




APRILE 2017

BASI DI DATI 2

PROGETTAZIONE ED IMPLEMENTAZIONE DI UN DATABASE E DI UNA
WEBAPP PER INTERAGIRVI

GIUSEPPE LORUSSO

UNIVERSITÀ DEGLI STUDI DI BASI "ALDO MODO"
Dipartimento di Informatica



Analisi delle specifiche dei requisiti

Scegliere il giusto livello di astrazione

- Ciascuno prodotto, sia i farmaci sia gli articoli di profumeria, è identificato da un ID univoco (rigo 2).
- Per *lista di medicine prescrivibili* si intende un elenco degli ID dei farmaci prescrivibili (rigo 10-11).

Linearizzare le frasi e scomporre quelle articolate

“Ogni farmaco è individuato univocamente dal proprio nome e dall’informazione relativa alla casa farmaceutica che lo produce, identificata univocamente da nome e recapito” (rigo 5 – 6).

Diventa

“Ogni farmaco è individuato da un ID, ed è associata inoltre alla casa farmaceutica che lo produce.”

“Ogni casa farmaceutica è identificata univocamente da nome e recapito.”

“La farmacia vuole memorizzare le prescrizioni ricevute. Una prescrizione contiene la lista degli id dei farmaci prescrivibili, è effettuata da un medico ed è intestata ad un paziente.”

Diventa

“La farmacia vuole memorizzare le prescrizioni ricevute, dove ogni prescrizione contiene la lista degli id dei farmaci prescritti”

“Ad effettuare una prescrizione è un medico”

“Ogni prescrizione deve avere un paziente intestatario”

Individuare omonimi e sinonimi

Farmaci brevettati = farmaci di marca

Medicine prescrivibili = farmaci prescrivibili

Glossario dei termini

TERMINE	DESCRIZIONE	COLLEGAMENTO
Paziente	Gli individui ai quali un medico intesta una prescrizione	Prescrizione
Prescrizione	Documento che consente ad un paziente di acquistare uno o più prodotti prescrivibili	Paziente, Medico, Prodotto
Medico	Specialista con l’autorità di prescrivere dei farmaci ai pazienti	Prescrizione
Prodotto	Tutti i farmaci e gli articoli di profumeria venduti dalla farmacia	Prodotto, Vendita, Prescrizione

Vendita	Si riferisce ad una transazione battuta alla cassa da un funzionario della farmacia	Prodotto
---------	---	----------

Riorganizzazione delle frasi

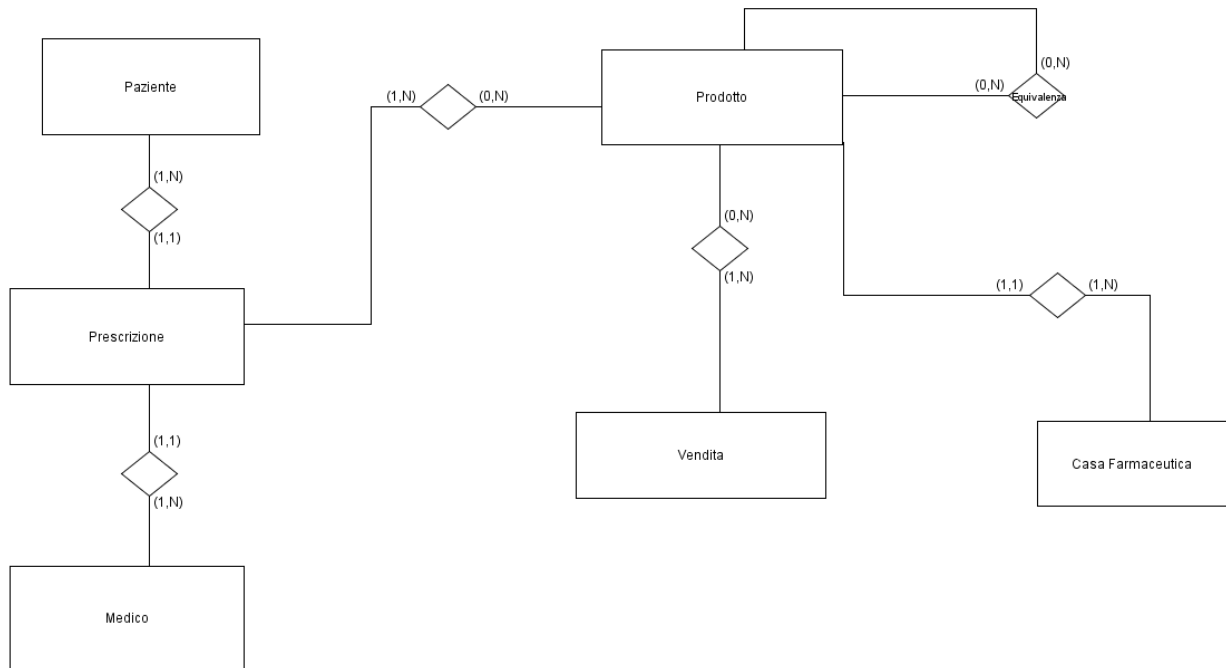
Generali	<ul style="list-style-type: none"> • Si vuole rappresentare una base di dati per la gestione dei prodotti disponibili in una farmacia tenendo conto delle seguenti informazioni • Inoltre, la farmacia è interessata a svolgere indagini statistiche sulle vendite e intende memorizzare i report, semestrali e annuali, delle vendite realizzate (IMPORTANTE: questa frase verrà utilizzata esclusivamente per la realizzazione del datawarehouse, e sarà quindi ignorata nella fase di progettazione del db)
Prodotto	<ul style="list-style-type: none"> • Ciascun prodotto è caratterizzato da un nome e una descrizione. I prodotti presenti nella farmacia possono essere farmaci oppure prodotti di profumeria. I prodotti di profumeria sono di diverso tipo: cosmetici, prodotti per l'igiene e prodotti per la cura del bambino. • Ciascuno prodotto, sia i farmaci sia gli articoli di profumeria, è identificato da un ID univoco, e dall'informazione relativa alla casa farmaceutica che lo produce. • I farmaci possono essere prescrivibili o non prescrivibili. I farmaci possono essere coperti da brevetto per un certo numero di anni; in tal caso si parla di farmaci brevettati o di marca. Inoltre la farmacia vende farmaci generici. Un farmaco generico è un farmaco senza marca equivalente a un farmaco brevettato. Ogni farmaco brevettato può avere uno o più farmaci generici equivalenti.
Casa farmaceutica	<ul style="list-style-type: none"> • Ogni casa farmaceutica è identificata univocamente da nome e recapito.
Prescrizione	<ul style="list-style-type: none"> • La farmacia vuole memorizzare le prescrizioni ricevute, dove ogni prescrizione contiene la lista degli id dei farmaci prescritti
Vendita	<ul style="list-style-type: none"> • Si vuole tenere traccia della vendita dei prodotti della farmacia, con l'indicazione sulla quantità e il giorno della vendita e le eventuali prescrizioni associate.
Paziente	<ul style="list-style-type: none"> • Ogni prescrizione deve avere un paziente intestatario

Medico

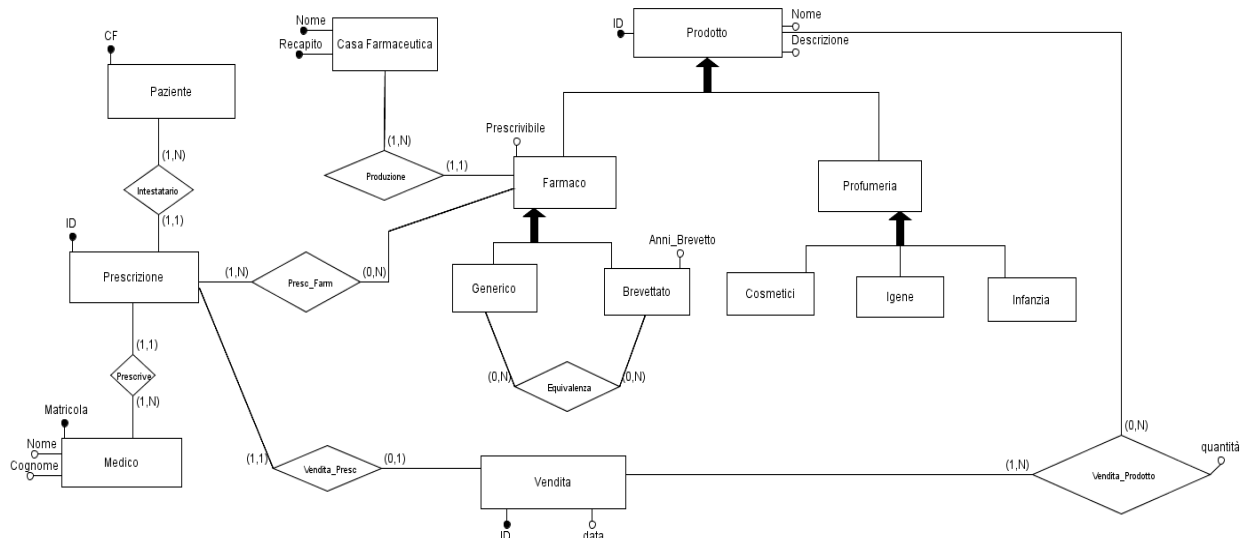
- Ad effettuare una prescrizione è un medico

Rappresentazione delle specifiche – SCHEMA SCHELETRO

Seguendo un approccio IBRIDO e seguendo l'analisi delle specifiche condotta finora, rappresentiamo le entità e le relazioni fra esse tramite uno "schema scheletro".



In seguito ad una fase di raffinamento iterata più volte, il diagramma concettuale è il seguente:



(per una consultazione più comoda: <https://www.dropbox.com/s/z6zusctexwh1oc8/concettuale.png?dl=0>)

PROGETTAZIONE LOGICA

OPERAZIONI

- **OP1** → Inserimento di una nuova vendita (100 volte al giorno)
- **OP2** → Ritrovamento di tutti i farmaci generici equivalenti ad un dato farmaco brevettato (10 volte al giorno)
- **OP3** → Stampa i medici che hanno prescritto un dato farmaco (10 volte al giorno)
- **OP4** → Inserimento di una prescrizione (10 volte al giorno)
- **OP5** → Inserimento di un nuovo prodotto (10 volte al giorno)
- **OP6** → Stampa le vendite di tutti i prodotti di tipo “farmaco brevettato” (1 volta al giorno)

Si suppone che:

- Il sistema sia in funzione da 5 anni, in un anno ci sono 300 giorni lavorativi, e che quindi il sistema funziona da 1500 giorni
- 2/3 dei prodotti sono farmaci e 1/3 sono articoli di profumeria (mi aspetto che i farmaci siano il doppio dei profumi)
- Solo il 20% dei farmaci sono brevettati
- Per acquistare un farmaco brevettato è necessaria una prescrizione medica (mi aspetto che le prescrizioni siano meno delle vendite)
- Mediamente il 10% delle vendite riguardano farmaci prescrivibili, e quindi hanno bisogno di prescrizione medica (mi aspetto che le prescrizioni siano il 10% delle vendite, stessa proporzione esiste fra vendita_presc e vendita)
- Mediamente una casa farmaceutica produce 100 farmaci
- È stato inserito un medico al giorno finora, quindi abbiamo 1500 medici
- Ogni medico gestisce in media 200 pazienti
- Mediamente ogni articolo brevettato ha 2 articoli generici equivalenti (mi aspetto che in equivalenza ci siano il doppio dei brevettati)
- Mediamente ogni prescrizione medica prevede due farmaci (mi aspetto che presc_farm sia il doppio di prescrizioni)
- Mediamente ogni vendita è associata a 2 prodotti (mi aspetto che vendita_prodotto sia doppio rispetto a vendita)

TAVOLA DEI VOLUMI

Prodotto	Entità	15000
Farmaco	Entità	10000
Farmaco generico	Entità	8000
Farmaco brevettato	Entità	2000
Profumeria	Entità	5000
Casa farmaceutica	Entità	100
Vendita	Entità	150000

Prescrizione	Entità	15000
Medico	Entità	1500
Paziente	Entità	300000
Produzione	Relazione	10000
Presc_Farm	Relazione	30000
Equivalenza	Relazione	4000
Vendita_Prodotto	Relazione	300000
Vendita_Presc	Relazione	15000
Prescrive	Relazione	30000
Intestatario	Relazione	30000

TAVOLA DELLE OPERAZIONI

OP1	I	100 / giorno
OP2	I	10 / giorno
OP3	I	2 / giorno
OP4	I	1 / giorno
OP5	I	10 / giorno
OP6	B	1 / giorno

TAVOLA DEGLI ACCESSI

Op1 “Inserimento di una nuova vendita (100 volte al giorno)”

Nota: affrontiamo il caso pessimo, il cui la vendita è accompagnata da una prescrizione medica, inserita precedentemente nel sistema. Per semplicità, si assume che nell'esempio in questione si vendano esclusivamente i farmaci previsti dalla prescrizione, non altri prodotti.

Prescrizione	E	L	1
Vendita_Presc	R	S	1
Vendita	E	S	1
Prodotto	E	L	2
Vendita_Prodotto	R	S	2

Costo singola operazione: $1 + 2 + 2 + 2 + 4 = 11$

Costo giornaliero: $11 * 100 = \mathbf{1100}$

Op2 “Ritrovamento di tutti i farmaci generici equivalenti ad un dato farmaco brevettato (10 volte al giorno)”

Nota: Mediamente ogni articolo brevettato ha 2 articoli generici equivalenti

Farmaco_Brevettato	E	L	1
Equivalenza	R	L	2

Costo singola operazione: $1 + 2 = 3$

Costo giornaliero = $3 * 10 = 30$

Op3 "Stampa i medici che hanno prescritto un dato farmaco (10 volte al giorno)"

Nota: Le prescrizioni sono 30000 e i farmaci 10000. Si evince che mediamente ogni farmaco è presente in 3 prescrizioni, quindi andremo a leggere tre prescrizioni. Le prescrizioni che stiamo leggendo sono esattamente quelle che contengono il farmaco che cerchiamo, e per ognuna di esse leggiamo il medico che lo ha prescritto, affrontando il caso pessimo tutte e tre le prescrizioni sono firmate da medici differenti

Presc_Farm	R	L	3
Prescrizione	E	L	3
Prescrive	R	L	3

Costo singola operazione: $3 + 3 + 3 = 9$

Costo giornaliero: $9 * 10 = 90$

Op4 "Inserimento di una prescrizione (10 volte al giorno)"

Nota: mediamente una prescrizione fa riferimento a due farmaci prescrivibili

Prescrizione	E	S	1
Farmaco	R	L	2
Presc_Farm	E	S	2
Paziente	E	L	1
Intestatario	R	S	1
Medico	E	L	1
Prescrive	R	S	1

Costo singola operazione: $2 + 2 + 4 + 1 + 2 + 1 + 2 = 14$

Costo giornaliero = $14 * 10 = 140$

Op5 "Inserimento di un nuovo prodotto (10 volte al giorno)"

Prodotto	E	S	1
----------	---	---	---

Costo singola operazione: $1 * 2 = 2$

Costo giornaliero = $2 * 10 = 20$

Op6 “Stampa le vendite di tutti i prodotti di tipo farmaco brevettato (1 volta al giorno)”

Nota: I farmaci brevettati sono 2000, dobbiamo leggerli tutti supponendo che tutti sono coinvolti in almeno una vendita. Per fare una proporzione realista e capire quanti record vanno letti dalla tabella vendita_prodotto e poi leggere le 2000 vendite eventualmente associate viene fatta una proporzione basata sulla tavola dei volumi

Farmaci brevettati : tutti i prodotti = x : tutte le vendite_prodotto

$$2000 : 15000 = x : 300000$$

$$X = 40000$$

Prodotto	E	L	2000
Vendita_Prodotto	R	L	40000

Costo singola operazione: $2000 + 40000 = 42000$

*Costo giornaliero = $42000 * 1 = 42000$*

(è evidente che su questa operazione sarà necessario inserire un indice in quanto molto dispendiosa)

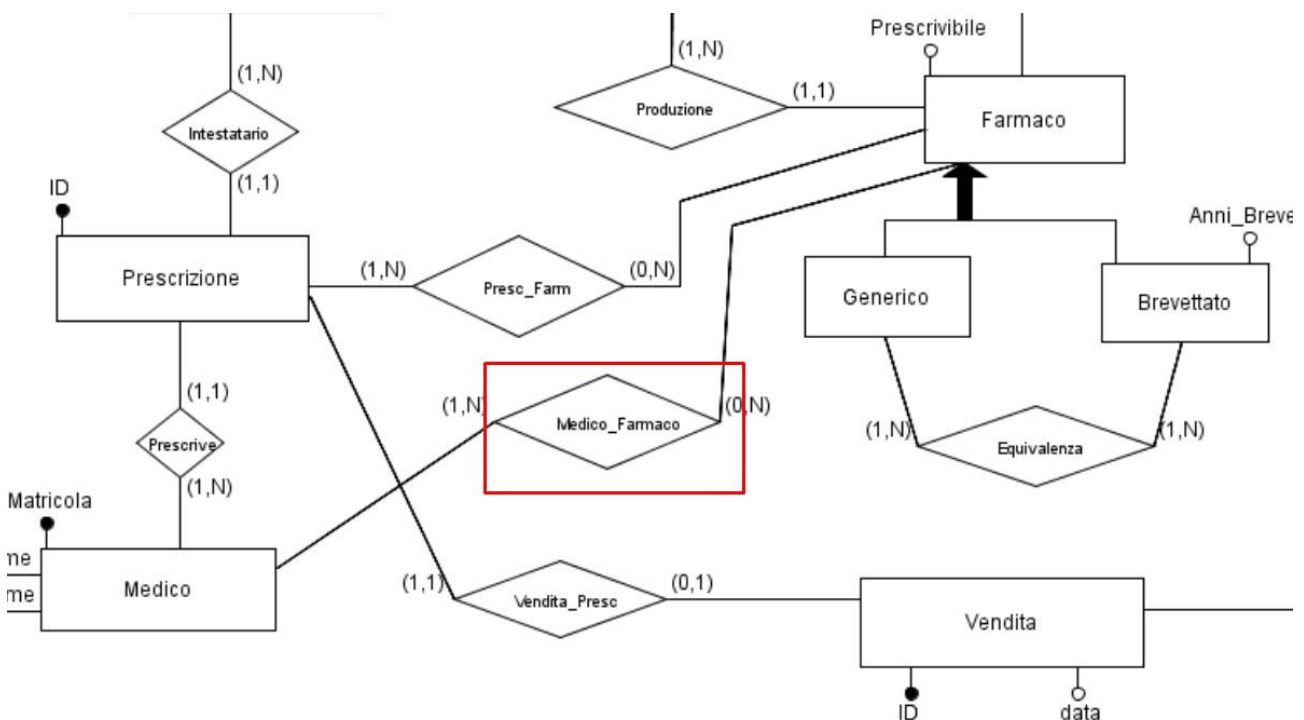
ANALISI DELLE RIDONDANZE

Per le operazioni previste, solo per l'operazione 3 (*Stampa i dati di tutti i medici che hanno prescritto un dato farmaco*) è possibile verificare eventuali migliorie con l'inserimento di una ridondanza. Piuttosto che analizzare tutte le prescrizioni contenenti il farmaco dato, e estrarre i medici che hanno firmato tale prescrizione, ogni volta che un medico firma una prescrizione, possiamo avere una tabella in cui salviamo per ogni farmaco i nomi del medico che lo ha prescritto, se non ancora presente. Con questa soluzione possiamo recuperare direttamente i medici senza passare dall'entità prescrizione.

Questo migliora l'efficienza della ricerca, ma rende più articolato l'inserimento di una prescrizione nel database. L'inserimento di una eventuale ridondanza coinvolge inevitabilmente l'operazione 4 (*inserimento di una prescrizione*).

Facciamo una analisi per vedere se la ristrutturazione dello schema porta effettivamente delle migliorie in termini di efficienza, sommando le complessità dell'operazione 3 dell'operazione 4 prima e dopo l'inserimento della ridondanza.

Segue uno screenshot per visualizzare la ridondanza che intendiamo inserire



Op3

Medico_Farmaco	R	L	3
----------------	---	---	---

Costo giornaliero: 3

Costo effettivo = $3 * 10 = 30$ (rispetto ai 90 senza ridondanza)

Op4

Prescrizione	E	S	1
Farmaco	R	L	2
Presc_Farm	E	S	2
Paziente	E	L	1
Intestatario	R	S	1
Medico	E	L	1
Prescrive	R	S	1
Medico_Farmaco	R	S	2

Costo singola operazione: $2 + 2 + 4 + 1 + 2 + 1 + 2 + 4 = 18$

Costo giornaliero = $18 * 10 = 180$ (rispetto ai 140 senza ridondanza)

Costo senza della ridondanza: $90 + 140 = 230$

Costo con la ridondanza: $30 + 180 = 210$

In conclusione: conviene inserire la ridondanza

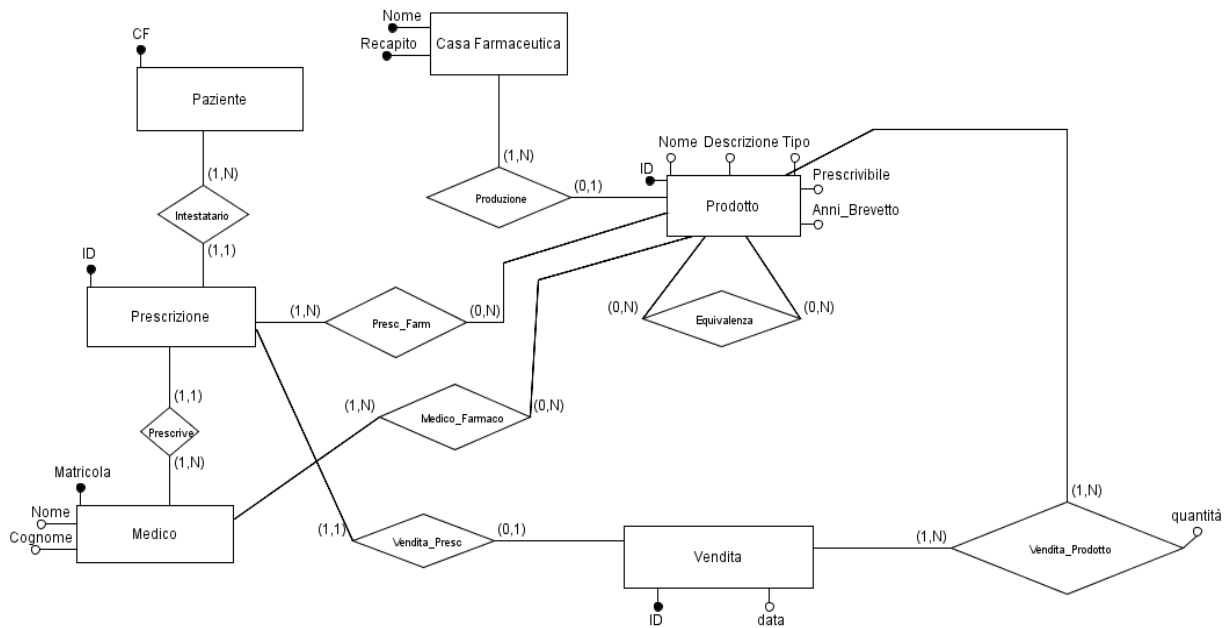
ELIMINAZIONE DELLE GERARCHIE E RISTRUTTURAZIONE DELLO SCHEMA

Nella progettazione concettuale sono state previste alcune generalizzazioni: un prodotto può essere un farmaco o un prodotto di profumeria. I farmaci si dividono in brevettati e generici. I profumi si dividono in prodotti di cosmetica, igiene ed infanzia. L'approccio seguito in questa progettazione è basato sul modello relazionale, quindi le generalizzazioni vanno rimosse, collassando verso l'alto o verso il basso.

La scelta è stata quella di collassare verso l'alto. L'operazione 1 (la più frequente e la più costosa) prevede di poter inserire vendite di prodotti, indipendentemente se questi sono farmaci brevettati, farmaci generici, cosmetici, ecc. Eliminando le entità figlie e lasciando vivere l'entità madre *prodotto* rendiamo più istantanea questa associazione. Viene aggiunto un campo *tipo* all'entità prodotto che potrà assumere i seguenti valori: farmaco brevettato, farmaco generico, cosmetica, igiene, infanzia. Verrà inoltre aggiunto un campo "anno brevetto" che indica gli anni di brevetto rimanenti per ogni prodotto, e tutti i prodotti non brevettati avranno tale campo settato a -1.

Da questa scelta progettuale consegue la necessità di definire alcuni trigger in più, al fine di stabilire i vincoli strutturali e semantici che avremmo ottenuto naturalmente mantenendo le generalizzazioni e seguendo un approccio object-relational. Prendiamo in esempio la frase "ogni farmaco brevettato può avere uno o più farmaci generici equivalenti". Se con un approccio OR sarebbe stato possibile creare una relazione che legasse l'entità *farmaco brevettato* a *farmaco generico*, con l'approccio relazione è necessario inserire un trigger per verificare che ad ogni inserimento sulla tabella *equivalenza*, la coppia di elementi faccia riferimento a due prodotti con il campo *tipo* settato rispettivamente a *farmaco brevettato* e *farmaco generico*. Ancora, quando inseriamo un nuovo prodotto, dobbiamo controllare che nel caso esso sia un *farmaco brevettato*, il campo *anni_brevetto* dovrà necessariamente essere avvalorato ≥ 0 . Viceversa, per tutti i prodotti che non sono farmaci brevettati, bisognerà assicurarsi che il campo dovrà essere avvalorato con il valore di default -1. I trigger verranno approfonditi in seguito in una apposita sezione.

Segue uno screenshot dello schema ristrutturato, da cui verrà costruito lo schema logico.



(per una miglior consultazione: <https://www.dropbox.com/s/sjngst3eny4oemy/ristrutturazione.png?dl=0>)

SCHEMA LOGICO

PAZIENTE (CF)

MEDICO (Matricola, Nome, Cognome)

PRESCRIZIONE (ID, Paziente, Medico)

PRODOTTO (ID, Nome, Descrizione, Tipo, Prescrivibile, Anni_Brevetto)

CASA FARMACEUTICA (Nome, Recapito)

PRODUZIONE (NomeCasaFarmaceutica, RecapitoCasaFarmaceutica, Prodotto)

PRESC_FARM (Prescrizione, Prodotto)

MEDICO_FARMACO (Medico, Prodotto)

VENDITA (ID, data, Prescrizione)

VENDITA_PRODOTTO (Vendita, Prodotto)

EQUIVALENZA (ProdottoBrevettato, ProdottoEquivalente)

PROGETTAZIONE FISICA

TRIGGER

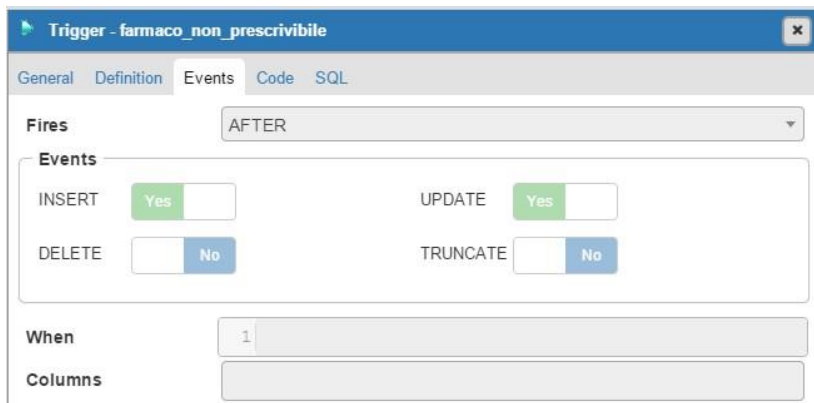
Segue l'elenco dei trigger previsti, con una apposita descrizione e due screenshot ciascuno. I trigger sono stati creati con pgAdmin4, quindi viene allegato lo screenshot della definizione del trigger e quello della definizione della funzione che il trigger chiama.

T1

Tabella: Prescr_Farm

Descrizione: Nel momento in cui si inserisce una coppia (prescrizione – prodotto) bisogna assicurarsi che il prodotto sia di tipo “farmaco prescrivibile”, ovvero che il valore prodotto.prescrivibile sia TRUE. In questo caso viene fatto il rollback dell'inserimento sulla tabella *prescrizione_farmaci*.

Screenshot:



```
1 DECLARE
2     result BOOLEAN;
3 BEGIN
4     SELECT prescrivibile INTO result FROM prodotto WHERE id = new.farmaco;
5     IF result = false THEN
6         RAISE EXCEPTION '% non è un farmaco prescrivibile', new.farmaco;
7     END IF;
8     RETURN new;
9 END;
10
```

T2

Tabella: Presc_Farm

Descrizione: Nel momento in cui viene inserita una nuova coppia (prescrizione, prodotto) nella tabella *presc_farm*, viene aggiornata la tabella *medico_farmaco*. Questo viene fatto per mantenere automaticamente aggiornata la ridondanza che si è deciso di inserire.

Screenshot:

Trigger - aggiorna_medico_farmaco

General Definition Events Code SQL

Fires: AFTER

Events:

INSERT: Yes ☐ UPDATE: No ☐

DELETE: No ☐ TRUNCATE: No ☐

When: 1

Columns:

```

1 DECLARE
2     med BIGINT;
3 BEGIN
4     SELECT medico INTO med FROM prescrizione WHERE id = new.prescrizione;
5     INSERT INTO medico_farmaco values (med, new.farmaco);
6     RETURN new;
7 END;

```

T3

Tabella: Vendita_Prodotto

Descrizione: Nel momento in cui si cerca di registrare la vendita di un prodotto acquistabile previa prescrizione medica, bisogna controllare: 1) se c'è una prescrizione medica associata alla vendita 2) se la prescrizione prevede l'acquisto del prodotto che si vuole registrare

Screenshot:

Trigger - check_vendita_prescritta

General Definition Events Code SQL

Fires: AFTER

Events:

INSERT: Yes ☐ UPDATE: Yes ☐

DELETE: No ☐ TRUNCATE: No ☐

When: 1

Columns:

```

1 DECLARE is_presc BOOLEAN;
2 DECLARE presc INTEGER;
3 DECLARE n INTEGER;
4
5 BEGIN
6     SELECT prescrivibile INTO is_presc FROM prodotto WHERE id = new.prodotto;
7     IF is_presc = true THEN
8         SELECT prescrizione INTO presc FROM vendita WHERE id = new.vendita;
9         IF presc IS NULL THEN
10             RAISE EXCEPTION 'Stai cercando di acquistare un prodotto senza prescrizione medica';
11         END IF;
12
13         SELECT COUNT(*) INTO n FROM prescrizione_farmaci WHERE prescrizione = presc AND farmaco = new.prodotto;
14         IF n = 0 THEN
15             RAISE EXCEPTION 'La prescrizione medica non è idonea per acquistare il prodotto %', new.prodotto;
16         END IF;
17     END IF;
18     RETURN new;
19 END;

```

T4

Tabella: Prodotto

Descrizione: Quando si inserisce un prodotto di tipo diverso da *farmaco brevettato* bisogna assicurarsi che il campo *anni_brevetto* sia avvalorato con il valore di default -1.

Screenshot:

Trigger - check_anni_brevetto_prodotto_non_brevettato

General Definition Events Code SQL

Fires: AFTER

Events:

INSERT	<input checked="" type="checkbox"/> Yes	UPDATE	<input checked="" type="checkbox"/> Yes
DELETE	<input type="checkbox"/> No	TRUNCATE	<input type="checkbox"/> No

When: 1 (((new.tipo)::text <> 'farmaco brevettato')::text))

Columns:

```

1 DECLARE anni INTEGER;
2 BEGIN
3     SELECT anni_brevetto INTO anni FROM prodotto WHERE id = new.id;
4     IF anni <> -1 THEN
5         RAISE EXCEPTION 'Non puoi inserire prodotto nonbrevettato avvalorando il campo anni_brevetto';
6     END IF;
7     RETURN new;
8 END;

```

T5

Tabella: Prodotto

Descrizione: In analogia a quanto avviene in T4, bisogna verificare che quando si inserisce un prodotto di tipo *farmaco brevettato*, il campo *anni_brevetto* sia maggiore o uguale a 0 (al massimo il brevetto può essere scaduto se uguale a 0).

Screenshot:



```

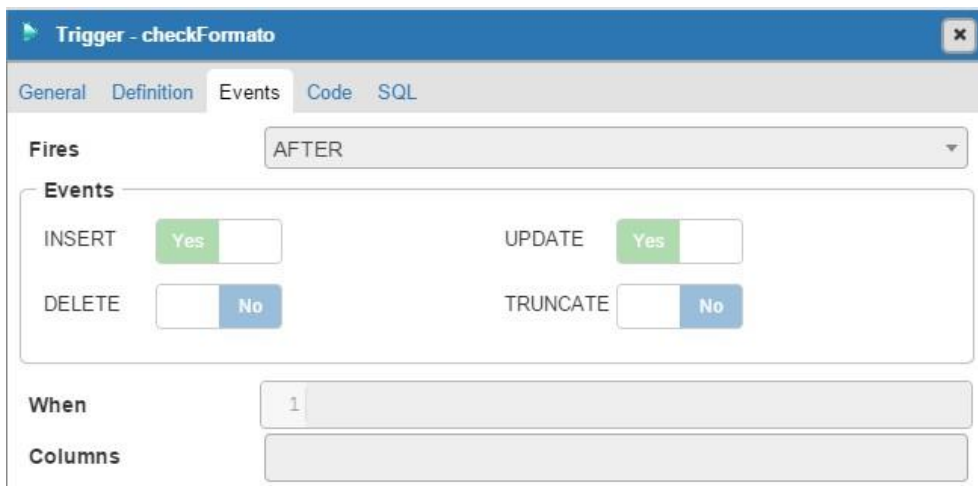
1 DECLARE anni INTEGER;
2 BEGIN
3     SELECT anni_brevetto INTO anni FROM prodotto WHERE id = new.id;
4     IF anni < 0 THEN
5         RAISE EXCEPTION 'Non puoi inserire un farmaco brevettato senza avvalorare il campo anni_brevetto';
6     END IF;
7     RETURN new;
8 END;

```

T6**Tabella: Equivalenza**

Descrizione: Quando viene inserita una nuova coppia (farmaco brevettato – farmaco equivalente) nella tabella bisogna assicurarsi che il primo prodotto inserito sia di tipo *farmaco brevettato* e il secondo sia di tipo *farmaco generico*.

Screenshot:




```

1 DECLARE brev TEXT;
2 DECLARE equiv TEXT;
3
4 BEGIN
5     SELECT tipo INTO brev FROM prodotto WHERE id = new.farmaco_brevettato;
6     SELECT tipo INTO equiv FROM prodotto WHERE id = new.farmaco_equivalente;
7     IF brev <> 'farmaco brevettato' OR equiv <> 'farmaco generico'
8     THEN RAISE EXCEPTION 'la coppia inserita non è nel formato (farmaco brevettato - farmaco equivalente)';
9     END IF;
10    RETURN new;
11 END;

```

INDICI

Per quasi tutte le operazioni vanno bene gli indici automaticamente definiti sulle chiavi primarie delle relazioni, ad eccezione dell'operazione 6, di cui per comodità viene riportato il testo: *"Stampa le vendite di tutti i prodotti di tipo farmaco brevettato"*.

farmacia on postgres@PostgreSQL 9.6

```

1 SELECT p.id AS prodotto, p.nome, v.id AS vendita
2 FROM prodotto p, vendita_prodotto vp, vendita v
3 WHERE p.id = vp.prodotto AND vp.vendita = v.id
4 AND p.tipo = 'farmaco brevettato';

```

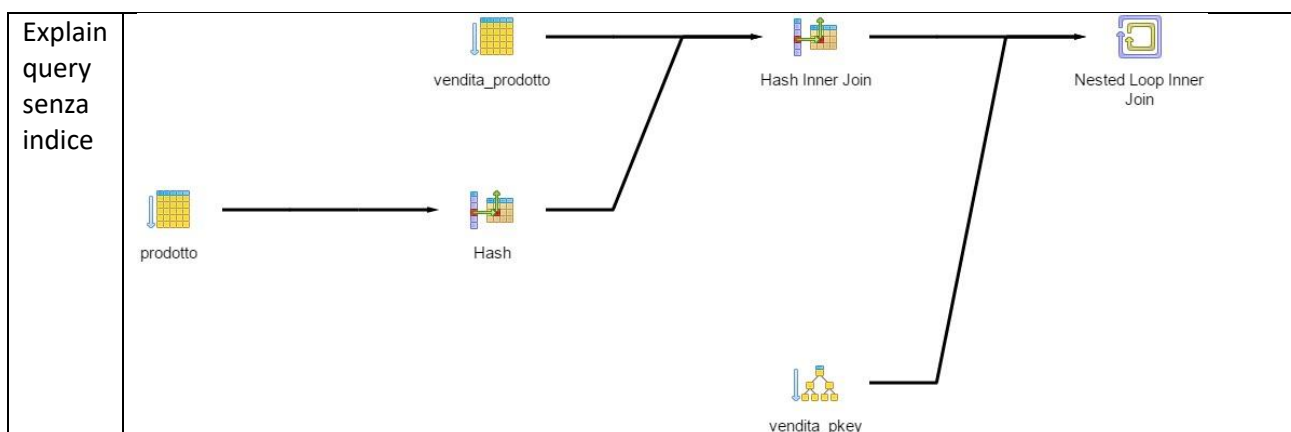
L'operazione coinvolge una operazione di filtering dei record basata sul campo *tipo*, in quanto l'intenzione è quella di isolare solo le vendite che coinvolgono prodotti di un certo tipo. Un indice di tipo BTREE è stato creato sul campo *Prodotto.Tipo* e i miglioramenti sono stati evidenti.

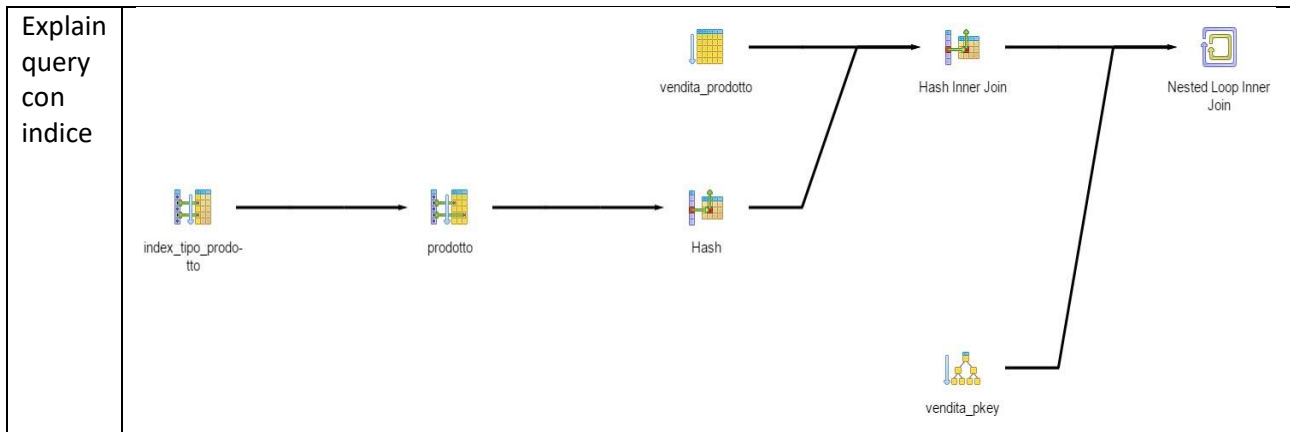
Comando per la creazione dell'indice:

CREATE INDEX index_tipo_prodotto ON prodotto USING btree (tipo);

Per testare l'efficienza dell'indice si è popolato il database con 45000 record generati casualmente, simulando un caso pessimo (inserimento di blocchi sparsi di record con il campo *tipo* differente). La query è stata lanciata sul database prima senza indice, e poi con. La query senza indice è stata eseguita in *1 secondo*, mentre la stessa query eseguita dopo la creazione dell'indice è stata eseguita in tempo decisamente inferiore, ovvero *667 msec*.

Seguono degli screenshot degli explain, estratti dal *query tool* di *pgAdmin4*

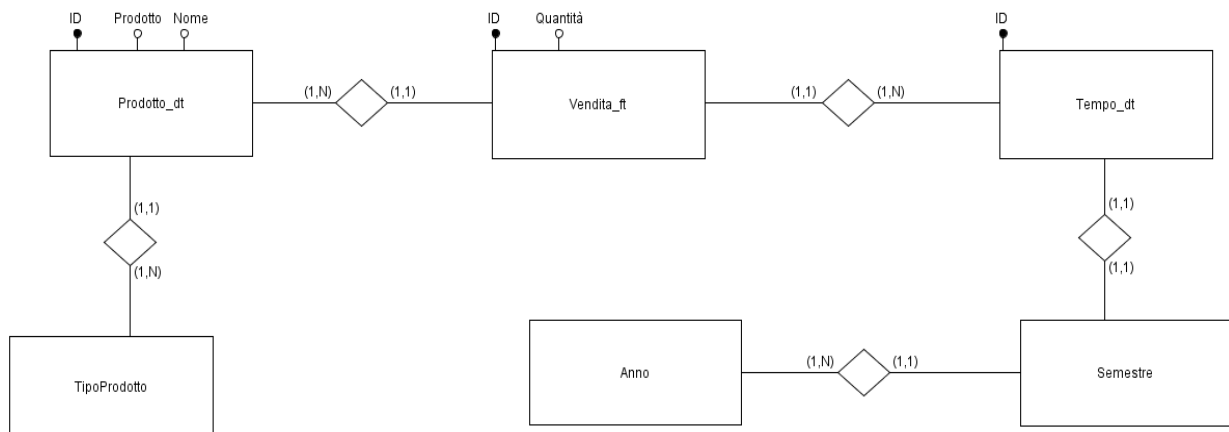




DATAWAREHOUSE

“Inoltre la farmacia è interessata a svolgere indagini statistiche sulle vendite e intende memorizzare i report, semestrali e annuali, delle vendite realizzate”

Questa frase ha guidato la progettazione del datawarehouse per step successivi. Sicuramente la tabella dei fatti è costituita dalle vendite. Le dimensioni scelte sono state quella del tempo (semestre e anno) e quella del tipo dei prodotti, per vedere quanti prodotti di un certo tipo sono stati venduti in un certo periodo.



VENDITA_FT (ID, Prodotto, Quantità, Tempo)

TEMPO_DT (ID, Semestre, Anno)

PRODOTTO_DT (ID, Prodotto, Nome, Tipo)

Dopo aver progettato il datawarehouse, tramite il tool *Schema Workbench* è stato generato il file XML, consultabile al seguente link:

<https://github.com/holydrinker/bdii/blob/master/database/datawarehouse/farmacia.xml>

Sono state previste delle tabelle “*audit*”, popolate da un apposito trigger, che memorizzano i dati che sono stati inseriti nella base di dati ma non sono ancora stati sincronizzati con il database. La fase di aggiornamento può essere gestita dal web server scritto in java, in quanto è stato previsto un apposito servizio.

Il trigger scatta ogni volta che aggiornata la tabella *vendita_prodotto* e si preoccupa di mantenere aggiornate le tabelle audit di vendita e prodotto, e aggiorna la dimensione tempo. Più nel dettaglio:

- Aggiornare la tabella della dimensione tempo se la nuova vendita è la prima di un nuovo semestre
- Aggiornare la tabella prodotto audit (se il prodotto appena inserito nella vendita non era già presente né nella dimensione prodotto, né nella tabella audit stessa)
- Aggiornare la tabella vendita audit

Seguono degli screenshot per documentare il trigger:

Trigger - aggiorna_dw

General Definition Events Code SQL

Fires: AFTER

Events:

INSERT: ☒ Yes ☐ No UPDATE: ☐ No ☒ No

DELETE: ☐ No ☒ No TRUNCATE: ☐ No ☒ No

When: 1

Columns:

```

1 DECLARE anno_dt TEXT;
2 DECLARE mese INTEGER;
3 DECLARE semestre_dt TEXT;
4 DECLARE counter INTEGER;
5 DECLARE tempo_id INTEGER;
6 DECLARE prodotto_id INTEGER;
7 DECLARE quantita_n INTEGER;
8 DECLARE id_max INTEGER;
9 DECLARE prodotto_count INTEGER;
10 DECLARE prodotto_nome TEXT;
11 DECLARE prodotto_tipo TEXT;
12 DECLARE prodotto_id_str TEXT;
13 DECLARE fake_id INTEGER;
14
15 BEGIN
16     -- Recuperare tutti i dati che servono per il dw
17     SELECT EXTRACT (year FROM data), EXTRACT (month FROM data),
18     vendita_prodotto.prodotto, vendita_prodotto.quantita,
19     prodotto.nome, prodotto.tipo
20     INTO anno_dt , mese, prodotto_id, quantita_n, prodotto_nome, prodotto_tipo
21     FROM vendita, vendita_prodotto, prodotto
22     WHERE vendita_prodotto.vendita = new.vendita
23     AND vendita_prodotto.prodotto = new.prodotto
24     AND vendita.id = vendita_prodotto.vendita
25     AND vendita_prodotto.prodotto = prodotto.id;
26

```

```

27  -- Gestione del tempo
28  IF mese <= 6 THEN
29      semestre_dt = '01-' || anno_dt;
30  ELSE
31      semestre_dt = '02-' || anno_dt;
32  END IF;
33
34  SELECT COUNT(*) INTO counter FROM tempo_dt WHERE anno = anno_dt AND semestre = semestre_dt;
35  IF counter = 0 THEN
36      INSERT INTO tempo_dt(semestre, anno) VALUES (semestre_dt, anno_dt);
37  END IF;
38
39  SELECT id INTO tempo_id FROM tempo_dt WHERE semestre = semestre_dt AND anno = anno_dt;
40
41  -- Aggiornare la tabella prodotto_audit se serve
42  prodotto_id_str = prodotto_id::text;
43  SELECT COUNT(*) INTO prodotto_count FROM prodotto_audit WHERE prodotto = prodotto_id_str;
44  IF prodotto_count = 0 THEN
45      SELECT COUNT(*) INTO prodotto_count FROM prodotto_dt WHERE prodotto = prodotto_id_str;
46      IF prodotto_count = 0 THEN
47          INSERT INTO prodotto_audit(prodotto, nome_prodotto, tipo_prodotto)
48          VALUES (prodotto_id, prodotto_nome, prodotto_tipo);
49      END IF;
50  END IF;
51
52  -- Aggiornare la vendita
53  INSERT INTO vendita_audit(tempo, quantita, prodotto) VALUES (tempo_id, quantita_n, prodotto_id);
54
55  RETURN new;
56 END;
57

```

Una volta popolato il datawarehouse è stato possibile usufruire delle funzionalità OLAP di *Mondrian*, che permettono di navigare il datawarehouse tramite l'interfaccia grafica *JPivot*. Per funzionare, è stato necessario scrivere una pagina.jsp, consultabile a questo link:

<https://github.com/holydrinker/bdii/blob/master/database/datawarehouse/farmacia.jsp>

Nel file.jsp viene effettuata una query MDX, di seguito riportata:

```

SELECT {[Measures].[quantita]} ON COLUMNS,
{([Prodotto],[Tempo])} ON ROWS
FROM [Vendite]

```

ALTRE TECNOLOGIE

Dopo aver realizzato il database è stata scritta una semplice web app per interagirvi. L'applicazione segue il classico modello architetturale three-tier, ed è quindi dotata di:

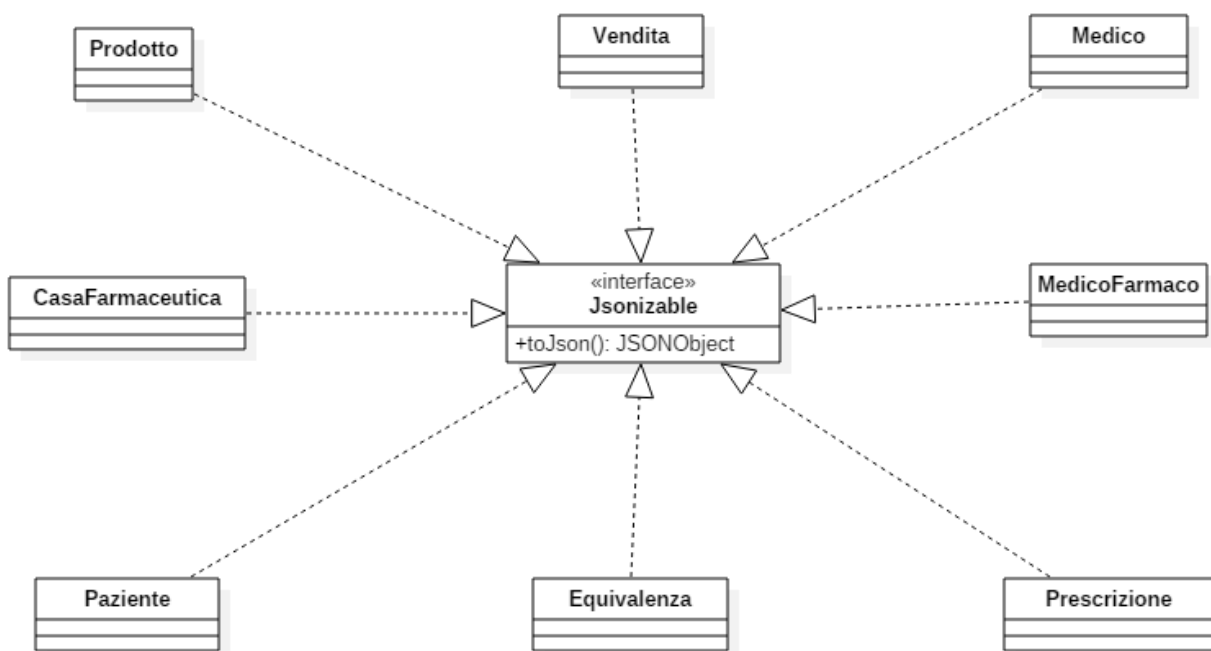
- Una componente grafica lato client, che permette agli utenti di visualizzare il database ed interagirvi in maniera semplice e immediata
- Una componente di business logic lato server, responsabile della diretta interazione con la base di dati
- Una componente dati, appunto il database, ampiamente descritta finora.

Server

Per realizzare il server è stato utilizzato [SparkJava](#), un micro framework che permette di creare delle REST APIs in maniera semplice e veloce grazie all'utilizzo di Java 8.

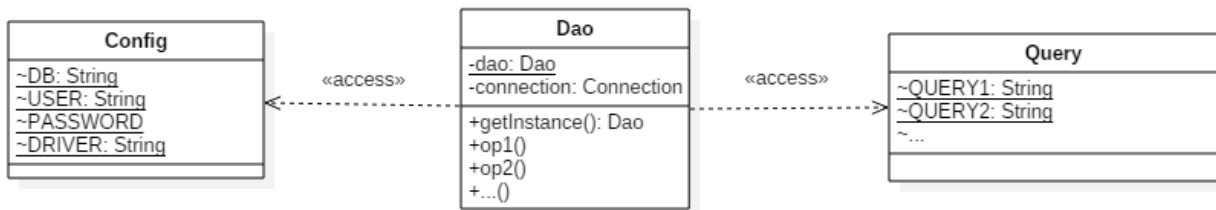
La fase di scrittura di codice è stata preceduta da una fase di progettazione. L'ingegnerizzazione della componente server è stata raggiunta tramite la realizzazione di tre blocchi interagenti.

- *Classi entity*: per limitare problema del [impedance mismatch](#) fra la base di dati e le strutture dati nel livello di business sono state create delle classi entity, ognuna corrispondente a una tabella del database. Come già detto Spark Java permette di creare delle REST APIs che, come da protocollo, scambiano i dati in formato JSON. Per questo motivo tutte le classi entity estendono una interfaccia dotata di un unico metodo, che permette ad ognuna di esse di convertirsi in formato JSON, pronte per essere trasferire dal lato server al lato client.

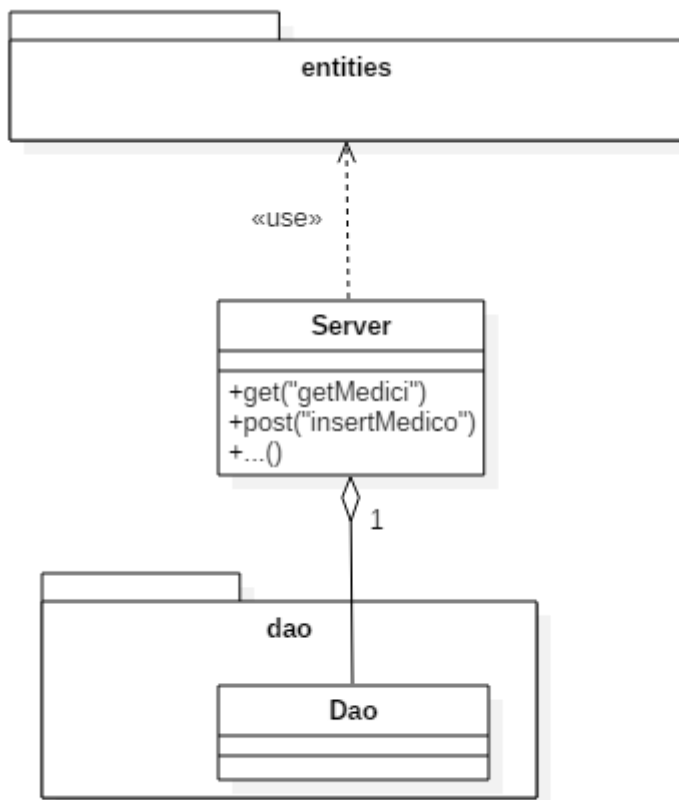


- *Data Access Object*: per distribuire gli interessi in maniera equa e sensata fra le varie classi, si è ritenuto opportuno isolare la responsabilità di connessione al database e di effettuare le query in package apposito. L'approccio seguito si ispira al pattern denominato *DataAccessObject* (seppur in una versione semplificata). Il package è dotato di tre classi: una contiene gli elementi per la configurazione e la connessione al database (visibilità di package), un'altra contiene tutte le query (visibilità di package), l'ultima (pubblica) è quella nella quale vengono centralizzate le responsabilità di questa fase, e utilizza le due classi precedenti per operare effettivamente sulla base di dati.

Inoltre, si è adottato il pattern *Singleton*, per garantire che tutte le richieste verso il database siano intercettate da una e una sola istanza di classe DAO.



- *Servizi*: rappresentati sostanzialmente da una sola classe, che non è altro che una collezione di servizi con lo scopo di intercettare le richieste del client, recuperare i dati dal database tramite l'ausilio del package dao e aiutandosi con le classi entity, da "jsonizzare" e rispedire al client in risposta (necessaria per le GET, non richiesta per le POST).



Lato Client

La componente client è stata realizzata con il framework MVC [AngularJS](#). Il framework nasce con l'idea di favorire lo sviluppo delle Single Page Application, ovvero Web Application che girano all'interno di una singola pagina HTML allo scopo di favorire una UX più fluida, comparabile a quella di una applicazione desktop. Questo avviene tramite:

- *Chunking*: caricare piccoli frammenti di HTML da iniettare nella (unica) pagina in base alla operazione da svolgere e modificare l'interfaccia grafica

- *Templating*: creare dei “binding a due vie”, ovvero mappare un elemento dell’interfaccia grafica con un elemento del modello (le strutture dati). Il binding farà sì che i valori verranno tenuti aggiornati sia quando modificati lato client, sia quando modificati lato server
- *Routing*: navigazione all’interno dell’interfaccia iniettando html (*chucking*) e preservando lo stato delle pagine. Questo avviene associando ad ogni *view* un *controller* apposito. La configurazione del routing avviene tramite un apposito file javascript (segue uno screenshot di esempio)

```
$routeProvider
  .when('/', {
    templateUrl: 'views/home.html'
  })
  .when('/pazienti', {
    templateUrl: 'views/pazienti.html',
    controller: 'PazientiCtrl'
  })
  .when('/medici', {
    templateUrl: 'views/medici.html',
    controller: 'MediciCtrl'
  })
```

Infine, ultimo concetto ingegneristicamente importante, è il principio della *dependency injection*, secondo il quale i controller javascript non creano dipendenze fra loro, bensì in ogni controller vengono iniettate le dipendenze utili a realizzare i task di cui sono responsabili. Esistono sia servizi già presenti nel framework, sia installabili da terze parti, sia implementabili dal programmatore. Nell’ambito di questo progetto è stata una caratteristica molto utile, in quanto mi ha permesso di scrivere un servizio per ogni entità, e ogni servizio incapsulava un insieme chiamate al server, per interagire indirettamente con la base di dati. Ogni servizio veniva iniettato nell’apposito controller. Questo è un ottimo aspetto che permette di realizzare al meglio la separazione di interessi, il basso accoppiamento e la manutenibilità del software. Segue uno screenshot del controller responsabile dei pazienti nel quale vengono iniettati i servizi **\$scope** e **\$route** già presenti nel sistema, e il servizio **pazientiFactory** implementato da me.

```
app.controller('PazientiCtrl', ['$scope', '$route', 'pazientiFactory',
  function ($scope, $route, pazientiFactory) {
```

Per aumentare l’usabilità e l’UX della componente client è stato utilizzato [bootstrap](#).

Tutto il codice scritto per questo progetto è reperibile al seguente link, database e datawarehouse compresi: <https://github.com/holydrinker/bdii>

Istruzioni per l’utilizzo: <https://github.com/holydrinker/bdii/blob/master/README.md>