Progettazione e implementazione di una base di dati per la gestione di un supermercato

BAZZANA LORENZO	147569	bazzana.lorenzo@spes.uniud.it
BORELLI ROBERTO	147025	borelli.roberto@spes.uniud.it
D'AMBROSI DENIS	147681	dambrosi.denis@spes.uniud.it
ZANOLIN LORENZO	148199	zanolin.lorenzo@spes.uniud.it

Progetto Basi di Dati anno accademico 2021/2022
Corso di laurea in Informatica, Università degli studi di Udine

Indice

1 Raccolta e analisi dei requisiti

1.1 Studio della richiesta

1.1.1 Richiesta originale

Si vuole progettare una base di dati per la gestione di un supermercato, contenente le seguenti informazioni:

- Per ogni dipendente, il codice identificativo, il nome e il cognome, le eventuali persone a carico, l'indirizzo e il reparto di appartenenza.
- Per ogni reparto, il nome, i dipendenti, il responsabile del reparto e gli articoli in vendita.
- Per ogni articolo in vendita, il nome, il fornitore, il prezzo di vendita e due codici identificativi (uno assegnatogli dal fornitore, che identifica univocamente l'articolo nell'insieme degli articoli da lui forniti, l'altro dal supermercato, che identifica univocamente l'articolo all'interno del reparto cui è stato assegnato).
- Per ogni fornitore, il nome, l'indirizzo e gli articoli che esso fornisce al supermercato (con i relativi prezzi).

Si assuma che, in ogni istante, ogni articolo venga fornito da un solo fornitore e che tale fornitore possa variare nel tempo.

1.1.2 Ampliamento della richiesta

Al fine di gestire alcuni ricorrenti problemi correlati alla creazione di basi di dati che altrimenti non sarebbero stati trattati nella risoluzione del problema originale, abbiamo introdotto alcune ulteriori richieste alla consegna prima di proseguire con la fase di analisi e progettazione.

- Si vuole tenere traccia di uno storico degli ordini effettuati ai fornitori. Ogni ordine deve contenere una lista di articoli, tutti forniti al momento dell'ordine dallo stesso fornitore.
- Per ogni fornitore (presente o passato) si vuole tenere traccia degli ordini effettuati e l'indirizzo.
- Per ogni reparto si vuole tenere traccia del numero di dipendenti.
- Per ogni dipendente si vogliono salvare il/i numero/i di telefono.

1.1.3 Assunzioni effettuate

Per proseguire il lavoro con la fase di progettazione concettuale, abbiamo effettuato le seguenti assunzioni sulle richieste del testo:

- Un articolo che viene introdotto nel database apparterrà sempre all'inventario del suo reparto. Se si vuole smettere di vendere un prodotto, si mantiene comunque l'articolo nella base di dati con una corrispondenza al reparto in cui era esposto.
- Un fornitore è identificato dalla sua Partita IVA e ha un unico indirizzo di cui si vuole tenere traccia.
- Si vogliono tenere traccia di persone a carico di dipendenti solo all'interno dell'insieme dei dipendenti stessi e ogni persona a carico può dipendere al massimo da un altro dipendente.
- Ciascun reparto ha un unico responsabile, che è responsabile esclusivamente di quel reparto e non ci lavora come dipendente normale.
- Si vuole tenere traccia anche dello stipendio percepito da ciascun dipendente.

1.2 Requisiti strutturati

1.2.1 Glossario

Per rendere più chiaro il significato dei termini chiave e le relazioni che i requisiti strutturati definiscono tra essi viene fornito un glossario esplicativo:

Termine	**Descrizione**	**Termini correlati**
Dipendente	Lavoratore del supermercato. Sono assegnati a un unico reparto e possono avere persone a carico.	Reparto, persone a carico
Reparto	Componente del supermercato. Ogni reparto contiene una certa categoria di articoli ed è gestito da un responsabile del reparto.	Dipendente, articolo, responsabile del reparto
Persone a carico	Persone all'interno dell'azienda legate a un dipendente da un legame familiare (es. il padre ha a carico uno o più figli).	Dipendente
Articolo	Tipo di prodotto venduto dal supermercato. Ogni istanza di un determinato articolo si trova in un singolo reparto ed è fornito da un fornitore.	Reparto, fornitore
Fornitore	Entità che rifornisce il supermercato di uno o più articoli in ogni dato momento	Articolo
Responsabile del reparto	Dipendente che gestisce un determinato reparto.	Dipendente, Reparto

Tabella 1: Glossario

1.3 Requisiti operazionali

Verranno di seguito descritte alcune operazioni significative pensate per analizzare i costi di input/output degli accessi alla base di dati in uno scenario realistico per un supermercato in attività. Non essendo esplicitamente dichiarate nella consegna, esse sono state individuate in modo da ricoprire sezioni diverse all'interno dello schema concettuale per cercare di distribuire l'interesse dell'analisi verso tutto il suo contenuto. Le operazioni vengono effettuate con varie frequenze e comprendono la lettura e la scrittura di dati ridondanti appositamente per fornire la base ad un'estensiva attività di analisi dei costi e delle ridondanze.

- *OPERAZIONE 1* : Per ogni reparto è richiesto di restituire il *Numero di Dipendenti* e lo stipendio medio. Questa operazione permette di effettuare una successiva analisi delle ridondanze in quanto l'attributo numero di dipendenti è ricavabile eseguendo un'interrogazione sulla tabella *DIPENDENTE*. Questa operazione è prevista per essere usata circa 13 volte l'anno.
- *OPERAZIONE 2* : Effettuare un'assunzione, effettuare un licenziamento. Questa operazione permette di rimuovere e aggiungere del personale dalla tabella *DIPENDENTE*. Questa operazione è prevista per essere usata circa 50 volte all'anno.

- *OPERAZIONE 3* : Restituire il numero di ordini effettuati presso un fornitore che al momento fornisce almeno un articolo. Questa operazione tiene conto degli ordini effettuati (anche in passato) da un fornitore che al momento ha un contratto con noi. Questa operazione è prevista per essere usata circa 2 volte al mese.
- *OPERAZIONE 4* : Creazione di un nuovo ordine presso un fornitore con cui abbiamo un contratto attivo. Questa è l'operazione più utilizzata dell'intera base di dati. Questa operazione è prevista per essere usata circa 20 volte al giorno.
- *OPERAZIONE 5* : Visualizzare la quantità ordinata di un determinato articolo in un preciso periodo. Questa operazione richiede il reparto e un range di date in cui cercare lo storico degli ordini. Questa operazione è prevista per essere usata 20 volte al mese, precisamente una volta al mese per 20 articoli differenti.

2 Progettazione concettuale

2.1 Costruzione dello schema Entità Relazione

Per la progettazione dello schema E-R è stata utilizzata la tecnica bottom-up: per prima cosa si isolano le richieste del testo in concetti indipendenti e successivamente si costruisce un piccolo schema entità-relazione per ciascuno di essi. I singoli diagrammi sono stati poi integrati in un unico schema che soddisfa tutti le richieste del documento.

2.1.1 Dipendenti e le loro persone a carico

Siccome le persone a carico appartengono al dominio dei dipendenti, per modellare questa situazione abbiamo introdotto un'entità *DIPENDENTE* e la relazione *SI OCCUPA DI* ricorsiva su di essa, con due ruoli: *CAPOFAMIGLIA* e *PERSONA A CARICO*. La cardinalità della relazione è *(0,N)* per il lato del *CAPOFAMIGLIA* perché una persona si può occupare di uno o più familiari, mentre è *(0,1)* per la *PERSONA A CARICO* in quanto dipende al massimo da un *CAPOFAMIGLIA*.

L'entità *DIPENDENTE* è identificata univocamente dalla chiave candidata *CodID* ed è caratterizzata dal nome, dal cognome, dall'indirizzo di casa e da uno o più numeri di telefono.

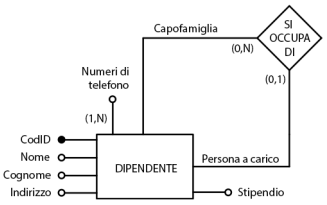


Figura 1: Sottoschema concettuale delle persone a carico dei dipendenti.

2.1.2 Dipendenti e il loro ruolo all'interno di un reparto

Dalle richieste e dalle nostre assunzioni possiamo dedurre la presenza di due ruoli mutualmente esclusivi all'interno del supermercato: gli impiegati appartenenti a *RESPONSABILE* e quelli appartenenti a *NON RESPONSABILE*. Abbiamo dunque introdotto due entità, specializzazioni disgiunte dell'entità generica *DIPENDENTE*, che presentano relazioni differenti con l'entità *REPARTO* in base al loro ruolo. Un *REPARTO* è identificato univocamente dal suo *Nome* ed è caratterizzato dall'attributo derivato *Numero dipendenti*. I responsabili gestiscono un reparto, che è gestito esclusivamente da loro: di conseguenza, la relazione *GESTISCE* è una relazione *one-to-one*. Al contrario, molti dipendenti afferiscono esclusivamente un reparto, quindi la relazione *AFFERISCE* presenta cardinalità *(1,N)* sul lato del *REPARTO* e *(1,1)* su quello di *NON RESPONSABILE*. Per semplificare lo schema si sarebbe potuta evitare la generalizzazione di *DIPENDENTE* nelle sue specializzazioni, tuttavia sarebbe stato necessario rafforzare i vincoli di esclusività del manager col suo reparto successivamente con dei vincoli aziendali: senza questi ultimi si potrebbero infatti verificare situazioni erronee come ad esempio impiegati che sono contemporaneamente manager e dipendenti "ordinari" di un reparto, così come responsabili che gestiscono un reparto che però afferiscono un altro reparto. Si è preferito garantire la coerenza della base di dati direttamente attraverso il formalismo dello schema E-R.

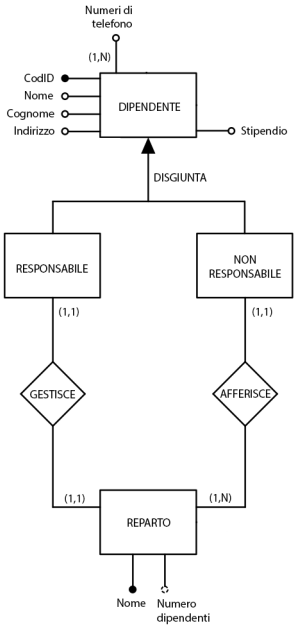


Figura 2: Sottoschema concettuale sui ruoli dei dipendenti.

2.1.3 Reparti e il loro inventario

Ciascun reparto vende un insieme di articoli, che sono univocamente legati ad esso. Dalle richieste si evince che *ARTICOLO* debba essere un'entità debole, in quanto il suo codice lo identifica univocamente solo all'interno del suo reparto. Di conseguenza, dopo aver introdotto la relazione *many-to-one VENDE* tra *REPARTO* e *ARTICOLO* (con cardinalità *(1,N)* dal lato di *REPARTO*) discriminiamo quest'ultimo attraverso la chiave esterna *REPARTO* e l'attributo *Codice Supermercato*. Ulteriori attributi che lo caratterizzano sono il nome e il prezzo al dettaglio. Da notare le cardinalità della relazione *VENDE*: il minimo dalla parte di *REPARTO* è *1* in quanto non avrebbe senso un reparto senza prodotti, mentre dal lato *ARTICOLO* la relazione ha cardinalità *(1,1)* in quanto entità debole. La nostra prima assunzione ci permette quindi questa modellazione concettuale.

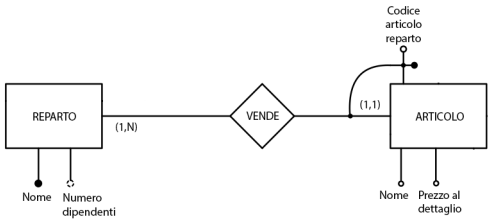


Figura 3: Sottoschema concettuale sull'inventario dei reparti.

2.1.4 Articoli e fornitori

Siccome i fornitori possono variare nel tempo, si è utilizzato uno schema di progettazione che consentisse la storicizzazione. Avendo introdotto precedentemente *ARTICOLO*, introduciamo l'entità *FORNITORE*, caratterizzata dal nome, l'indirizzo (attributo composto a sua volta da via, civico, città e CAP), l'attributo derivato *Numero ordini* e identificato univocamente dalla chiave candidata *Partita IVA*.
Ciascun fornitore fornisce al momento almeno un articolo: di conseguenza la relazione *one-to-many FORNISCE AL MOMENTO* presenta cardinalità *(1,N)* dalla parte del *FORNITORE*. Questa rela-

zione è caratterizzata dal prezzo all'ingrosso dell'articolo stesso. Dalle richieste possiamo notare che *ARTICOLO* può essere identificato dal suo fornitore con un codice assegnatogli da quest'ultimo: di conseguenza *ARTICOLO* è un'entità debole anche rispetto a *FORNITORE*. Nello schema E-R abbiamo introdotto un'apparente violazione d'integrità causata dalla partecipazione non obbligatoria di *FORNITORE* alla relazione con *ARTICOLO*: questo infatti non sarebbe possibile teoricamente in quanto *FORNITORE* è chiave esterna. Nonostante ciò, questo "errore" è necessario per permettere l'esistenza di articoli per i quali è cessata la fornitura e non ha conseguenze sull'implementazione reale della base di dati in quanto sappiamo che *ARTICOLO* presenta un'ulteriore chiave candidata che possiamo utilizzare per discriminare tra le sue istanze. Per mantenere lo storico delle forniture passate, abbiamo reificato la relazione *FORNIVA* nell'entità debole *FORNITURA PASSATA*. Quest'ultima è in relazione *FPF* con *FORNITORE* e *FPA* con *ARTICOLO*. Le due relazioni, assieme alla *Data Fine* la identificano, mentre solo l'attributo *Data Inizio* la caratterizza. Un *FORNITORE* può avere fornito 0 o più forniture passate e ciascun *ARTICOLO* può essere oggetto di 0 o più di esse. Al fine di garantire la coerenza nella base di dati, è necessario imporre un vincolo aziendale sulle forniture passate per controllare che per ciascun articolo non siano presenti due intervalli di fornitura sovrapposti.

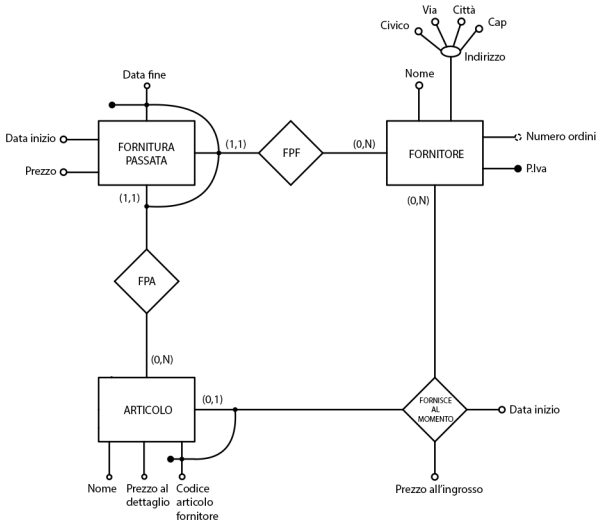


Figura 4: Sottoschema concettuale del rapporto tra gli articoli e i loro fornitori.

2.1.5 Articoli e ordini

Abbiamo deciso di modellare lo schema E-R riguardante gli ordini con una relazione *many-to-many* tra l'entità *ORDINE* e l'entità *ARTICOLO* caratterizzata dalla quantità di ciascun articolo. Tale relazione presenta partecipazione obbligatoria da entrambi i lati in quanto un articolo per essere presente nella base di dati deve essere stato ordinato almeno una volta e un ordine vuoto non avrebbe senso. La parte più delicata di questo schema è la decisione (o meno) di legare mediante una relazione *ORDINE* a *FORNITORE*. Al fine di evitare cicli incoerenti e informazioni ridondanti, abbiamo deciso di includere la data di un ordine all'interno dell'insieme dei suoi attributi, per permettere di utilizzare le entità *FORNITORE* o *FORNITURA PASSATA* per ottenere le informazioni rispetto ad esso. Per garantire che ciascun articolo presente in un ordine appartenga allo stesso fornitore corrente, abbiamo imposto due vincoli aziendali. Inoltre, ciascun ordine è caratterizzato univocamente da un *Codice Ordine*.

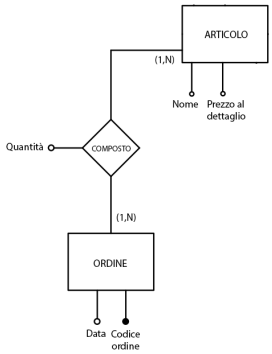


Figura 5: Sottoschema concettuale sul rapporto tra gli articoli e i loro ordini.

2.2 Schema concettuale

Integrando i precedenti sottoschemi ed esplicitando i vincoli aziendali posti su di essi, otteniamo lo schema E-R complessivo per la base di dati. Con questo documento possiamo quindi iniziare la prossima fase di progettazione: la ristrutturazione logica.

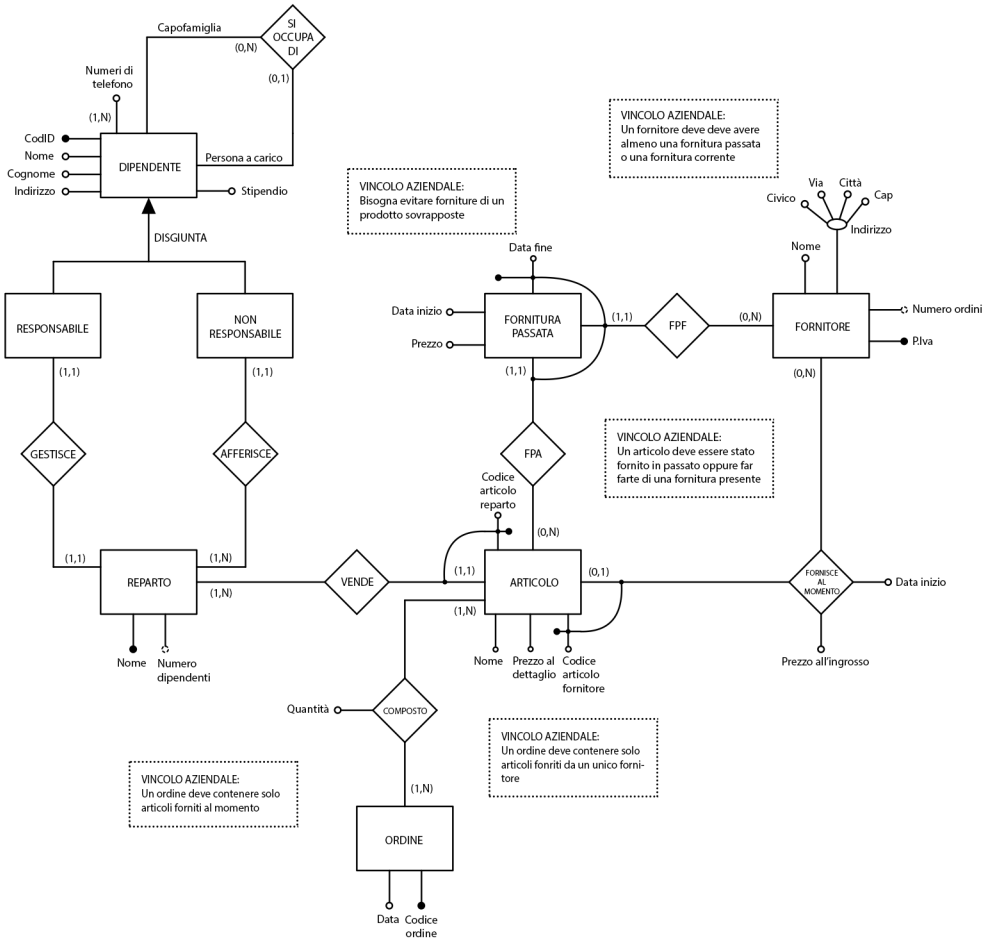


Figura 6: Schema concettuale nel modello Entità Relazioni.

3 Progettazione logica

3.1 Tabella dei volumi

Per valutare i costi delle operazioni, per effettuare una corretta analisi delle ridondanze e quindi per effettuare una corretta ristrutturazione dello schema E-R si rende necessaria una tabella dei volumi in cui per ogni oggetto stimiamo la sua quantità media presente nella base di dati. I volumi di alcuni oggetti sono costanti o poco variabili mentre altri crescono nel tempo; per questi ultimi stimiamo una durata della nostra base di dati 15 anni e le quantità che sono riportate nella tabella 2 saranno le quantità stimate a metà del ciclo di vita (ossia tra i 7 e gli 8 anni).

Concetto	**Tipo**	**Volume**
DIPENDENTE	E	200
SI OCCUPA DI	R	10
RESPONSABILE	E	10
GESTISCE	R	10
NON RESPONSABILE	E	190
AFFERISCE	R	190
REPARTO	E	10
VENDE	R	2500
ARTICOLO	E	2500
ORDINE	E	55000
COMPOSTO	R	275000
FPA	R	3000
FORNITURA PASSATA	E	3000
FPF	R	3000
FORNITORE	E	220
FORNISCE AL MOMENTO	R	1000

Tabella 2: Tabella dei volumi.

La tabella dei volumi (tabella 2) è stata ricavata dalle seguenti considerazioni: Nel supermercato avvengono circa 50 assunzioni/licenziamenti all'anno ma in media troviamo 200 *DIPENDENTI*.[1] Il supermercato è composto da 10 *REPARTI* e siccome ogni reparto è gestito da un solo manager ricaviamo direttamente 10 *RESPONSABILI* e 190 *NON RESPONSABILI*. In ogni istante ci sono circa 10 coppie <*DIPENDENTE, DIPENDENTE*> nella relazione *SI OCCUPA DI*. Istantaneamente sono venduti in media 1000 *ARTICOLI*, tenendo traccia degli articoli venduti in passato e considerando che in media tutti i prodotti cambiano (ossia un articolo x non viene più rifornito e al suo posto viene introdotto un articolo y che lo sostituisce) nell'arco di 5 anni, ricaviamo che a fine del ciclo di vita avremo 4000 articoli e a metà ne avremo 2500 in media come raffigurato nella figura 7a.

Ogni *ARTICOLO* è venduto da uno e un solo reparto quindi la cardinalità della relazione *VENDE* varrà 2500 a metà del ciclo di vita [2]. In media vengono effettuati 20 ordini al giorno, quindi circa 7300 ordini all'anno e 55000 ordini a metà del ciclo di vita come mostrato in figura 7b.

Un *ORDINE* è in media composto da 5 articoli quindi la relazione *COMPOSTO* avrà 275000 coppie <*ARTICOLO, ORDINE*> a metà del ciclo di vita. In media ci sono 100 fornitori correnti ciascuno dei quali fornisce in media 10 articoli da cui ricaviamo che la cardinalità della relazione

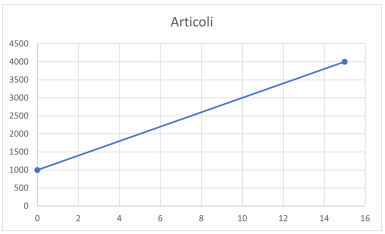
[1] Nnella base di dati teniamo traccia solo dei dipendenti che lavorano attualmente per il supermercato.

[2] Anche gli articoli per cui non esiste una fornitura corrente sono legati al reparto.

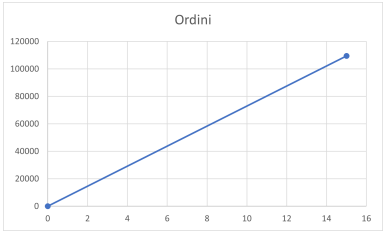
FORNISCE AL MOMENTO è 1000 [3]. Sappiamo che ogni 5 anni abbiamo 1000 nuovi articoli introdotti nella base di dati, di questi il 20% viene fornito da un nuovo fornitore quindi abbiamo approssimativamente 20 nuovi fornitori (ciascuno dei quali fornisce in media 10 articoli) ogni 5 anni. Inoltre in media un articolo durante i suoi 5 anni di vendita presso il negozio, cambia fornitore due volte e una di queste coinvolge un nuovo fornitore, quindi abbiamo 100 nuovi fornitori ogni 5 anni. In totale partendo da 100 fornitori abbiamo un incremento di 120 fornitori ogni 5 anni e a metà del ciclo di vita abbiamo 220 fornitori (vedere figura 7c).

All'anno 0 abbiamo chiaramente 0 *FORNITURE PASSATE* e ogni articolo cambia fornitura 2 volte in 5 anni, quindi ogni 5 anni ci sono 2000 nuove forniture passate e a metà del ciclo di vita ne avremo 3000 (vedere figura 7d).

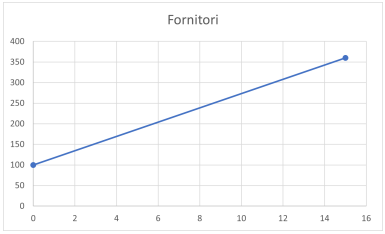
Segue che anche le relazioni *FPF* e *FPA* hanno cardinalità 3000 a metà del ciclo di vita.



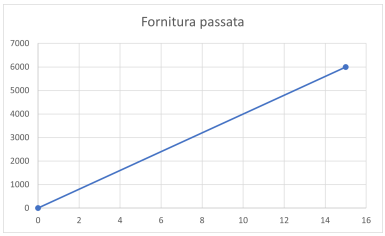
(a) Entità *ARTICOLO*.



(b) Entità *ORDINE*.



(c) Entità *FORNITORE*.



(d) Entità *FORNITURA PASSATA*.

Figura 7: Crescita delle istanze durante il ciclo di vita.

3.2 Analisi delle ridondanze

3.2.1 Numero di ordini

In questa sezione verrà discusso se mantenere o meno l'attributo ridondante *Numero d'Ordini* nell'entità *FORNITORE*. Cominciamo col definire i task che coinvolgono tale attributo:

- **Task 1**: Inserimento di un nuovo ordine.
- **Task 2**: Dato un fornitore, contare il numero totale d'ordini [4] che il supermercato ha effettuato verso tale fornitore.

In figura 8 viene focalizzata la porzione di schema concettuale che riguarda l'attributo e i relativi task.

[3] Anche se a metà del ciclo di vita abbiamo 2500 articoli (ciascuno dei quali è legato al reparto) non tutti fanno parte di una fornitura corrente.

[4] Parteciperanno al conto sia ordini della fornitura corrente e sia ordini delle forniture passate.

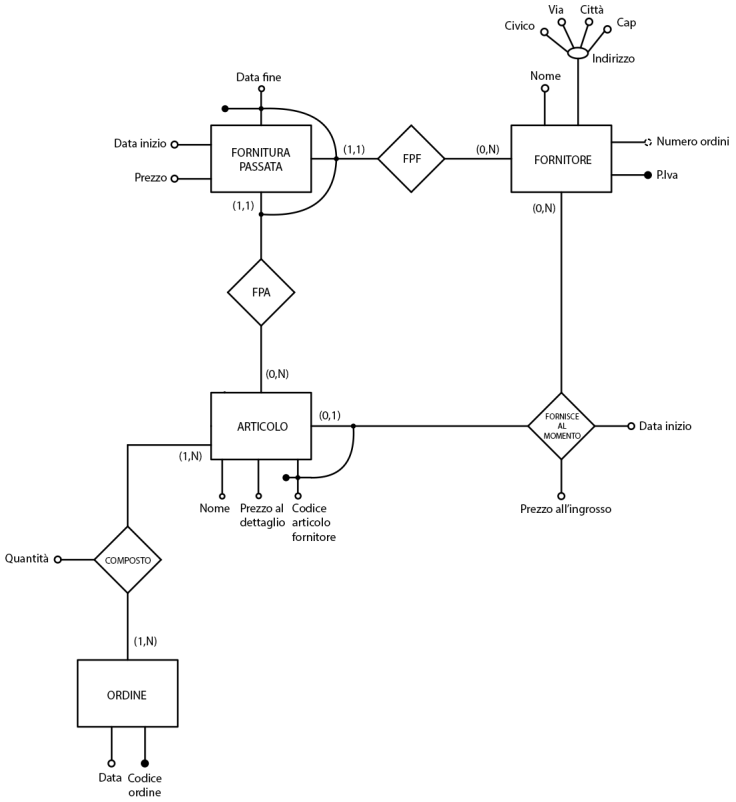


Figura 8: Schema concettuale - Attributo ridondante *numero d'ordini*

Per quanto riguarda le frequenze (sintetizzate in tabella 3) il task 1 viene effettuato 20 volte al giorno mentre il task 2 viene effettuato 2 volte al mese per ogni fornitore che ha una fornitura attiva, quindi in totale 200 volte al mese.

Task	**Tipo**	**Frequenza [Volte al mese]**
Task 1	Interactive	600
Task 2	Batch	200

Tabella 3: Frequenza dei task.

Accessi in assenza della ridondanza

Task 1. In assenza della ridondanza l'inserimento di un ordine (che in media ha 5 articoli [5]) comporta solo una scrittura in *ORDINE*, 5 scritture in *COMPOSTO* e 5 letture in *ORDINE* (vedere tabella 4).

Oggetto	**Tipo**	**Accessi**	**Tipologia Accessi**
ORDINE	E	1	W
COMPOSTO	R	5	W
ARTICOLO	E	5	R

Tabella 4: Accessi *Task 1* in assenza della ridondanza.

Task 2. Essendo il task 2 complesso mostriamo la tabella degli accessi per 2 modi di implementarlo;

[5] Verificabile nella tabella dei volumi.

la decisione sulla ridondanza verrà poi presa considerando la tabella degli accessi relativa al modo più efficiente di realizzare tale operazione. Dapprima partiamo dall'entità *ORDINE* per arrivare all'entità *FORNITORE*. Scandiamo tutte le istanze dell'entità *ORDINE* (che sono in media 55000 come indicato in tabella 2) e per ogni ordine, ricordando che un ordine è composto solo da articoli forniti dallo stesso fornitore, ricaviamo una (e una sola) coppia <*ORDINE, ARTICOLO*> [6] dalle istanze della relazione *COMPOSTO*. Quindi 55000 letture in *ORDINE* e 55000 letture in *COMPOSTO*. Sia ora α la probabilità che un ordine relativo al fornitore f (rappresentato dall'articolo e il timestamp al momento dell'inserimento d'ordine) sia relativo ad una fornitura presente (ci preoccuperemo in seguito di come calcolare tale coefficiente). Per ogni coppia <*ARTICOLO, data*> dobbiamo verificare se viene fornito correntemente, per fare ciò facciamo un accesso in lettura alla relazione *FORNISCE AL MOMENTO* e otteniamo la entry (se presente) identificata da <*ARTICOLO, FORNITORE*> e verifichiamo che l'attributo *Data inizio* sia minore della data dell'ordine. Se quest'ultima condizione non è verificata o se non è stata trovata nessuna entry, ci troviamo nella situazione (che avviene con probabilità $1-\alpha$) in cui l'ordine è relativo ad una fornitura passata. Accediamo quindi a tutte le istanze della relazione *FPA* in cui troviamo l'articolo (che sono in media $\frac{|FPA|}{|ARTICOLO|} = \frac{3000}{2500}$) e quindi accediamo lo stesso numero di volte all'entità *FORNITURA PASSATA* controlliamo che la data sia consistente (questo deve avvenire per esattamente una fornitura passata legata all'articolo rappresentativo del nostro ordine). Per la *FORNITURA PASSATA* la cui data risulta consistente controlliamo che il fornitore[7] sia proprio f [8].

Quindi il numero di accessi totali (sintetizzati in tabella 5a) vale:

$$55000 + 55000 + 55000(1 + (1-\alpha)(\frac{3000}{2500} + \frac{3000}{2500})) \tag{1}$$

Stimiamo $\alpha = \frac{|FORNISCE\ AL\ MOMENTO|}{|FORNISCE\ AL\ MOMENTO| + |FORNITURA\ PASSATA|} = \frac{1000}{4000}$

Proviamo ora il percorso opposto, partendo da *FORNITORE* e arrivando a *ORDINE*: conoscendo il fornitore, bisogna sommare il numero di accessi a *COMPOSTO* e *ORDINE* ricavati dagli articoli forniti correntemente al numero di accessi ricavati da forniture passate.
Questo significa che si può ricondurre il calcolo degli accessi a due percorsi nello schema E-R:

- <*FORNISCE AL MOMENTO, COMPOSTO, ORDINE* >, eseguito in media γ volte.
- <*FORNITURA PASSATA, COMPOSTO, ORDINE* >, eseguito in media δ volte.

dove γ e δ rappresentano il numero medio di forniture correnti e passate di ogni fornitore:

$$\gamma = \frac{Fornitori\ che\ hanno\ forniture\ correnti}{Fornitori\ totali} = \frac{100}{220}$$
$$\delta = \frac{Fornitori\ totali\ -\ Fornitori\ correnti}{Fornitori\ totali} = \frac{120}{220} = 1 - \gamma$$

Dato che in media un fornitore corrente fornisce 10 articoli e che un articolo compare mediamente in 110 ordini[9], il numero di accessi per il primo percorso sarà:

- $\lceil \gamma \cdot 10 \rceil = 5$ per *FORNISCE AL MOMENTO*
- $\lceil \gamma \cdot 10 \cdot\ 110 \rceil = 500$ per *COMPOSTO* e per *ORDINE*

[6] L'articolo e la data d'ordine rappresentano (ossia identificano univocamente) l'intero ordine. Fissato l'ordine la scelta della coppia può avvenire in modo arbitrario in quanto siamo interessati al fornitore, e ciascun articolo a presente nell'ordine o è relativo allo stesso fornitore f il quale nel momento d'inserimento ordine forniva ciascun articolo a.

[7] Essendo *FORNITURA PASSATA* entità debole rispetto a *Fornitore*, accedendo all'entità, ottetiamo anche la chiave di fornitore.

[8] Se non è *f*, significa che l'ordine (rappresentato dalla coppia <*ARTICOLO, data*>) non contribuirà alla quantità *Numero ordini* per il fornitore *f*.

[9] $Numero\ medio\ di\ ordini\ per\ articolo = \frac{(Numero\ medio\ di\ articoli\ per\ ordine)\ \cdot\ (Numero\ totale\ di\ ordini)}{Numero\ totale\ di\ articoli} = 110$

Analogamente, per il secondo percorso si ha che mediamente il numero di forniture passate per ogni fornitore è 13,6 [10] e di conseguenza il numero di accessi è:

- $\lceil \delta \cdot 13,6 \rceil = 8$ per *FORNITURA PASSATA*
- $\lceil \delta \cdot 13,6 \cdot \ 110 \rceil = 816$ per *COMPOSTO* e per *ORDINE*

Il numero totale di accessi è contenuto nella tabella 5b.

(a) Percorso da *ORDINE* a *FORNITORE*.

Oggetto	**Tipo**	**Accessi**	**Tipologia Accessi**
ORDINE	E	55000	R
COMPOSTO	R	55000	R
FORNISCE AL MOMENTO	R	55000	R
FORNITURA PASSATA	E	49500	R
FORNITORE	E	1	R

(b) Percorso da *FORNITORE* a *ORDINE*.

Oggetto	**Tipo**	**Accessi**	**Tipologia Accessi**
FORNITORE	E	1	R
FORNISCE AL MOMENTO	R	5	R
FORNITURA PASSATA	E	8	R
COMPOSTO	R	1316	R
ORDINE	E	1316	R

Tabella 5: Accessi *task 2* in assenza della ridondanza.

Accessi in presenza della ridondanza In presenza della ridondanza ad ogni inserimento di un ordine dobbiamo accedere (tramite una sequenza di diversi accessi ad entità/relazioni) al fornitore che fornisce correntemente tutti gli articoli dell'ordine, leggere il valore dell'attributo *Numero d'Ordini* e incrementarlo di 1. In tabella 6 sono specificati tutti gli accessi necessari. Per quanto riguarda il task 2, dato un fornitore f ci basta semplicemente accedere all'attributo *Numero d'Ordini*. Vedere la tabella 7.

Oggetto	**Tipo**	**Accessi**	**Tipologia Accessi**
ORDINE	E	1	W
COMPOSTO	R	5	W
ARTICOLO	E	5	R
FORNISCE AL MOMENTO	R	1	R
FORNITORE	E	1	W

Tabella 6: Accessi *task 1* in presenza della ridondanza.

Oggetto	**Tipo**	**Accessi**	**Tipologia Accessi**
FORNITORE	E	1	R

Tabella 7: Accessi *task 2* in presenza della ridondanza.

[10] $\frac{|FORNITURA\ PASSATA|}{|FORNITORE|} = \frac{3000}{220}$.

Decisione sulla ridondanza Teniamo in considerazione della tabella 5b per quanto riguarda il task 2 in assenza di ridondanze, infatti come si può facilmente vedere dal confronto tra le tabelle 5a e 5b il secondo modo di operare (ossia partendo da *FORNITORE* per arrivare a *ORDINE*) risulta molto più efficente. Usando la proporzione *1w = 2r* trasformiamo tutti gli accessi in accessi in lettura e facendo delle opportune somme otteniamo la tabella 8.

Task	**Con ridondanza**	**Senza ridondanza**
Task 1	12000	10200
Task 2	200	529000
Totale	12200	539200

Tabella 8: Accessi totali per task.

Quindi mantenendo la ridondanza, a fronte del task 1 che diventa del 18% più inefficente (un overhead comunque decisamente accettabile considerando che ad ogni inserimento di ordine verranno effetuati 20 accessi anzichè 17 e quindi l'ordine di grandezza rimane lo stesso), il task 2 diventa circa 2700 volte meno costoso. Si decide quindi di mantenere la ridondanza *Numero d'Ordini*.

3.2.2 Numero di dipendenti

Un ulteriore attributo ridondante all'interno dello schema E-R di questa base di dati consiste in *Numero Dipendenti* all'interno di *REPARTO*. È quindi necessario effettuare un'ulteriore analisi delle ridondanze, a partire dai task che agiscono su tale attributo:

- **Task 1**: Dato un reparto, fornire il numero di dipendenti che vi lavorano.
- **Task 2**: Licenziamento o assunzione di un dipendente.

Il primo task è necessario (virtualmente) per alcune statistiche sugli stipendi, per cui verrà eseguita per ogni reparto una volta al mese, con una doppia esecuzione a fine anno (per un totale di 130 volte all'anno). Per il secondo task, invece, abbiamo immaginato che il numero di dipendenti rimanga circa costante nel tempo, tuttavia vi sia un *turnover* di circa 50 persone l'anno.

In figura 9 viene visualizzata la sezione di schema E-R riguardante l'attributo e i relativi task.

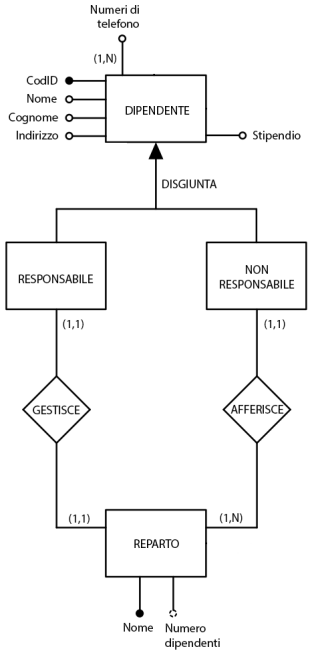


Figura 9: Schema concettuale - Attributo ridondante *Numero Dipendenti*

Task	**Tipo**	**Frequenza [Volte all'anno]**
Task 1	Batch	130
Task 2	Interactive	50

Tabella 9: Frequenza dei task.

Accessi in assenza della ridondanza

Task 1. Dato un reparto, sappiamo che avrà necessariamente un responsabile che vi lavora, per cui in assenza di ridondanza è sufficiente contare i dipendenti che vi afferiscono effettuando una scansione di tutta la relazione *AFFERISCE*, popolata in media da 190 istanze. Di conseguenza, nel complesso, il task 1 richiede 190 letture, ossia 190 accessi in memoria [11]. Questo task avviene 13 volte all'anno per ciascuno dei 10 reparti, per cui il costo complessivo è di

$$190 \cdot 13 \cdot 10 = 24700 \text{ accessi in memoria all'anno} \tag{2}$$

Oggetto	**Tipo**	**Accessi**	**Tipologia Accessi**
AFFERISCE	R	190	R

Tabella 10: Accessi *Task 1* in assenza di ridondanza.

Task 2. L'assunzione (o il licenziamento) di un dipendente richiede un'operazione di scrittura all'interno di una delle specializzazioni dell'entità *DIPENDENTE*, e un'ulteriore operazione di scrittura per aggiornare la rispettiva relazione con *REPARTO*. Quindi vengono fatte 2 scritture, per un totale di 4 accessi in memoria. Siccome quest'operazione avviene 50 volte all'anno, abbiamo un costo annuale corrispondente a

[11] Abbiamo escluso l'accesso alla relazione *GESTISCE* in quanto definiremo in futuro una *User Defined Function* per incrementare di 1 il conto di dipendenti, dato che tra questi dobbiamo contare anche il responsabile del reparto.

$$50 \cdot 4 = 200 \text{ accessi in memoria all'anno} \tag{3}$$

Oggetto	Tipo	Accessi	Tipologia Accessi
RESPONSABILE	E	1	W
GESTISCE	R	1	W

Tabella 11: Accessi *Task 2* in assenza di ridondanza nel caso dell'inserimento di un nuovo responsabile.

Oggetto	Tipo	Accessi	Tipologia Accessi
NON_RESPONSABILE	E	1	W
AFFERISCE	R	1	W

Tabella 12: Accessi *Task 2* in assenza di ridondanza nel caso dell'inserimento di un nuovo impiegato.

Accessi in presenza della ridondanza

Task 1. Dato un reparto, per contare il suo numero di dipendenti è sufficiente accedere alla sua istanza all'interno dell'entità *REPARTO* e leggere il valore dell'attributo ridondante. Questo richiede chiaramente una sola operazione di lettura, per cui tenendo conto della tabella delle frequenze e di quella dei volumi, il costo complessivo è di

$$1 \cdot 13 \cdot 10 = 130 \text{ accessi in memoria all'anno} \tag{4}$$

Oggetto	Tipo	Accessi	Tipologia Accessi
REPARTO	E	1	R

Tabella 13: Accessi *Task 1* in presenza della ridondanza.

Task 2. Al contrario del task precedente, la ridondanza aumenta la complessità dell'assunzione (o licenziamento) di un dipendente, in quanto è necessario mantenere aggiornato l'attributo ridondante all'interno dell'istanza relativa di *REPARTO*. Di conseguenza, oltre alle 2 scritture calcolate nel caso senza ridondanza, ne va aggiunta una terza a ogni esecuzione del task, per un totale di 6 accessi in memoria. Di conseguenza, il costo complessivo sarà di

$$50 \cdot 6 = 300 \text{ accessi in memoria all'anno} \tag{5}$$

Oggetto	Tipo	Accessi	Tipologia Accessi
RESPONSABILE	E	1	W
GESTISCE	R	1	W
REPARTO	E	1	W

Tabella 14: Accessi *Task 2* in presenza di ridondanza nel caso dell'inserimento di un nuovo responsabile.

Oggetto	Tipo	Accessi	Tipologia Accessi
NON_RESPONSABILE	E	1	W
AFFERISCE	R	1	W
REPARTO	E	1	W

Tabella 15: Accessi *Task 2* in presenza di ridondanza nel caso dell'inserimento di un nuovo impiegato.

Decisione sulla ridondanza In questo caso è bene mantenere la ridondanza in quanto il numero di accessi in memoria diminuisce drasticamente in presenza dell'attributo aggiuntivo. Se le somme dei due casi fossero state più vicine tra loro, invece, sarebbe stato più opportuno scegliere di eliminare *Numero Dipendenti* in quanto quest'ultimo beneficia un task batch interferendo con la performance di un'operazione interattiva, molto più influente in questo tipo di analisi in quanto rischierebbe di aggiungere carico al sistema in un istante già saturato.

Task	Con ridondanza	Senza ridondanza
Task 1	130	24700
Task 2	300	200
Totale	430	24900

Tabella 16: Accessi totali per operazione.

3.3 Rimozione delle generalizzazioni

Al fine di ristrutturare lo schema E-R per proseguire con la fase di progettazione logica, è necessario rimuovere la generalizzazione introdotta per i tipi di dipendenti. Esistono 3 pattern di ristrutturazione per far ciò: rimuovere i figli, rimuovere il genitore e mantenere tutte le entità sostituendo i costrutti delle specializzazioni con relazioni ad hoc.

Si opta per mantenere l'entità *DIPENDENTE* e rimuovere i figli, in quanto le operazioni individuate non sfruttano la partizione in *RESPONSABILE* e *NON RESPONSABILE*. Inoltre, siccome tutti gli attributi dei figli sono attributi ereditati dall'entità genitore, non si ha spreco di memoria causato dai valori *NULL*. Per garantire la consistenza delle informazioni nella base di dati, tuttavia, è necessario introdurre un vincolo aziendale aggiuntivo che verifichi che un dipendente o afferisca o diriga un determinato reparto. Se non facessimo ciò, si avrebbe un'alterazione delle assunzioni iniziali che prevedevano che la specializzazione fosse disgiunta.

Adottare il secondo pattern di progetto sarebbe scorretto in questo caso in quanto la relazione ricorsiva *SI OCCUPA DI* non sarebbe più rappresentabile nello schema E-R, né implementabile successivamente se non adottando dubbie strategie di definizione dei constraints.

Il terzo pattern non sarebbe stato necessariamente scorretto in questa evenienza, tuttavia avrebbe introdotto molta ridondanza nelle informazioni dei dipendenti, senza fornire un vero beneficio all'implementazione: anche questa soluzione dovrebbe infatti prevedere un controllo sulla mutua esclusione dei dipendenti all'interno delle due partizioni.

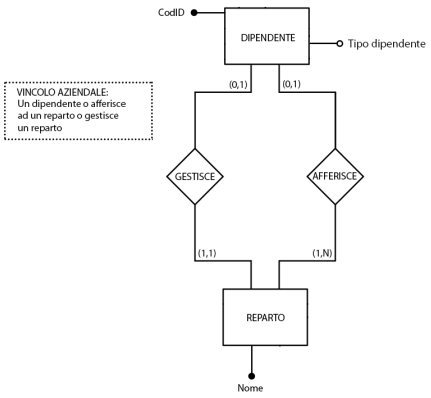


Figura 10: Schema concettuale dopo la rimozione della generalizzazione

3.4 Rimozione degli attributi multivalore

L'unica entità in cui è presente un attributo multivalore è l'entità *DIPENDENTE*. In questo caso è sufficente introdure una nuova entità *NUMERO DI TELEFONO* e legarla con una relazione *POSSIEDE* a *DIPENDENTE*. La figura 11 mostra il cambiamento effettuato.

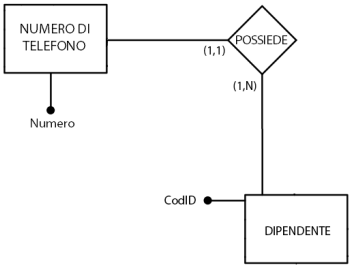


Figura 11: Rimozione dell'attributo multivalore *Numeri di telefono*.

3.5 Selezione delle chiavi primarie

Per le entità in cui abbiamo un unica chiave candidata nello schema concettuale, questa diventerà la chiave primaria nello schema logico. Rimane da considerare l'entità *ARTICOLO* che ha due chiavi candidate come mostrato in figura 12.

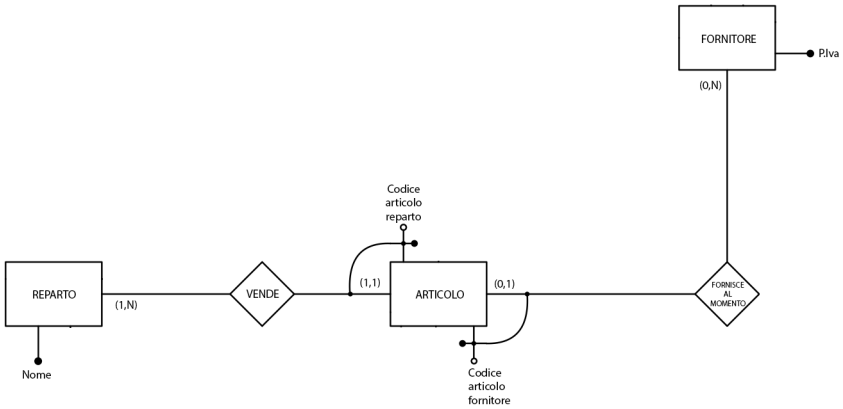


Figura 12: Chiavi candidate dell'entità *ARTICOLO*.

In questo caso l'unica scelta ragionevole sia identificare l'entità *ARTICOLO* dall'attributo *Codice articolo reparto* insieme all'identificativo dell'entità *REPARTO*. Possiamo evidenziare 3 motivi per cui identificare ARTICOLO con la coppia <*Codice articolo reparto*, *FORNITORE*> **non** sia una buona scelta.

- **Partecipazione alla relazione non obbligatoria.** Come prima cosa notiamo che fissata un'istanza di *ARTICOLO*, questa non deve per forza comparire in una coppia della relazione *FORNISCE AL MOMENTO* (potrebbe trattarsi di un articolo che non viene più fornito correntemente da un fornitore).
- **Non omogeneità tra articoli.** Il dominio è quello di un supermercato e all'interno del supermercato stesso sarebbe scomodo utilizzare degli identificativi definiti dal *FORNITORE*: Fornitori diversi infatti potrebbero usare codifiche e numerazioni diverse. Risulta quindi chiaro che un codice definito dal supermercato ci permette di avere lo stesso dominio (tra articoli) nell'attributo chiave.
- **Discontinuità nell'identificazione dello stesso articolo.** Consideriamo un'articolo a che viene venduto per un certo periodo da un fornitore f'. Se in un certo momento il supermercato interrompe la fornitura da f' e si fa fornire lo stesso articolo a da un nuovo fornitore f'', l'articolo pur rimanendo lo stesso si vedrebbe cambiare parte della chiave e quindi ad esempio tutte le coppie della relazione *COMPOSTO* in cui compare a dovrebbero essere aggiornate. Più in generale, al momento del cambio di fornitura di un certo articolo a, tutte le tuple presenti nella basi di dati in cui compariva la vecchia chiave di a andrebbero aggiornate e questo oltre a provocare inutili costi computazionali aggiuntivi, può portare anche a inconsistenze nella basi dati stessa.

3.6 Schema E-R ristrutturato

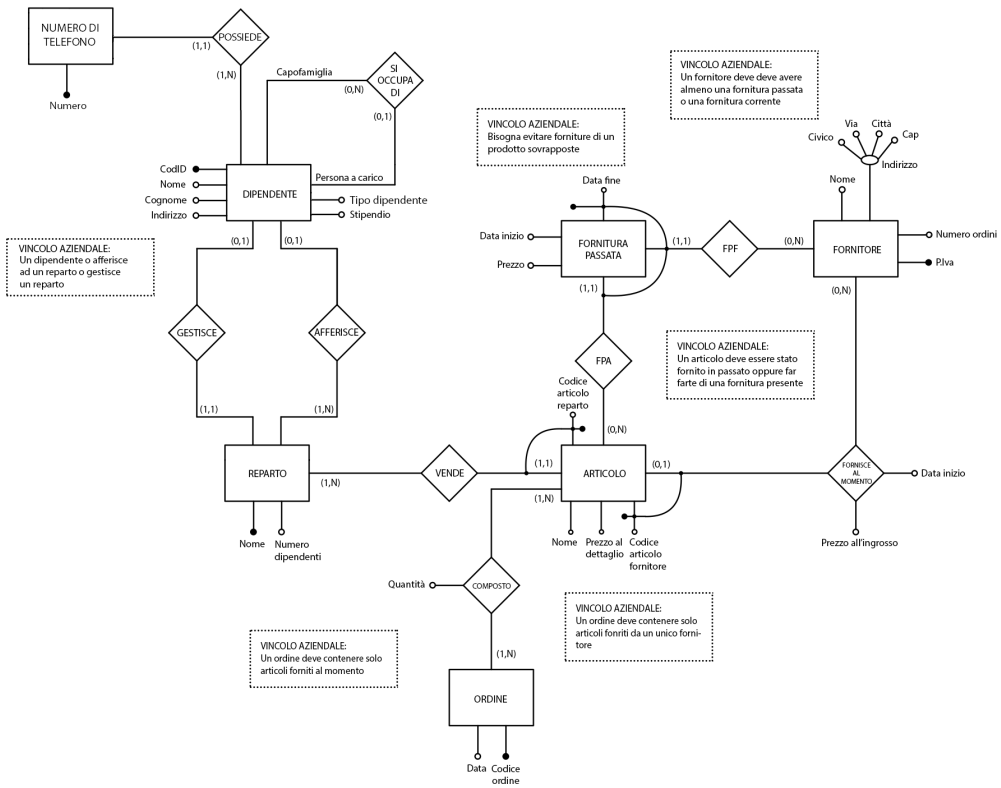


Figura 13: Schema Entità Relazioni ristrutturato.

3.7 Schema logico

Schema relazionale

- NUMERO_DI_TELEFONO(Numero, *CodID*)
  - VNN:{CodID}
- DIPENDENTE(CodID, Nome, Cognome, Indirizzo, Stipendio, TipoDipendente, *CodIDCapoFamiglia*, *Reparto*)
  - Si impone VNN su tutti gli attributi tranne che su CodIDCapoFamiglia e su Reparto.
- REPARTO(Nome, NumeroDipendenti, *Manager*)
  - VNN:{Manager}
  - VUN:{Manager}
- ARTICOLO(CodiceArticoloReparto, *Reparto*, Nome, PrezzoAlDettaglio, CodiceArticoloFornitore, *Fornitore*, DataInizio, PrezzoIngrosso)
  - Si impone VNN su tutti gli attributi tranne che su Fornitore.
- FORNITORE(PartitaIva, Nome, Indirizzo, NumeroOrdini)
- FORNITURA_PASSATA(*CodiceArticolo*, *Reparto*, *Fornitore*, DataFine, DataInizio, PrezzoFornitura)
- COMPOSTO(*Ordine*, *CodiceArticolo*, *Reparto*, Quanità)
- ORDINE(CodiceOrdine, Dataordine)

Legenda: le chiavi primarie sono sottolineate, le chiavi esterne sono in corsivo, VNN indica il vincolo not null, VUN indica il vincolo di unicità.

Chiavi esterne

- *CodID* è chiave esterna di NUMERO_DI_TELEFONO rispetto a DIPENDENTE
- *CodIDCapoFamiglia* è chiave esterna di DIPENDENTE rispetto a DIPENDENTE
- *Reparto* è chiave esterna di DIPENDENTE rispetto a REPARTO
- *Manager* è chiave esterna di REPARTO rispetto a DIPENDENTE
- *Reparto* è chiave esterna di ARTICOLO rispetto a REPARTO
- *Fornitore* è chiave esterna di ARTICOLO rispetto a FORNITORE
- (*CodiceArticolo, Reparto*) è chiave esterna di FORNITURA_PASSATA rispetto a ARTICOLO
- *Fornitore* è chiave esterna di FORNITURA_PASSATA rispetto a FORNITORE
- (*CodiceArticolo, Reparto*) è chiave esterna di COMPOSTO rispetto a ARTICOLO
- *Ordine* è chiave esterna di COMPOSTO rispetto a ORDINE

Vincoli aziendali

Dalla traduzione da schema concettuale a schema logico, alcuni vincoli legati alle cardinalità delle relazioni non sono stati esprimibili nel formalismo del modello relazionale. Dobbiamo quindi imporre i seguenti vincoli aggiuntivi che andranno implementati con dei meccanismi esterni come i trigger:

- Un dipendente deve possedere almeno un numero di telefono.
- Ad un reparto deve afferire almeno un dipendente.
- Un reparto deve vendere almeno un articolo.
- Un ordine è composto di almeno un articolo.
- Un articolo compare in almeno un ordine.

Oltre a questi rimangono i vincoli aziendali identificati fin dall'inizio:

- Un dipendente o afferisce ad un reparto o gestisce un reparto.
- Un ordine deve contenere solo articoli forniti al momento.
- Un ordine deve contenere solo articoli forniti da un unico fornitore.
- Un articolo deve essere stato fornito in passato oppure far parte di una fornitura presente.
- Bisogna evitare forniture di un prodotto sovrapposte.
- Un fornitore deve avere almeno una fornitura passata o una fornitura corrente.

4 Progettazione fisica

4.1 Definizione delle relazioni in SQL

Di seguito viene presentato lo schema relazionale in Data Definition Language.

```
create domain dom_CodID as char(16); --e' il codice fiscale dei dipendenti
create domain dom_NumeroTelefono varchar(13); --include il prefisso +xx
create domain dom_CodOrdine as char(13);
create domain dom_NomeReparto varchar(30);
create domain dom_NomeArticolo varchar(50);
create domain dom_CodArticolo varchar(15);
create domain dom_CodArticoloFornitore varchar(60); --e' grande per gestire la
    eterogeneita' dei vari codici assegnati dai fornitori
create domain dom_PartitaIva varchar(20);
create domain dom_TipoDipendente varchar(10)
    DEFAULT 'Dipendente'
    CONSTRAINT valoreTipoDipendete
    CHECK(value = 'Manager' or value = 'Dipendente');

create table Dipendente(
    CodID dom_CodID,
    Nome varchar(50) not null,
    Cognome varchar(50) not null,
    Indirizzo varchar(100) not null,
    Stipendio integer not null,
    TipoDipendente dom_TipoDipendente not null,
    CodIDCapoFamiglia dom_CodID,
    Reparto dom_NomeReparto,

    PRIMARY KEY(CodID),
    CHECK (Stipendio > 0),
    CONSTRAINT FkCapoFamiglia FOREIGN KEY(CodIDCapoFamiglia)
        references Dipendente(CodID)
        ON UPDATE CASCADE
        ON DELETE NO ACTION
);

create table NumeroDiTelefono(
    Numero dom_NumeroTelefono,
    Proprietario dom_CodID not null,

    PRIMARY KEY(Numero),
    CONSTRAINT FkProprietario FOREIGN KEY(Proprietario)
        references Dipendente(CodID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

create table Reparto(
    Nome dom_NomeReparto,
    NumeroDipendenti integer not null,
```

```
    Manager dom_CodID not null unique,

    PRIMARY KEY(Nome),
    CONSTRAINT FkManager FOREIGN KEY(Manager)
        references Dipendente(CodID)
        ON UPDATE CASCADE
        ON DELETE NO ACTION
);

alter table Dipendente ADD CONSTRAINT FkReparto
    FOREIGN KEY(Reparto)
    references Reparto(Nome)
    ON UPDATE CASCADE
    ON DELETE NO ACTION;

create table Fornitore(
    PartitaIva dom_PartitaIva,
    Nome varchar(30) not null,
    Indirizzo varchar(50) not null,
    NumeroOrdini integer not null,

    PRIMARY KEY (PartitaIva),
    CHECK (NumeroOrdini >= 0)
);

create table Articolo(
    CodiceArticoloReparto dom_CodArticolo,
    Reparto dom_NomeReparto not null,
    Nome dom_NomeArticolo not null,
    PrezzoAlDettaglio decimal(5,2) not null,
    CodiceArticoloFornitore dom_CodArticoloFornitore not null,
    Fornitore dom_PartitaIva,
    DataInizio date not null,
    PrezzoIngrosso decimal(5,2) not null ,

    PRIMARY KEY(CodiceArticoloReparto,Reparto),
    CHECK (PrezzoAlDettaglio > 0),
    CHECK (PrezzoIngrosso > 0),
    CONSTRAINT FkArticolo FOREIGN KEY(Reparto)
        references Reparto(Nome)
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT FkFornitore FOREIGN KEY(Fornitore)
        references Fornitore(PartitaIva)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);

create table ForniturаPassata(
    CodiceArticolo dom_CodArticolo,
    Reparto dom_NomeReparto,
```

```
    Fornitore dom_PartitaIva,
    DataFine date,
    DataInizio date not null,
    PrezzoFornitura decimal(5,2) not null,

    PRIMARY KEY(CodiceArticolo,Reparto,Fornitore,DataFine),
    CHECK (PrezzoFornitura > 0),
    CHECK (DataFine > DataInizio),
    CONSTRAINT FkFornituraPassataArticolo FOREIGN KEY(CodiceArticolo, Reparto)
        references Articolo(CodiceArticoloReparto, Reparto)
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT FkFornituraPassataFornitore FOREIGN KEY(Fornitore)
        references Fornitore(PartitaIva)
        ON UPDATE CASCADE
        ON DELETE NO ACTION
);

create table Ordine(
    CodiceOrdine dom_CodOrdine,
    DataOrdine date not null,

    PRIMARY KEY(CodiceOrdine)
);

create table Composto(
    Ordine dom_CodOrdine,
    CodiceArticolo dom_CodArticolo,
    Reparto dom_NomeReparto,
    Quantita integer not null,

    PRIMARY KEY(Ordine,CodiceArticolo,Reparto),
    CHECK (Quantita > 0),
    CONSTRAINT FkCompostoOrdine FOREIGN KEY(Ordine)
        references Ordine(CodiceOrdine)
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT FkCompostoArticolo FOREIGN KEY(CodiceArticolo, Reparto)
        references Articolo(CodiceArticoloReparto, Reparto)
        ON UPDATE CASCADE
        ON DELETE NO ACTION
);
```

4.2 Analisi e scelta degli indici

Per rendere più efficiente l'esecuzione di alcune operazioni effettuate sulla base di dati si è deciso di implementare indici sulle tabelle che presentano un alto numero di accessi. PostgresSQL offre diverse tipologie di indici, tuttavia i B-tree (implementati in Postgres in modo simile ai B+-tree) coprono la maggior parte dei casi d'uso richiesti in questo studio e la loro flessibilità li rende ideali

anche nel caso di definizione di nuove operazioni sulla base di dati. Tenendo conto delle operazioni precedentemente definite, gli indici implementati sono:

- Indice sull'attributo *Fornitore* di ARTICOLO: l'*OPERAZIONE 3* richiede il controllo che il fornitore richiesto abbia almeno una fornitura attiva. Senza indici, questo può richiedere un'intera scansione lineare nel caso peggiore.
- Indice multicolonna su *Articolo, Reparto* in COMPOSTO: per l'*OPERAZIONE 5* è necessario recuperare tutte le occorrenze di una coppia (Articolo, Reparto) in COMPOSTO, per poi controllare la data dell'ordine associato. Poiché tale tabella è la più popolosa di tutta la base di dati, un indice abbatte notevolmente il costo dell'operazione. L'indice deve per forza essere definito su entrambe le colonne perché ARTICOLO è entità debole rispetto a REPARTO.

Le altre operazioni (tra cui in particolare l'*OPERAZIONE 4*, effettuata un numero elevato di volte ogni giorno) accedono alle tuple attraverso le chiavi primarie delle relative tabelle, quindi non richiedono la definizione di ulteriori indici oltre a quelli creati di default da Postgres su tali campi.

```
CREATE INDEX indice_fornitore_in_articolo ON ARTICOLO (Fornitore);

CREATE INDEX indice_articolo_reparto_in_composto ON COMPOSTO (articolo,
   reparto);
```

5 Implementazione

5.1 Popolamento della base di dati

5.1.1 Generazione dei dati

Per creare un *dataset* realistico per la base di dati non basta semplicemente generare valori randomici del tipo corretto, bensì bisogna produrre insiemi di tuple consistenti tra loro e non in conflitto con i vincoli del *database*. Al fine di raggiungere questo scopo, abbiamo scritto un lungo *script* in *Python*, di cui evidenzieremo soltanto alcune caratteristiche chiave per contenere la lunghezza di questa sezione.

Innanzitutto, abbiamo scelto *Python* come linguaggio in quanto fornisce i dizionari (o array associativi) come tipo *built-in*: ciò rende molto facile tenere in ordine strutture dati con organizzazione eterogenea. Oltre a ciò, *Python* presenta varie librerie esterne per la generazione di singoli valori casuali, come *names*, *random_address* e *faker*. Un esempio di questo moduli in uso si può vedere nello snippet successivo, in cui vengono generati i dati dei dipendenti:

```
dipendente = [
    {
        "CodID" : str(i),
        "Nome" : names.get_first_name(),
        "Cognome" : names.get_last_name(),
        "Indirizzo" :
            random_address.real_random_address()["address1"].replace("'", ""),
        "Stipendio" : random.randint(60000,120000) if i<10 else
            random.randint(30000,59999),
        "TipoDipendente" : "Manager" if i<10 else "Dipendente",
        "CodIDCapoFamiglia" : "null",
        "Reparto" : nomiReparti[i%10] if i<20 else
            nomiReparti[random.randint(0,9)]
    }
    for i in range(200)]
```

Tutti i dati sono stati generati in numerosità pari alla corrispondente stima dei volumi a metà del ciclo di vita presentata nella tabella 2. La parte più complessa della procedura è stata ovviamente il corretto popolamento della tabella *ORDINE*, in quanto richiede una particolare attenzione alla variazione temporale delle forniture attive.

Generati correttamente i dati, abbiamo usato il modulo *psycopg2* per collegare il nostro script alla base di dati ed inserire i valori al suo interno. Un esempio dell'uso di questa libreria si può vedere nel seguente snippet, in cui i dipendenti creati randomicamente al passo precedente vengono inseriti all'interno del *database*.

```
conn = psycopg2.connect(
    host = "localhost",
    user = "postgres",
    database = "supermercato",
    password = "#######")
cur = conn.cursor()
cur.execute("SET search_path TO supermercato;")
for dip in dipendente :
    cur.execute(f"""INSERT INTO Dipendente (codid, nome, cognome, indirizzo,
        stipendio, tipodipendente, codidcapofamiglia, reparto) VALUES
        ('{dip["CodID"]}', '{dip["Nome"]}', '{dip["Cognome"]}',
```

```
        '{dip["Indirizzo"]}', {dip["Stipendio"]}, '{dip["TipoDipendente"]}',
        {"'"+dip["CodIDCapoFamiglia"]+"'" if dip["CodIDCapoFamiglia"] != 'null'
        else 'null'}, '{dip["Reparto"]}');""")
conn.commit()
```

Una volta eseguito lo script si ottiene quindi un popolamento corretto della base di dati, per cui si può procedere con la fase di analisi statistica con il linguaggio *R*.

5.2 Definizione dei Trigger

5.2.1 Inserimento di un nuovo ordine

Implementiamo i trigger per garantire i vincoli aziendali relativi agli ordini:

1. Un ordine deve contenere solo articoli forniti da un unico fornitore.
2. Un ordine deve contenere solo articoli forniti al momento.
3. Un ordine è composto di almeno un articolo.

Per quanto riguarda il primo vincolo, possiamo creare un trigger che operi su ogni riga inserita o aggiornata della tabella *COMPOSTO*. Considerando la nuova tupla *(Ordine co, CodiceArticolo ca, Reparto r, Quantità q)*, se nella tabella *COMPOSTO* ci sono già righe con lo stesso codice ordine *co* allora basta prendere una tupla qualsiasi della forma *(co, ca', r', q')* e verificare che *Fornitore(ca')* = *Fornitore(ca)*; se invece nella tabella composto non ci sono altre tuple con *Ordine* = *co* allora l'inserimento ha successo senza ulteriori controlli.

```
create or replace function ottieni_fornitore(codArticolo dom_CodArticolo, rep
    dom_NomeReparto)
returns dom_PartitaIva
language plpgsql as $$
declare
    result dom_PartitaIva;
begin
    result := null;
    select a.Fornitore into result
    from Articolo a
    where a.Reparto = rep and a.CodiceArticoloReparto = codArticolo;

    return result;
end;
$$;

create or replace function valida_entry_ordine()
returns trigger
language plpgsql as $$
declare
    old_f dom_PartitaIva;
    new_f dom_PartitaIva;
    old_entry record;
begin
    perform *
    from Composto c
    where c.Ordine = new.Ordine;
```

```
    if found
    then
        select * into old_entry
        from Composto c
        where c.Ordine = new.Ordine
        limit 1;

        old_f := ottieni_fornitore(old_entry.CodiceArticolo, old_entry.Reparto);
        new_f := ottieni_fornitore(new.CodiceArticolo, new.Reparto);

        if old_f = new_f
        then
            --il fornitore e' lo stesso degli altri aricoli
            return new;
        else
            --il fornitore non e' lo stesso
            raise exception 'Tutti gli articoli per questo ordine devono essere
                forniti da %.',old_f;
            return null;
        end if;
    else
        --e' il primo articolo inserito
        return new;
    end if;
end;
$$;

create trigger valida_articolo_ordine before insert or update
on Composto
for each row
execute procedure valida_entry_ordine();
```

Per testare il funzionamento eseguiamo il seguente codice:

```
-- popolamento minimo della base di dati:
insert into Dipendente values ('1234567890123456', 'Mario', 'Rossi', 'Via delle
    guide', 9000, 'Dipendente', null);
insert into Reparto values ('Scarpe', 0, '1234567890123456');
update Dipendente set TipoDipendente = 'Manager' where CodId =
    '1234567890123456';
insert into Fornitore values
    ('f1', 'Azienda 1', 'via dei pioppi', 0),
    ('f2', 'Azienda 2', 'via dei pioppi', 0);
insert into Articolo values
    ('a1', 'Scarpe', 5, 'abc', 'f1', '05-03-2022', 4),
    ('a2', 'Scarpe', 4, 'bca', 'f1', '05-03-2022', 3),
    ('a3', 'Scarpe', 3, 'cba', 'f2', '05-03-2022', 2),
    ('a4', 'Scarpe', 8, 'abs', null, '05-03-2022', 4);
insert into Ordine values ('1111111111111', '05-03-2022');

-- test del trigger:
```

```
-- 1: ha successo
insert into Composto values('1111111111111', 'a1', 'Scarpe', 3);
-- 2: ha successo
insert into Composto values('1111111111111', 'a2', 'Scarpe', 3);
-- 3: ERRORE: Tutti gli articoli per questo ordine devono essere forniti da f1.
insert into Composto values('1111111111111', 'a3', 'Scarpe', 3);
-- 4: ERRORE: Tutti gli articoli per questo ordine devono essere forniti da f1.
insert into Composto values('1111111111111', 'a4', 'Scarpe', 3);
```

Come da aspettative i primi due inserimenti hanno successo mentre gli ultimi due no.
Tornando alla lista dei vincoli, per implementare il secondo ci basterebbe creare una U.D.F. *valida_ordine(CodOrdine)* invocata al momento di invio ordine che reperisca tutti gli articoli *(CodArt, Rep)* dalla tabella *COMPOSTO* dove Composto.Ordine = CodOrdine e dalla tabella *ARTICOLO* controlli che l'attributo *fornitore* non sia *null* per ogni *(CodArt, Rep)*. Per implementare il terzo vincolo invece è sufficiente implementare un trigger *BEFORE DELETE or UPDATE* che impedisca la rimozione di un articolo da un ordine, nel caso sia l'unico articolo appartenente a tale ordine.

5.2.2 Inserimento/Modifica di un reparto

Possiamo creare un trigger che operi su ogni riga inserita o aggiornata della tabella *REPARTO*. Considerando la nuova tupla *(Reparto r, NumeroDipendenti n, Manager m)* se nella tabella *DIPENDENTE* esiste il dipendente che ha l'attributo *CodID* = *m* allora devo verificare che questo non afferisca ad un altro dipartimento oppure che sia un dipendente e non un manager. Nel caso in cui venisse aggiornato il manager di un reparto allora il precedente manager deve ritornare dipendente, ovvero all'interno della tupla del vecchio manager in *DIPENDENTE* deve essere aggiornato il campo *tipoDipendente* = *dipendente*. I manager presenti in *DIPENDENTE* hanno l'attributo *reparto* = *null*, questo per evitare dipendenze cicliche tra *DIPENDENTE* e *REPARTO*.

```
create or replace function valida_manager_reparto() -- controllo che il nuovo
    reparto che vado a creare/modificare abbia un manager che: o afferisce a
    quel reparto o a nessuno, nel caso in cui afferisse ad un altro reparto c'e'
    un errore
returns trigger
language plpgsql as $$
declare
    reparto dom_NomeReparto; --il reparto del nuovo manager
    tipoDipendente dom_TipoDipendente; --il tipo di dipendente del nuovo manager
begin
    select d.Reparto into reparto
    from Dipendente d
    where d.CodID = new.ManageR;

    select d.tipoDipendente into tipoDipendente
    from Dipendente d
    where d.CodID = new.ManageR;

    if reparto is null and tipoDipendente = 'Dipendente'
    then
        --allora e' un dipendente appena creato che va ancora assegnato al
            reparto
        raise notice 'Il dipendente % e stato assegnato a manager per il reparto
            %',new.ManageR,new.Nome;
```

```
        update Dipendente set TipoDipendente = 'Manager' where CodId =
            new.ManageR; -- aggiorno il tipoDipendente a manager
        return new;
    else
    if reparto = new.Nome and tipoDipendente = 'Dipendente'
    then
        --allora e' un dipendente del reparto e va fatta una promozione => passa
            da Dipendente a Manager e devo "de-rankare" quello vecchio
        raise notice 'Il dipendente % e stato elevato a manager per il reparto
            %',new.ManageR,new.Nome;
      update Dipendente set TipoDipendente = 'Dipendente', reparto = old.nome
          where CodId = old.ManageR; -- aggiorno il tipoDipendente al vecchio
          manager e gli riassegno il reparto
      update Dipendente set TipoDipendente = 'Manager', reparto = null where
          CodId = new.ManageR; -- aggiorno il tipoDipendente a manager
        return new;
    else
        --il dipendente afferisce ad un altro reparto
        raise exception 'Il dipendente % non afferisce al reparto %,
            aggiornamento fallito.',new.ManageR,new.Nome;
        return null;
    end if;
   end if;
end;
$$;

create trigger controlla_Manager before insert or update
on Reparto
for each row
execute procedure valida_manager_reparto();
```

Per testare il funzionamento eseguiamo il seguente codice:

```
-- popolamento minimo della base di dati:

insert into Dipendente values ('1234567890123456', 'Mario', 'Rossi', 'Via delle
    guide', 9000, 'Dipendente', null);
--PRIMO IF, ovvero assegno ad un reparto un dipendente non occupato e diventa
    manager
insert into Reparto values ('Scarpe', 0, '1234567890123456');

update Dipendente set TipoDipendente = 'Manager' where CodId =
    '1234567890123456';
insert into Dipendente values ('1234567890123457', 'Lorenzo', 'Zanolin', 'Via
    delle guide', 9000, 'Dipendente', 'Scarpe'); --ora LZ e' un dipendente del
    reparto e MR e' manager

--SECONDO IF, ovvero faccio una promozione a ZL e quindi MR diventa dipendente
    => (LZ: dipendente -> manager; MR: manager -> dipendente)
update Reparto set Manager = '1234567890123457' where nome = 'Scarpe';

-- ELSE, ovvero assegno ad un reparto un manager afferente ad un altro reparto
```

```
insert into Reparto values ('Vestiti', 0, '1234567890123457'); --da'
    giustamente errore
```

5.3 Definizione di Query

Per la definizione delle query si è scelto di implementare le tre operazioni più interessanti tra quelle definite nella sezione 1.3.

Query 1 : Corrisponde all'*OPERAZIONE 1*. Dal punto di vista del codice bisogna tenere in considerazione che i manager di reparto non contengono il nome del reparto che dirigono tra i loro attributi; per contarli correttamente è quindi necessario confrontare il codice del manager di un dato reparto con il codice di ogni dipendente.

```
-- 1 Per ogni reparto, restituire il numero di dipendenti e lo stipendio medio

SELECT R.Nome, R.NumeroDipendenti, AVG(D.Stipendio)
FROM Reparto AS R, Dipendente AS D
WHERE R.Nome=D.Reparto OR R.Manager=D.CodId
GROUP BY R.Nome

-- facendo cosi pero' l'inserimento di un dipendente va fatto con delle
    transazioni in modo da evitare che ci siano dei dipendenti senza reparto
```

Query 2 : Corrisponde all'*OPERAZIONE 3*. Per implementare questa operazione si è deciso di usare una UDF per controllare la condizione sui fornitori: l'operazione infatti viene eseguita solo su fornitori che hanno almeno una fornitura attuale attiva, quindi è necessario controllare che il fornitore in input compaia nella tabella ARTICOLO sotto l'attributo FORNITORE di qualche articolo. In caso contrario si notifica l'utente.

```
--2 Restituire il numero di ordini effettuati presso un fornitore che al
    momento fornisce almeno un articolo.

CREATE OR REPLACE FUNCTION numeroOrdini( pIvaFornitore dom_PartitaIva )
RETURNS integer LANGUAGE plpgsql AS
$$
   DECLARE
      numOrdini integer;
   BEGIN
       IF pIvaFornitore NOT IN (SELECT Fornitore FROM Articolo)
         THEN RAISE EXCEPTION '% non fornisce alcun articolo in questo momento',
             pIvaFornitore;
       END IF;
       SELECT NumeroOrdini INTO numOrdini FROM Fornitore WHERE PartitaIva =
          pIvaFornitore;
       RETURN numOrdini;
   END;
$$;
```

Query 3 : Corrisponde all'*OPERAZIONE 5*. Anche questa query viene implementata attraverso una UDF, poiché è necessario controllare che le date inserite dall'utente siano ordinate (dataInizio < dataFine). Se il periodo è valido si considerano tutte le tuple di (COMPOSTO *join* ORDINE) che riguardano l'articolo desiderato e rientrano nel periodo selezionato. La *join* è necessaria perché la data dell'ordine è contenuta nella relazione ORDINE.

```
--3 Visualizzare la quantita' ordinata di un determinato articolo in un preciso
   periodo.

CREATE OR REPLACE FUNCTION quantitaOrdinata (codArticolo dom_CodArticolo, rep
   dom_NomeReparto, dataInizio date, dataFine date)
RETURNS integer LANGUAGE plpgsql AS
$$
   DECLARE
      quantita integer;
   BEGIN
      IF dataInizio >= dataFine
         THEN RAISE EXCEPTION 'Periodo non valido';
      END IF;
      SELECT COUNT(*) AS quantitaOrdinata INTO quantita
      FROM (Composto JOIN Ordine ON Composto.Ordine = Ordine.CodiceOrdine) AS CO
      WHERE codArticolo = CO.CodiceArticolo AND
         reparto = CO.Reparto AND
         CO.DataOrdine BETWEEN dataInizio AND dataFine;
      RETURN quantita;
   END;
$$;
```

6 Analisi di dati in R

6.1 Connessione alla base di dati

Per prima cosa, è necessario collegarsi dall'ambiente interattivo di R al *DBMS* locale per poter richiedere e manipolare i dati presenti nel database.

```
library("RPostgreSQL")
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv,
    dbname = "supermercato",
    host = "localhost",
    port = 5432,
    user = "postgres",
    password = "#######")
res <- dbSendQuery(con, "SET search_path TO supermercato;")
```

6.2 Spesa in stipendi relativa al numero di articoli venduti

Vogliamo analizzare la spesa complessiva stipendiale di ciascun reparto in relazione al numero di articoli che vende. Per far ciò, è necessario prima effettuare una *query* al database per caricare in memoria primaria i dati richiesti, per poi visualizzarli attraverso la funzione di *plotting*.

```
res <- dbGetQuery(con,
    "SELECT reparto, sommastipendi, numeroarticoli
     FROM
       (SELECT reparto, SUM(stipendio) AS sommastipendi
       FROM dipendente
       GROUP BY reparto) AS RS
       NATURAL JOIN
       (SELECT reparto, COUNT(*) AS numeroarticoli
       FROM articolo
       GROUP BY reparto) AS RA;"
   )
par(mar=c(5, 7, 5, 5))
plot(res$numeroarticoli, res$sommastipendi,
    main="Distribuzione degli stipendi in relazione al numero di articoli
        venduti da un reparto",
    xlim=c(230,275), ylim=c(500000, 1200000),
    xlab="Numero di articoli venduti", ylab="",
    pch=19, bty="l")
title(ylab="Somma degli stipendi [Euro]", line=4)
text(res$numeroarticoli, res$sommastipendi, labels=res$reparto, pos=1, cex=0.8)
```

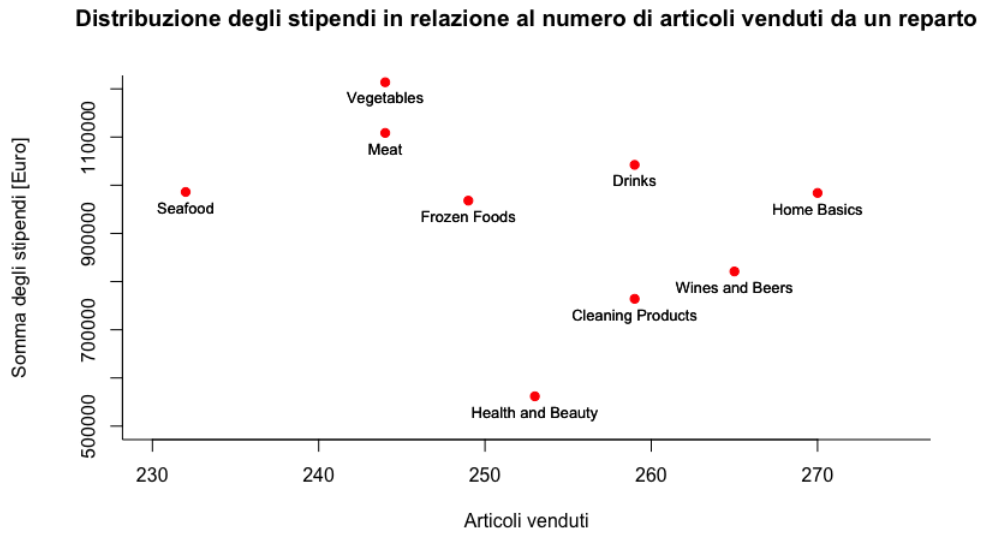


Figura 14: Grafico sulla distribuzione degli stipendi.

Abbiamo scelto lo *scatterplot 2D* per visualizzare questa informazione perché tale tipo di grafico permette di raffigurare in maniera molto intuitiva fenomeni bivariati.

6.3 Storico del numero di articoli venduti da un fornitore

Vogliamo visualizzare come varia di anno in anno la quantità di articoli venduti da un determinato fornitore al supermercato. Per rendere l'analisi più interessante, effettuiamo lo studio di questo andamento per più fornitori diversi. Ne scegliamo 5 arbitrariamente, ad esempio prendendo quelli con il numero maggiore di ordini effettuati:

```
res <- dbGetQuery(con,
    "SELECT partitaiva, anno, numeroforniture
     FROM fornitore, numero_forniture_per_anno(partitaiva)
     WHERE PartitaIva IN (
        SELECT PartitaIva
        FROM Fornitore
        ORDER BY NumeroOrdini DESC
        LIMIT 5
     )")
```

La funzione `numero_forniture_per_anno()` prende in input il codice di Partita Iva di un fornitore e restituisce il numero di forniture attive per tale fornitore anno per anno. Tale UDF è definita dal seguente codice:

```
CREATE OR REPLACE FUNCTION numero_forniture_per_anno (codice_fornitore
    dom_PartitaIva)
RETURNS table (anno integer, numeroforniture integer) LANGUAGE plpgsql AS
$$
    DECLARE
      anno_min integer;
      anno_max integer;
   BEGIN
```

```
      SELECT date_part('year', MIN(datainizio)) INTO anno_min FROM
          forniturapassata;
      SELECT date_part('year', CURRENT_DATE) INTO anno_max;
      DROP TABLE IF EXISTS temp_table;
      CREATE TEMP TABLE temp_table AS
         SELECT *
         FROM (
            SELECT generate_series AS anno
            FROM generate_series(anno_min,anno_max)
         ) AS anno, (
            SELECT generate_series AS numeroforniture
            FROM generate_series(0,0)
         ) AS numeroforniture;
      FOR i IN anno_min..anno_max LOOP
         UPDATE temp_table SET numeroforniture =
             forniture_annuali(codice_fornitore, i) WHERE temp_table.anno = i;
      END LOOP;
      RETURN QUERY
         SELECT *
         FROM temp_table;
   END;
$$;
```

numero_forniture_per_anno() si basa sulla funzione ausiliaria forniture_annuali() che, fissato un fornitore e un anno, restituisce il numero di forniture attive quell'anno:

```
CREATE OR REPLACE FUNCTION forniture_annuali (codice_fornitore dom_PartitaIva,
    anno integer)
RETURNS integer LANGUAGE plpgsql AS
$$
   DECLARE
   totale integer;
   BEGIN
      WITH forniturетotali AS (
         SELECT COUNT(*) AS numeroforniture
         FROM forniturapassata
         WHERE fornitore = codice_fornitore AND
            date_part('year', datainizio) <= anno AND
            date_part('year', datafine) >= anno
         UNION
         SELECT COUNT(*) AS numeroforniture
         FROM articolo
         WHERE fornitore = codice_fornitore AND
            date_part('year', datainizio) <= anno
      )
      SELECT SUM(numeroforniture) AS numeroforniture
      FROM forniturетotali
      INTO totale;
      RETURN totale;
   END;
$$;
```

Tornando all'analisi statistica in R, bisogna riorganizzare i dati in maniera da poter generare l'*heatmap* corrispondente:

```
anni <- unique(res$anno)
fornitori <- unique(res$partitaiva)
dati <- matrix(res$numeroforniture, nrow=length(anni))
par(mar=c(5, 10, 5, 5))
image(1:nrow(dati), 1:ncol(dati), dati, axes=FALSE, ylab="", xlab="",
   main="Numero di forniture attive per anno")
axis(1, at=seq(from=1, by=1, to=length(anni)), labels=anni, lwd=0)
title(xlab="Anni", line=3)
axis(2, at=seq(from=1, by=1, to=length(fornitori)), labels=fornitori, lwd=0,
   las=1)
title(ylab="Fornitori [Partita Iva]", line=8)
e <- expand.grid(seq(from=1, by=1, to=length(anni)), seq(from=1, by=1,
   to=length(fornitori)))
text(e, labels=(dati))
```

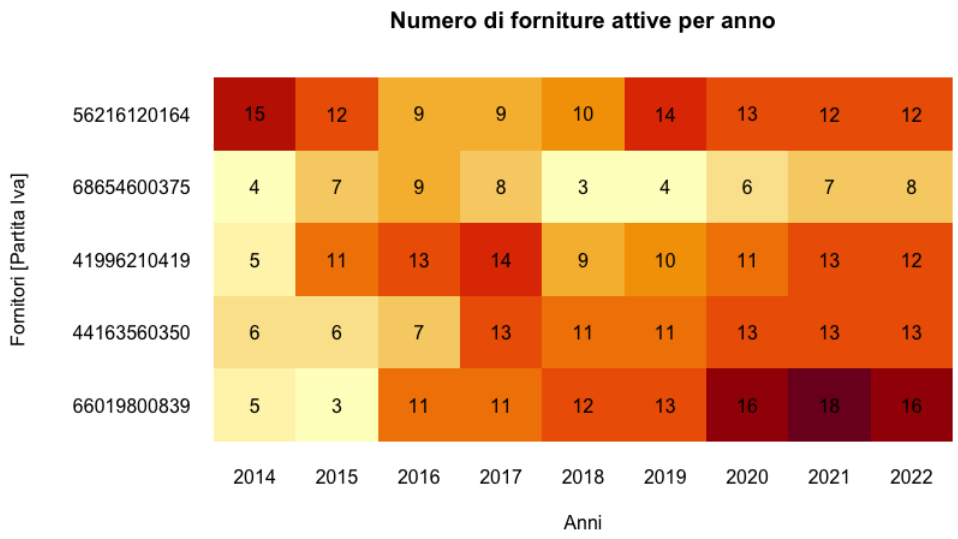


Figura 15: Grafico sulla variazione del numero di forniture.

L'uso dei colori consentito da una *heatmap* permette di visualizzare l'informazione in maniera semplice e rapida: l'intensità di rosso trasmette istantaneamente la connotazione del valore che si vuole reperire, seppur in maniera approssimativa. Per fornire dati più precisi, abbiamo quindi usato la funzione `text()` per sovrapporre i valori effettivi al grafico e, quindi, unire l'efficacia delle *heatmap* con l'accuratezza delle tabelle classiche.