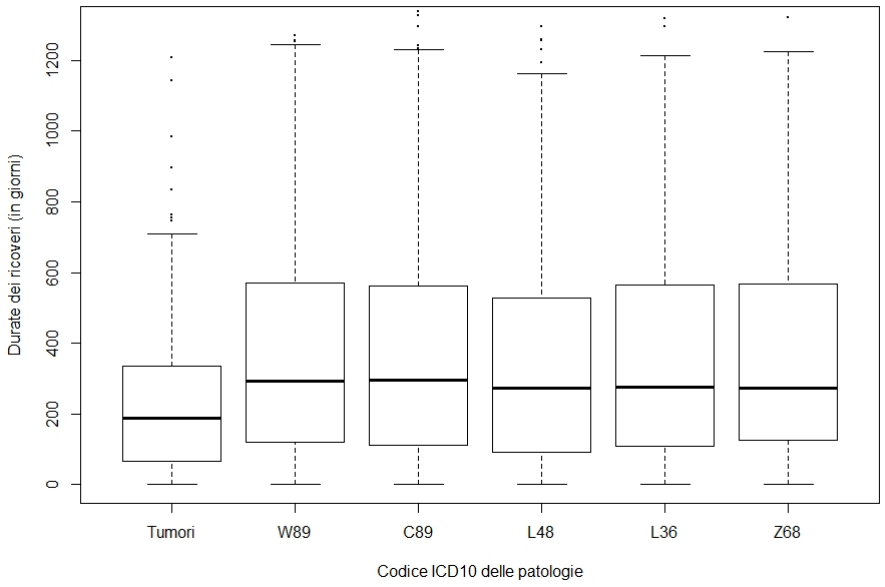

Figura 18: Confronto durate ricoveri per incidenze tumorali rispetto a quelli per patologie più frequenti

Il grafico mostra come la tendenza delle durate dei ricoveri per incidenze tumorali sia leggermente inferiore rispetto a quella dei ricoveri legati alle 5 patologie più frequenti. La distribuzione è asimmetrica in tutti i boxplot con una evidente variazione della distanza interquartilica rispetto alla mediana. Grazie a questa asimmetria è possibile notare come la durata dei ricoveri legati ai tumori sia leggermente meno variabile rispetto alle altre patologie considerate. In particolare, se la durata del ricovero supera i 200 giorni, potrebbe, statisticamente, proseguire per svariati giorni, in quanto un intervallo interquartilico maggiore implica una distribuzione più ampia. È interessante notare che alcuni valori anomali (outlier) indicano che per tutte le patologie considerate vi sono stati dei ricoveri che hanno avuto una durata di quasi 4 anni.