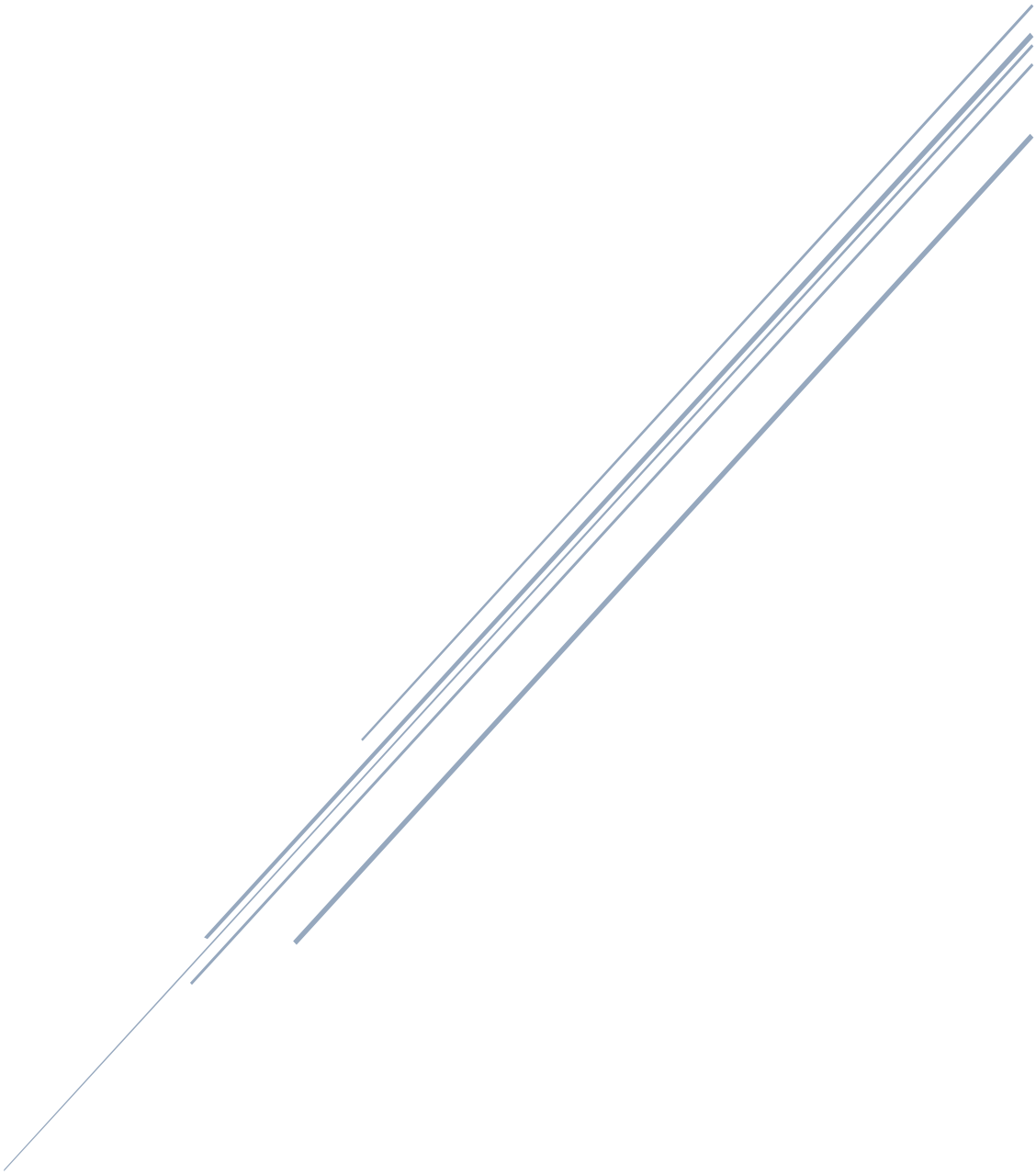
PROGETTO DI BASI DI DATI 2

Centro commerciale

Documentazione



Francesco Pontillo
Matricola 600119

Sommario

1 Requisiti

1.1 Raccolta dei requisiti

Si desidera creare un sistema informatico per un centro commerciale.

Centro commerciale

Il centro commerciale è costituito da un insieme di edifici nei quali si concentrano numerose attività commerciali, come quelle per la grande distribuzione organizzata, negozi specializzati, cinema e attività di ristorazione. L'accesso agli edifici che ospitano le diverse imprese commerciali è reso possibile da piccole vie e piazze, spesso al coperto (gallerie) aperte solo al traffico pedonale.
Tutte le strutture di cui è composto il centro (gli edifici, le piazze, le gallerie...) sono identificate univocamente da una stringa alfanumerica, composta da un carattere identificativo della tipologia di struttura (E per edificio, P per piazza, G per galleria ecc.) e da un numero. Le varie attività commerciali sono identificate univocamente dal codice della struttura nella quale sono collocate e dal proprio codice numerico.
Per ogni attività commerciale si deve, quindi, memorizzare il nome della struttura nel quale è collocata ed il piano (nel caso degli edifici). I negozi, inoltre, sono organizzati per categorie. Alcuni esempi di tipologie di negozi sono l'abbigliamento, l'arredo e l'informatica. Altre informazioni di interesse per le attività commerciali sono il nome (o ragione sociale), numero di dipendenti, la partita iva e il nome del proprietario, se l'attività non è in franchising o, in caso contrario, il nome del responsabile.
Anche le attività di ristorazione sono suddivise per tipologia, come i fast-food, i ristoranti tipici di una particolare nazione del mondo o area geografica (italiana o estera).
Il database, infine, tiene memoria di tutti i dipendenti del centro. Per ogni dipendente, si vuole conoscere, insieme ai dati anagrafici, anche la data di assunzione (ed eventualmente quella di licenziamento) e l'attività commerciale presso la quale solo stati assunti e il periodo di lavoro.

1.2 Analisi dei requisiti

Si analizzano i requisiti, chiarendo concetti ambigui, ponendo un adeguato livello di astrazione, standardizzando e linearizzando le frasi più complesse. Verranno risolte le ambiguità introdotte da eventuali omonimie e sinonimie, presentando i concetti in maniera strutturata.

Infine verrà introdotto un glossario finale dei termini.

Nell'analisi dei requisiti si utilizzeranno i numeri di riga come riportati al paragrafo 1.1.

1.2.1 Livello di astrazione

Ai righi 3 e 17 per “attività di ristorazione” si fa riferimento a ristoranti.

Al rigo 4, per “imprese commerciali” si fa riferimento ad attività commerciali.

Al rigo 15, per “nome del proprietario” si intende nome e cognome del proprietario dell'attività commerciale.

Al rigo 16, per “nome del responsabile” si intende nome e cognome del responsabile dell'attività commerciale.

Ai righi 17 e 18, si intende tenere traccia del nome dell'area geografica, nel modo più generico possibile.

Al rigo 20, per “dati anagrafici” di un dipendente si intende nome, cognome, data di nascita, luogo di nascita, sesso e codice fiscale del dipendente.

Al rigo 22, per “periodo di lavoro” si intende il numero di giorni dalla data di assunzione alla data di licenziamento, oppure alla data corrente.

1.2.2 Standardizzazione e linearizzazione

Il centro commerciale è costituito da un insieme di edifici nei quali si concentrano numerose attività commerciali.

Le attività commerciali possono essere quelle per la grande distribuzione organizzata, negozi specializzati, cinema e attività di ristorazione.

L'accesso agli edifici che ospitano le diverse imprese commerciali è reso possibile da piccole vie, piazze, gallerie (piazze coperte aperte solo al traffico pedonale).

Tutte le strutture di cui è composto il centro (gli edifici, le piazze, le gallerie, le vie) sono identificate univocamente da una stringa alfanumerica, composta da un carattere identificativo della tipologia di struttura (E per edificio, P per piazza, G per galleria, V per via) e da un numero.

Le varie attività commerciali sono identificate univocamente dal codice della struttura nella quale sono collocate e dal proprio codice numerico.

Per ogni attività commerciale si deve memorizzare il nome della struttura nel quale è collocata ed il piano (nel caso degli edifici).

I negozi sono organizzati per categorie. Alcuni esempi di tipologie di negozi sono l'abbigliamento, l'arredo e l'informatica.

Informazioni di interesse per le attività commerciali sono il nome (o ragione sociale), numero di dipendenti, la partita iva. Se l'attività è in franchising si vuole memorizzare il nome e il cognome del responsabile. Se l'attività non è in franchising si vuole memorizzare il nome e cognome del proprietario.

Anche le attività di ristorazione sono suddivise per tipologia, come i fast-food, i ristoranti tipici di una particolare nazione del mondo o area geografica (italiana o estera).

Il database, infine, tiene memoria di tutti i dipendenti del centro. Per ogni dipendente, si vuole conoscere, insieme ai dati anagrafici, anche la data di assunzione, l'eventuale data di licenziamento, l'attività commerciale presso la quale solo stati assunti e il periodo di lavoro.

1.2.3 Omonimie e sinonimie

Al rigo 1, per “centro commerciale” si utilizzerà “centro”.

Ai righi 2, 9, 11, 14, 21, per “attività commerciale” si utilizzerà “attività”.

Ai righi 2 e 3, per “negozio specializzato” si utilizzerà “negozio”.

Ai righi 3, 17, per “attività di ristorazione” si utilizzerà “ristorante”.

Al rigo 4, per “impresa commerciale” si utilizzerà “attività”.

Al rigo 13, per “categoria di negozio” e “tipologia di negozio” si utilizzerà “tipo di negozio”.

Al rigo 17, per “tipologia” si utilizzerà “tipo di ristorante”.

1.2.4 Requisiti trasformati

A questo punto, per facilitare il lavoro nelle fasi successive è possibile riscrivere i requisiti dopo le fasi di linearizzazione, scelta di un corretto livello di astrazione ed eliminazione di omonimie e sinonimie.

Centro commerciale (requisiti riscritti)
Il centro è costituito da un insieme di edifici nei quali si concentrano numerose attività. Le attività possono essere quelle per la grande distribuzione organizzata, negozi, cinema e ristoranti. L'accesso agli edifici che ospitano le diverse attività è reso possibile da piccole vie, piazze, gallerie (piazze coperte aperte solo al traffico pedonale). Tutte le strutture di cui è composto il centro (gli edifici, le piazze, le gallerie, le vie) sono identificate univocamente da una stringa alfanumerica, composta da un carattere identificativo della tipologia di struttura (E per edificio, P per piazza, G per galleria, V per via) e da un numero. Le varie attività sono identificate univocamente dal codice della struttura nella quale sono collocate e dal proprio codice numerico. Per ogni attività si deve memorizzare il nome della struttura nel quale è collocata ed il piano (nel caso degli edifici). I negozi sono organizzati per tipi di negozio. Alcuni esempi di tipi di negozi sono l'abbigliamento, l'arredo e l'informatica. Informazioni di interesse per le attività sono il nome (o ragione sociale), numero di dipendenti, la partita iva. Se l'attività è in franchising si vuole memorizzare il nome e il cognome del responsabile. Se l'attività non è in franchising si vuole memorizzare il nome e cognome del proprietario. Anche i ristoranti sono suddivisi per tipo di ristorante, come i fast-food, i ristoranti tipici di una particolare nazione del mondo o area geografica (italiana o estera). Il database, infine, tiene memoria di tutti i dipendenti del centro. Per ogni dipendente, si vuole conoscere, insieme ai dati anagrafici, anche la data di assunzione, l'eventuale data di licenziamento, l'attività presso la quale solo stati assunti e il periodo di lavoro.

1.2.5 Raggruppamento di frasi per concetti

A partire dai requisiti riscritti, è possibile raggruppare le frasi per concetti.

1.2.5.1 Frasi di carattere generale

Si desidera creare un sistema informatico per un centro commerciale.

1.2.5.2 Frasi relative alle strutture

Il centro è costituito da un insieme di edifici nei quali si concentrano numerose attività.

L'accesso agli edifici che ospitano le diverse attività è reso possibile da piccole vie, piazze, gallerie (piazze coperte aperte solo al traffico pedonale).

Tutte le strutture di cui è composto il centro (gli edifici, le piazze, le gallerie, le vie) sono identificate univocamente da una stringa alfanumerica, composta da un carattere identificativo della tipologia di struttura (E per edificio, P per piazza, G per galleria, V per via) e da un numero.

1.2.5.3 Frasi relative alle attività

Le attività possono essere quelle per la grande distribuzione organizzata, negozi, cinema e ristoranti.

Le varie attività sono identificate univocamente dal codice della struttura nella quale sono collocate e dal proprio codice numerico.

Per ogni attività si deve memorizzare il nome della struttura nel quale è collocata ed il piano (nel caso degli edifici).

Informazioni di interesse per le attività sono il nome (o ragione sociale), numero di dipendenti, la partita iva. Se l'attività è in franchising si vuole memorizzare il nome e il cognome del responsabile. Se l'attività non è in franchising si vuole memorizzare il nome e cognome del proprietario.

1.2.5.4 Frasi relative ai negozi

I negozi sono organizzati per tipi di negozio. Alcuni esempi di tipi di negozi sono l'abbigliamento, l'arredo e l'informatica.

1.2.5.5 Frasi relative ai ristoranti

Anche i ristoranti sono suddivisi per tipo di ristorante, come i fast-food, i ristoranti tipici di una particolare nazione del mondo o area geografica (italiana o estera).

1.2.5.6 Frasi relative ai dipendenti

Informazioni di interesse per le attività sono il nome (o ragione sociale), numero di dipendenti, la partita iva (frase ripetuta).

Il database, infine, tiene memoria di tutti i dipendenti del centro. Per ogni dipendente, si vuole conoscere, insieme ai dati anagrafici, anche la data di assunzione, l'eventuale data di licenziamento, l'attività presso la quale solo stati assunti e il periodo di lavoro.

1.2.6 Glossario dei termini

Nella seguente tabella vengono presentati tutti i concetti ritrovati, collegandoli ad eventuali altre entità e chiarendo eventuali sinonimie ritrovate ai passi precedenti.

Termine	Descrizione	Sinonimi	Collegamenti
Struttura	Contiene attività e può essere di diverso tipo.		Attività
Attività	Può essere di diverso tipo e si trova in una sola struttura.	Attività commerciale Impresa commerciale	Struttura Dipendente
Negozio	È un tipo di attività.	Negozio specializzato	Attività Tipo di negozio
Ristorante	È un tipo di attività.	Attività di ristorazione	Attività Tipo di ristorante
Tipo di negozio	Definisce la categoria del negozio.	Categoria Tipologia di negozio	Negozio
Tipo di ristorante	Definisce la categoria del ristorante.	Tipologia di ristorante	Ristorante
Dipendente	Persona impiegata per un certo periodo di tempo in un'attività.		Attività

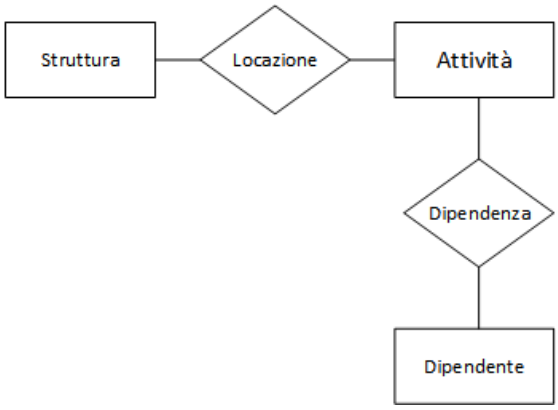
2 Progettazione concettuale

Si utilizza la strategie ibrida per progettare lo schema concettuale: a partire da uno schema scheletro contenente, a livello astratto, i concetti principali, si prosegue per raffinamenti successivi o estendendo lo schema con altre entità sviluppate separatamente.

2.1 Schema concettuale

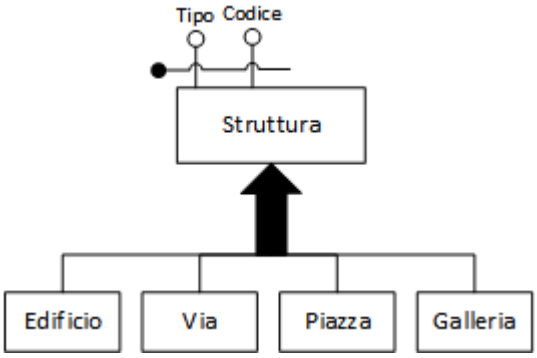
Per la realizzazione dello schema concettuale si è utilizzato Microsoft Visio 2013 (su piattaforma Windows 7).

2.1.1 Schema scheletro

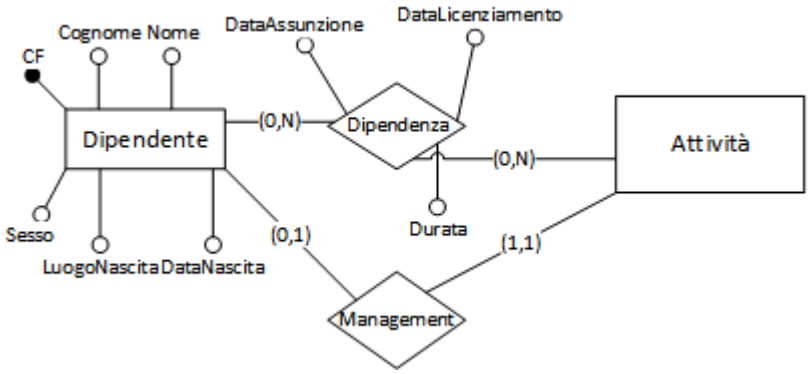


2.1.2 Esplosione di entità

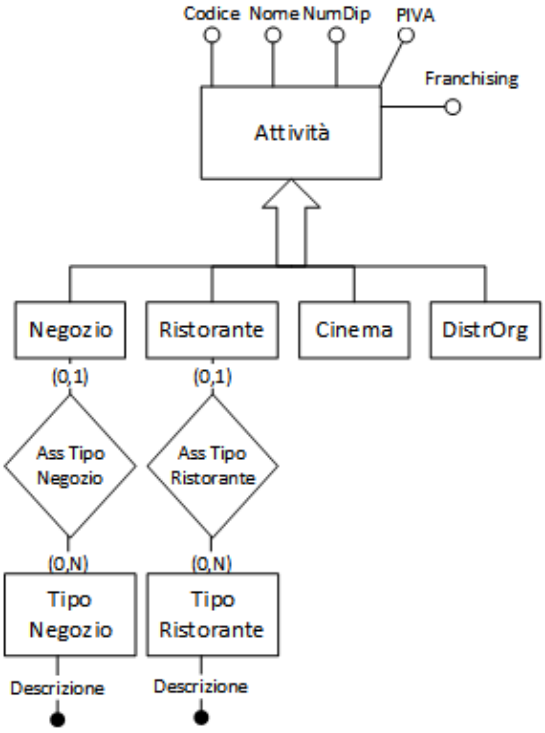
2.1.2.1 Struttura



2.1.2.2 Dipendente

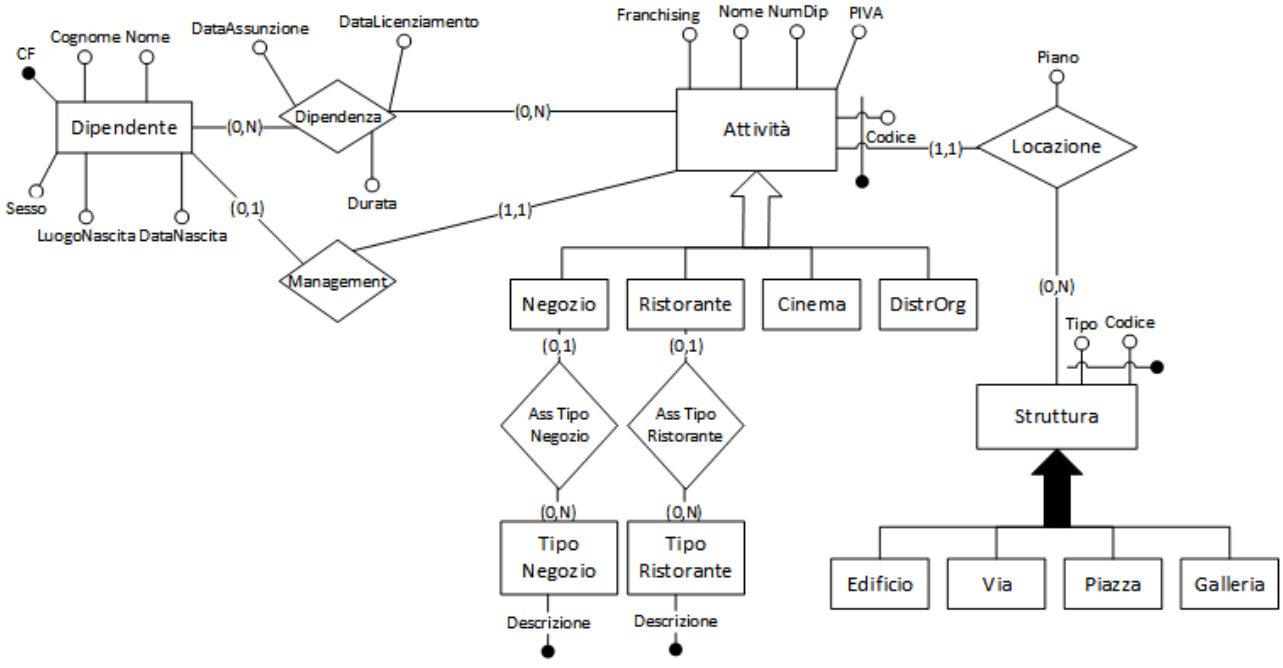


2.1.2.3 Attività



2.1.3 Schema concettuale finale

Integrando i diversi schemi nello schema scheletro otteniamo uno schema concettuale completo.



2.2 Dizionario dei dati

Per ogni concetto si definisce una descrizione, gli attributi e un identificatore principale.

2.2.1 Entità

Entità	Descrizione	Attributi	Identificatore
Attività	Può essere di diverso tipo e si trova in una sola	PIVA Nome	Codice Struttura

	struttura.	NumDip Franchising	
Negozio	È un tipo di attività.		
Ristorante	È un tipo di attività.		
Cinema	È un tipo di attività.		
Distribuzione Organizzata	È un tipo di attività.		
Tipo Negozio	Definisce la categoria del negozio.		Descrizione
Tipo Ristorante	Definisce la categoria del ristorante.		Descrizione
Struttura	Contiene attività e può essere di diverso tipo.		Codice, Tipo
Edificio	Tipo di struttura con diversi piani.		
Via	Tipo di struttura.		
Piazza	Tipo di struttura.		
Galleria	Tipo di struttura.		
Dipendente	Persona impiegata in qualche attività.	Cognome Nome Sesso LuogoNascita DataNascita	CF

2.2.2 Relazioni

Relazione	Descrizione	Attributi	Entità
Ass Tipo Negozio	Lega un negozio ad un possibile tipo.		Negozio (0,1) Tipo Negozio (0,N)
Ass Tipo Ristorante	Lega un ristorante ad un possibile tipo.		Ristorante (0,1) Tipo Ristorante (0,N)
Locazione	Collega un'attività ad una specifica struttura.	Piano	Attività (1,1) Struttura (0,N)
Dipendenza	Lega un dipendente a una attività.	DataAssunzione DataLicenziamento Durata	Attività (0,N) Dipendente (0,N)
Management	Lega un'attività ad un dipendente che funge da manager.		Attività (1,1) Dipendente (0,1)

2.3 Regole aziendali

2.3.1 Regole di vincolo

1. Un dipendente non può avere due dipendenze attive alla stessa data.
2. Ogni volta che viene aggiunto o eliminato un dipendente, si deve aggiornare NumDip dell'attività del dipendente modificato.
3. Periodicamente va aggiornata la durata della dipendenza.
4. Il proprietario o il responsabile di un'attività deve essere un dipendente della stessa.

2.3.2 Regole di derivazione

1. Il responsabile di una attività si ottiene, se l'attività è in franchising, visitando la relazione Management.
2. Il proprietario di una attività si ottiene, se l'attività non è in franchising, visitando la relazione Management.

3 Progettazione logica

La progettazione logica ha come obiettivo quello di tradurre lo schema ER prodotto nella fase precedente in uno schema logico in grado di descrivere gli stessi dati in maniera corretta ed efficiente, prestando particolare attenzione alle prestazioni. Per tale motivo verranno analizzati i dati, le operazioni ed i relativi costi di accesso ad ogni entità e associazione.

3.1 Operazioni

Alcune operazioni di esempio che la base di dati può supportare sono:

1. Inserimento di una struttura.
2. Modifica di una struttura.
3. Aggiunta di un'attività pre-esistente ad una struttura.
4. Spostamento di un'attività in una struttura.
5. Assunzione di un dipendente in una attività.
6. Cambio di proprietario di un'attività.
7. Stampa di tutte le attività del centro, incluso il numero dei dipendenti.
8. Stampa della storia delle assunzioni di ogni attività, con data di assunzione, data di licenziamento e durata.
9. Ricerca del proprietario di un'attività.

3.2 Ristrutturazione dello schema E-R

3.2.1 Tavola dei Volumi

Nella seguente tabella si elenca il volume di ogni classe di entità, in base ad alcune assunzioni presentate successivamente.

Concetto	Tipo	Volume
Struttura	E	40
Edificio	E	10
Via	E	8
Piazza	E	7
Galleria	E	15
Attività	E	400
Locazione	R	400
Negozio	E	250
Ristorante	E	20
Cinema	E	5
DistrOrg	E	10
Tipo Negozio	E	50
Tipo Ristorante	E	50
Ass Tipo Negozio	R	200
Ass Tipo Ristorante	R	15
Dipendente	E	15000
Dipendenza	R	20000
Management	R	400

Sui volumi sono state fatte alcune assunzioni:

- Ogni struttura ha in media 10 attività.
- Ci possono essere altri tipi di attività non specificati dalla gerarchia.
- Quasi tutti i negozi hanno specificato una tipologia di negozio.
- Quasi tutti i ristoranti hanno specificato una tipologia di ristorante.

- Ogni attività ha una media di 20 dipendenze attive in ogni momento.
- Circa 15000 dipendenti hanno transitato da una o più attività del centro nel corso della loro carriera.
- Circa 1 dipendente su 4 ha avuto almeno 2 assunzioni all'interno del centro.

3.2.2 Tavola delle Operazioni

Per ogni operazione definita in precedenza si definisce la tipologia (interattiva o batch) e la frequenza assunta per l'operazione in un certo periodo di tempo.

Op.	Tipo	Frequenza
1	I	5/anno
2	I	15/mese
3	I	5/mese
4	I	5/mese
5	I	100/mese
6	I	3/mese
7	B	1/mese
8	B	2/anno
9	I	10/giorno

3.2.3 Tavole degli Accessi

Per ogni operazione si calcolano gli accessi necessari, in lettura e scrittura, ad ogni entità, utilizzando la tavola dei volumi come base per il calcolo.

3.2.3.1 Operazione 1

Concetto	Costrutto	Tipo	Accessi
Struttura	E	S	1
Edificio/Via/...	E	S	1

3.2.3.2 Operazione 2

Concetto	Costrutto	Tipo	Accessi
Struttura	E	S	1
Edificio/Via/...	E	S	1

3.2.3.3 Operazione 3

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	1
Struttura	E	L	1
Locazione	R	S	1

3.2.3.4 Operazione 4

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	1
Struttura	E	L	1
Locazione	R	S	2

Vengono eseguite due attività di scrittura su Locazione, una di cancellazione, l'altra di inserimento.

3.2.3.5 Operazione 5

Concetto	Costrutto	Tipo	Accessi
Dipendente	E	L	1
Attività	E	L	1
Dipendenza	R	S	1

Attività	E	S	1

Si scrive prima la dipendenza, poi si incrementa il numero dei dipendenti dell'attività.

3.2.3.6 Operazione 6

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	1
Dipendente	E	L	1
Management	R	S	2

3.2.3.7 Operazione 7

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	400

3.2.3.8 Operazione 8

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	400
Dipendenza	R	L	20000
Dipendente	E	L	15000

3.2.3.9 Operazione 9

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	1
Management	R	L	1
Dipendente	E	L	1

3.2.4 Analisi delle ridondanze

Sono presenti due ridondanze nello schema concettuale in ingresso.

3.2.4.1 Durata assunzione

Durata nella relazione Dipendenza è ottenibile tramite l'esecuzione di una semplice differenza fra la data di licenziamento (o quella attuale se la data di licenziamento è nulla) e la data di assunzione.

L'unica operazione che fa uso della durata della dipendenza è l'operazione 8.

Consideriamo il caso in cui la ridondanza rimanga:

- Il costo dell'operazione 8 sarebbe di 35400.
- Ogni ridondanza aggrava la base di dati di 4 byte, quindi si ha uno spreco di spazio di 20000*4 byte = 80000 byte.

Se la ridondanza viene eliminata:

- Il costo dell'operazione 8 rimane di 35400, in quanto il costo di una sottrazione è irrilevante rispettivamente all'accesso ai blocchi in memoria secondaria.
- Si risparmierebbero 80000 byte.

Pertanto, si decide di rimuovere la ridondanza.

3.2.4.2 Numero dipendenti

Il numero dei dipendenti è ottenibile tramite un semplice conteggio delle dipendenze che hanno DataLicenziamento non impostata.

Le operazioni che fanno uso del numero dei dipendenti sono le operazioni 5 e 7.

Consideriamo il caso in cui la ridondanza rimanga:

- Assumendo che la dimensione dell'attributo NumDip sia di 4 byte, lo spreco di memoria è di 400*4 byte = 1600 byte.
- Il costo unitario dell'operazione 5 è di 6.
- Il costo unitario dell'operazione 7 è di 400.

Se si considera l'eliminazione della ridondanza, bisogna creare le tavole degli accessi delle operazioni 5 e 7 in sua assenza.

La tavola degli accessi per l'operazione 5 è così definita:

Concetto	Costrutto	Tipo	Accessi
Dipendente	E	L	1
Attività	E	L	1
Dipendenza	R	S	1

La tavola degli accessi per l'operazione 7 è così definita:

Concetto	Costrutto	Tipo	Accessi
Attività	E	L	400
Dipendenza	R	L	8000

Se la ridondanza viene eliminata, quindi:

- Si avrebbe un risparmio di memoria di 1600 byte.
- Il costo unitario dell'operazione 5 sarebbe di 4.
- Il costo unitario dell'operazione 7 sarebbe di 8400.

In presenza di ridondanza, quindi, il costo totale sarebbe di $(100 * 6) + (1 * 400) = 1000$.

In assenza di ridondanza, il costo totale sarebbe di $(100 * 4) + (1 * 8400) = 8800$.

Poiché il risparmio di memoria è trascurabile, e visto il guadagno di prestazioni ottenuto con la ridondanza, si decide di non eliminare la ridondanza relativa al numero dei dipendenti di un'attività.

3.2.5 Eliminazione delle gerarchie

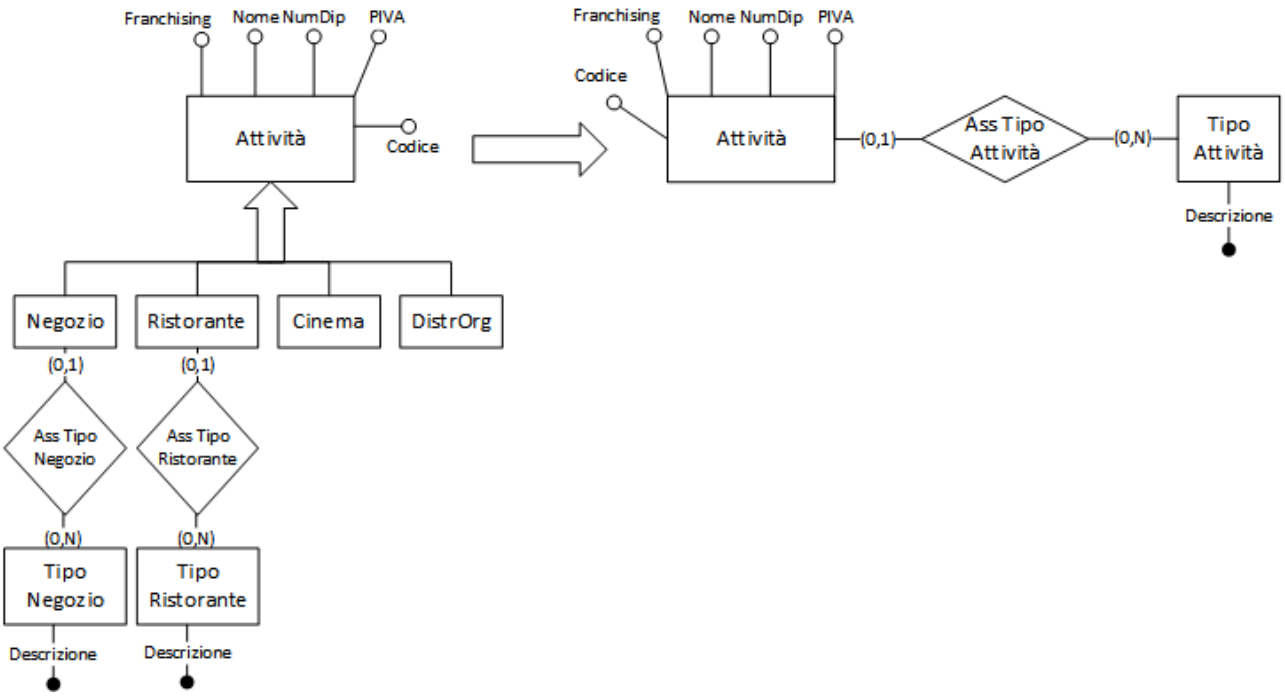
Sono presenti due gerarchie nello schema concettuale.

3.2.5.1 Gerarchia di Attività

Per eliminare la gerarchia di Attività si è scelto di eliminare le entità figlie. Queste, infatti, non partecipano singolarmente a nessuna relazione che non possa essere accorpata nel padre. Poiché ogni attività può avere al massimo un tipo associato, si unificano anche i tipi di attività e si utilizza un'unica relazione per eseguire le associazioni.

Ad esempio, Tipo Attività potrà contenere “Ristorante italiano”, o “Negozio di informatica”, o “Negozio di abbigliamento”, o ancora “Ristorante tipico lucano”.

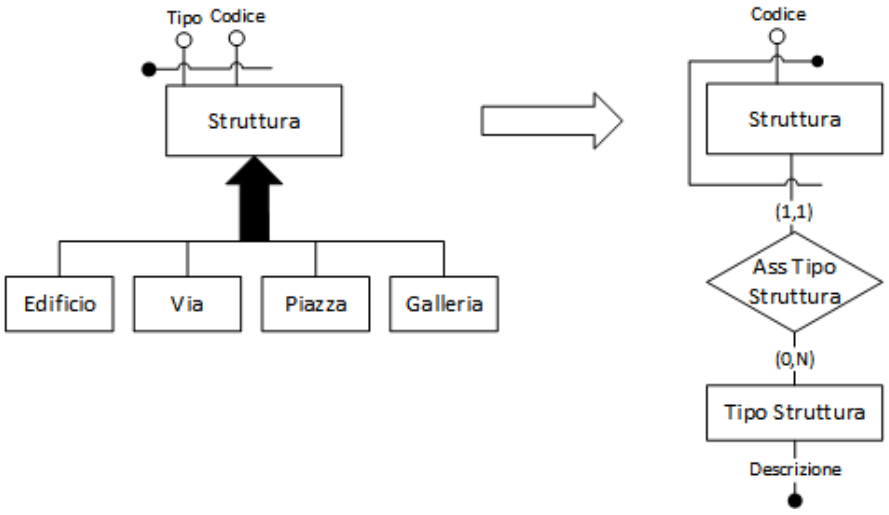
In questo modo sarà possibile inserire qualsiasi nuova attività senza essere forzati a dover scegliere fra le 4 macro-tipologie Negozio, Ristorante, Cinema, DistrOrg.



3.2.5.2 Gerarchia di attività

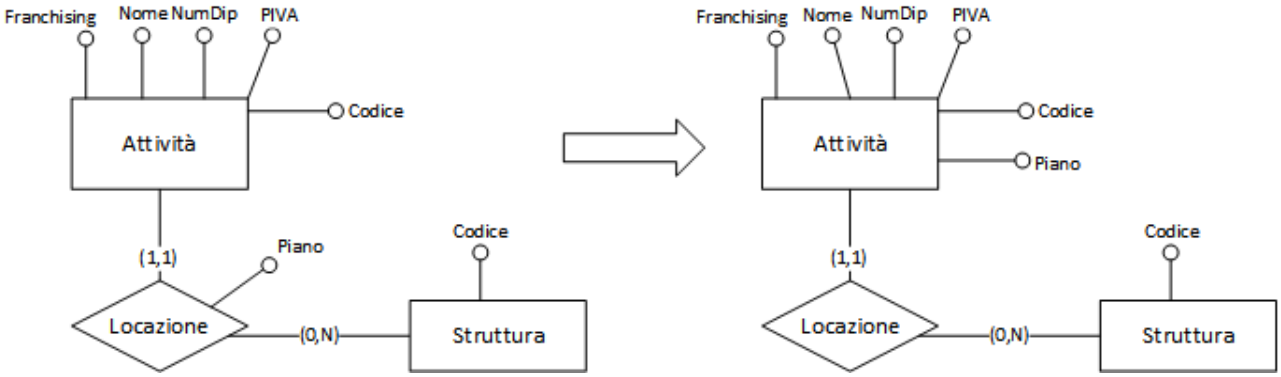
Nella gerarchia di Struttura, le entità figlie non hanno nessun attributo particolare, né partecipano ad alcuna relazione. Pertanto, si può decidere di eliminare la gerarchia, sostituendola con una tipizzazione della Struttura realizzata tramite una relazione.

Si introduce l'entità Tipo Struttura, che contiene una descrizione della tipologia di struttura. Ogni Struttura, in questo modo, dovrà essere di un solo tipo.

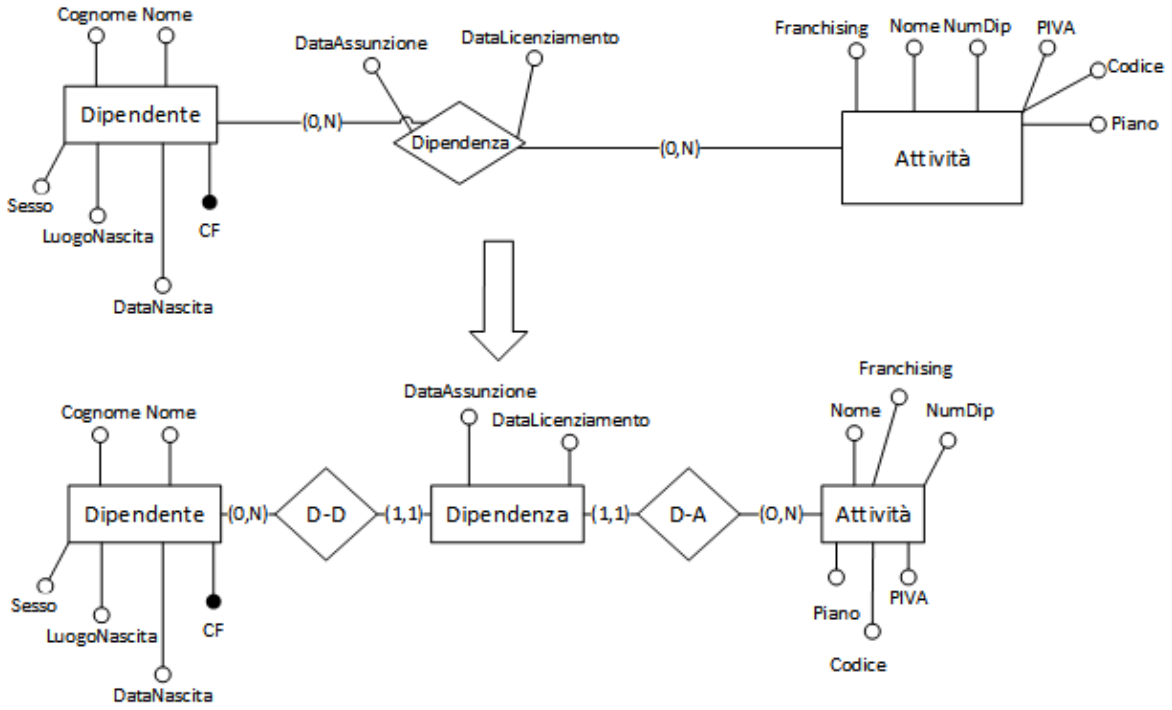


3.2.6 Partizionamento ed accorpamento di concetti

Locazione è una relazione, ma ha un attributo. Poiché Locazione è obbligatoria, si sposta l'attributo Piano in Struttura, ammettendo valori nulli.



Si sceglie anche di reificare l'associazione Dipendenza, in quanto è un concetto a sé stante e rappresenta una assunzione di un dipendente in un'attività, con due date, una di assunzione e una di licenziamento. Gli attributi dell'associazione vengono spostati sulla nuova entità.



3.2.7 Scelta degli identificatori principali

Fra tutti gli identificatori principali fin qui ritrovati si sceglie di adottare esclusivamente quelli che possono essere generati dal sistema. Gli altri, essendo passibili di modifica dall'utente, vengono considerati campi univoci (spesso anche con indice univoco), ma non vengono utilizzati come chiave primaria. Nella tabella seguente si elencano tutti gli identificatori delle tabelle.

Concetto	Identificatore
Dipendente	ID
Dipendenza	ID
Attività	IDAttivita
Tipo Attività	ID
Struttura	IDStruttura
Tipo Struttura	ID

Attività e Struttura partecipano ad alcune relazioni, pertanto un identificatore composto non risulta opportuno.

Per Tipo Struttura il codice viene mantenuto, poiché utile a rappresentare la struttura, ma viene introdotto anche un ID seriale generato automaticamente.

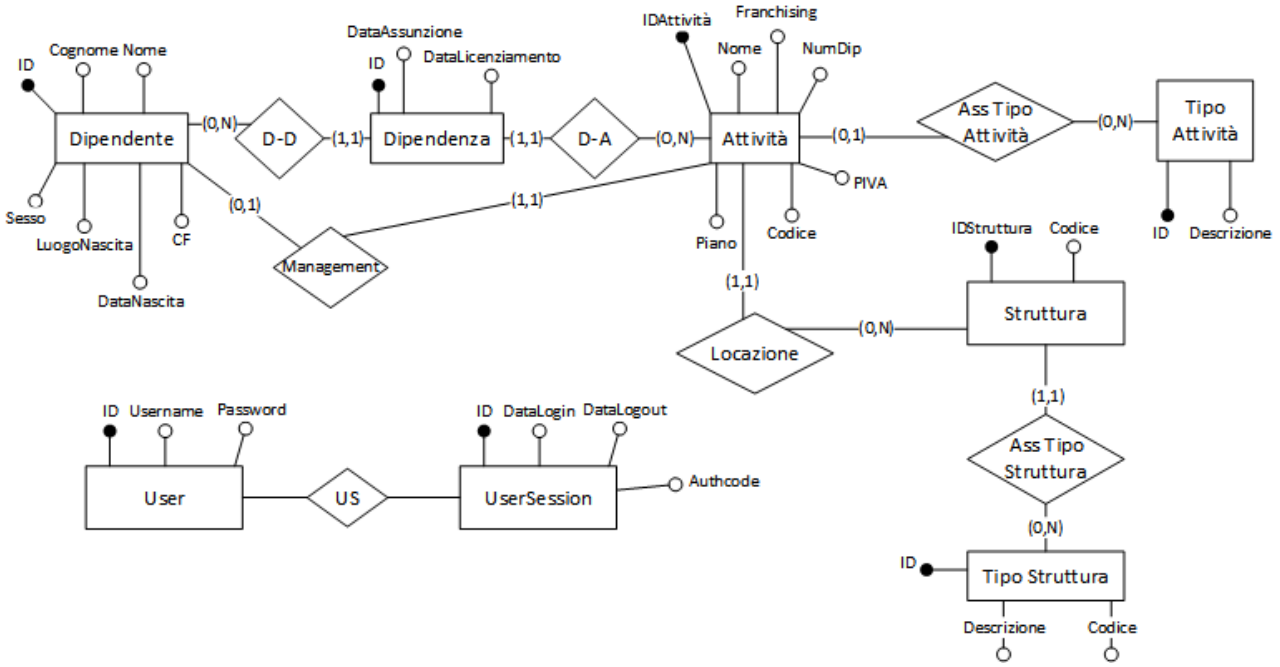
Anche in Tipo Attività e in Dipendenza si genera un ID seriale in maniera automatica, poiché sono entità che possono essere accedute atomicamente.

Per Dipendente l'utilizzo del codice fiscale come chiave primaria viene ritenuto non opportuno, visto che, ai fini applicativi, l'identificatore primario del dipendente sarà inserito nella query string delle richieste al servizio Web.

In generale si è cercato di mantenere l'identificazione degli oggetti la più immediata e compatta possibile.

3.3 Schema ER ristrutturato

A partire dalle analisi fatte lo schema concettuale è stato ristrutturato nel seguente schema ER. Sono state aggiunte, inoltre, due entità e una relazione che servono a gestire la sessione degli utenti che vogliano utilizzare la base di dati dall'apposito applicativo Web.



3.4 Modello logico

Si ottiene il seguente modello logico. Gli identificatori sono sottolineati e gli attributi opzionali sono indicati con un asterisco *.

- dipendente(**id**, cf, nome, cognome, sesso, luogo_nascita*, data_nascita*)
- tipo_struttura(**id**, codice, descrizione)
- struttura(**id_struttura**, codice, id_tipo_struttura)
- tipo_attivita(**id**, descrizione)
- attivita(**id_attivita**, nome, num_dip, piva, codice, franchising, id_struttura*, piano*, id_tipo_attivita*, id_dipendente_manager*)
- dipendenza(**id**, id_dipendente, id_attivita, data_assunzione, data_licenziamento*)
- user(**id**, username, password)
- user_session(**id**, id_user, date_login, date_logout*, authcode)

I vincoli di chiave esterna sono realizzati dai seguenti attributi.

- struttura.id_tipo_struttura è chiave esterna di tipo_struttura.id
- attivita.id_struttura è chiave esterna di struttura.id_struttura
- attivita.id_tipo_attivita è chiave esterna di tipo_attivita.id
- attivita.id_dipendente_manager è chiave esterna di dipendente.id
- dipendenza.id_dipendente è chiave esterna di dipendente.id
- dipendenza.id_attivita è chiave esterna di attivita.id
- user_session.id_user è chiave esterna di user.id

4 Progettazione fisica

Si definiscono gli script SQL utilizzati per la creazione del database. In un primo momento si definiscono i tipi e le tabelle alla base dello schema, quindi vengono specificati gli indici, i trigger e le funzioni della base di dati. Tutti gli script di creazione del database sono presenti nel file `db/script.sql`.

4.1 Tabelle

4.1.1 TipoStruttura

```
CREATE TABLE tipo_struttura
(
  id serial NOT NULL,
  descrizione character varying NOT NULL,
  codice character(1) NOT NULL,
  CONSTRAINT pk_tipo_struttura PRIMARY KEY (id),
  CONSTRAINT cn_tipo_struttura_unique_codice UNIQUE (codice)
)
```

4.1.2 TipoAttivita

```
CREATE TABLE tipo_attivita
(
  id serial NOT NULL,
  descrizione character varying NOT NULL,
  CONSTRAINT pk_tipo_attivita PRIMARY KEY (id)
)
```

4.1.3 Dipendente

```
CREATE TABLE dipendente
(
  cf character varying NOT NULL,
  nome character varying NOT NULL,
  cognome character varying NOT NULL,
  luogo_nascita character varying,
  data_nascita date,
  sesso boolean NOT NULL,
  id integer NOT NULL DEFAULT
nextval('dipendente_id_dipendente_seq'::regclass),
  CONSTRAINT pk_dipendente PRIMARY KEY (id)
)
```

4.1.4 Struttura

```
CREATE TABLE struttura
(
  id_struttura serial NOT NULL,
  codice character varying NOT NULL,
  id_tipo_struttura integer NOT NULL,
  CONSTRAINT pk_struttura PRIMARY KEY (id_struttura),
  CONSTRAINT fk_struttura_tipo_struttura FOREIGN KEY (id_tipo_struttura)
      REFERENCES tipo_struttura (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT cn_struttura_unique UNIQUE (codice, id_tipo_struttura)
)
```

4.1.5 Attivita

```
CREATE TABLE attivita
(
  id_attivita serial NOT NULL,
  nome character varying(20) NOT NULL,
  num_dip integer NOT NULL DEFAULT 0,
  piva character varying(50) NOT NULL,
  codice character varying NOT NULL,
  franchising boolean NOT NULL DEFAULT false,
  id_struttura integer,
  piano integer,
  id_tipo_attivita integer,
  id_dipendente_manager integer,
  CONSTRAINT pk_attivita PRIMARY KEY (id_attivita),
  CONSTRAINT fk_attivita_dipendente_manager FOREIGN KEY (id_dipendente_manager)
      REFERENCES dipendente (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_attivita_struttura FOREIGN KEY (id_struttura)
      REFERENCES struttura (id_struttura) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_attivita_tipo_attivita FOREIGN KEY (id_tipo_attivita)
      REFERENCES tipo_attivita (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

4.1.6 Dipendenza

```
CREATE TABLE dipendenza
(
  id_attivita integer NOT NULL,
  data_assunzione date NOT NULL,
  data_licenziamento date,
  id bigserial NOT NULL,
  id_dipendente integer NOT NULL,
  CONSTRAINT pk_dipendenza PRIMARY KEY (id),
  CONSTRAINT fk_dipendenza_attivita FOREIGN KEY (id_attivita)
      REFERENCES attivita (id_attivita) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_dipendenza_dipendente FOREIGN KEY (id_dipendente)
      REFERENCES dipendente (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT un_dipendenza_dipendente_attivita_assunzione UNIQUE (id_attivita,
data_assunzione, id_dipendente)
)
```

4.1.7 User

```
CREATE TABLE "user"
(
  id serial NOT NULL,
  username character varying NOT NULL,
  password character varying NOT NULL,
  CONSTRAINT pk_user PRIMARY KEY (id),
  CONSTRAINT cn_user_username UNIQUE (username)
)
```

4.1.8 UserSession

```
CREATE TABLE user_session
(
  id bigserial NOT NULL,
  id_user integer NOT NULL,
  date_login timestamp with time zone NOT NULL,
  date_logout timestamp with time zone,
  authcode character varying NOT NULL,
  CONSTRAINT pk_user_session PRIMARY KEY (id),
  CONSTRAINT fk_user_session_user FOREIGN KEY (id_user)
      REFERENCES "user" (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT cn_user_session_authcode UNIQUE (authcode)
)
```

4.2 Indici

4.2.1 Indice b-tree su attivita.nome

È stato realizzato un indice di tipo b-tree sul nome dell'attività poiché la ricerca di un'attività viene fatta proprio sul nome.

```
CREATE INDEX ix_attivita_nome
  ON attivita
  USING btree
  (nome COLLATE pg_catalog."default");
```

4.2.2 Indice b-tree su dipendente.cognome e dipendente.nome

L'indice b-tree sul nome e sul cognome del dipendente è utile in quanto la funzione di ricerca dipendente si basa esclusivamente su questi due campi. È possibile, inoltre, dover fare ricerche solo su uno dei due campi.

```
CREATE INDEX ix_dipendente_cognome_nome
  ON dipendente
  USING btree
  (cognome COLLATE pg_catalog."default", nome COLLATE pg_catalog."default");
```

4.2.3 Indice hash su struttura.codice

Poiché le strutture, come le attività, vengono ricercate in base alla loro descrizione, è necessario impostare un indice che permetta la ricerca e l'ordinamento delle strutture in base a tale campo.

```
CREATE INDEX ix_struttura_codice
  ON struttura
  USING btree
  (codice COLLATE pg_catalog."default");
```

4.2.4 Indice b-tree su tipo_attivita.descrizione

Alcune applicazioni potrebbero voler cercare in maniera dinamica ed immediata le tipologie di attività a partire dal loro nome o dai suoi primi caratteri. Pertanto si rende utile un indice b-tree sul nome della tipologia.

```
CREATE INDEX ix_tipo_attivita_descrizione
  ON tipo_attivita
  USING btree
  (descrizione COLLATE pg_catalog."default");
```

4.2.5 Indice b-tree su tipo_struttura.descrizione

Alcune applicazioni potrebbero voler cercare in maniera dinamica ed immediata le tipologie di struttura a partire dal loro nome o dai suoi primi caratteri. Pertanto si rende utile un indice b-tree sul nome della tipologia.

```
CREATE INDEX ix_tipo_struttura_descrizione
  ON tipo_struttura
  USING btree
  (descrizione COLLATE pg_catalog."default");
```

4.2.6 Indice hash su user.username

Le operazioni di login vengono fatte cercando non valori dell'indice primario, ma valori del campo `username`, e confrontando successivamente la password. È perciò utile un indice di tipo hash sul nome utente, in modo tale da accedere in maniera diretta al blocco di memoria contenente la tupla desiderata.

```
CREATE INDEX ix_user_username
  ON "user"
  USING hash
  (username COLLATE pg_catalog."default");
```

4.2.7 Indice hash su user_session.authcode

Anche per quanto riguarda la sessione si desidera accedere nella maniera più rapida possibile all'authcode, pertanto si crea un indice hash sul campo `authcode`.

```
CREATE INDEX ix_user_session_authcode
  ON user_session
  USING hash
  (authcode COLLATE pg_catalog."default");
```

4.3 Trigger

4.3.1 Gestione della ridondanza sul numero dei dipendenti

Poiché si è scelto, in fase di progettazione logica, di mantenere la ridondanza sul numero dei dipendenti in ogni attività, questa deve essere mantenuta andando a gestire gli inserimenti delle dipendenze, gli aggiornamenti e le cancellazioni.

4.3.1.1 Inserimento

In caso di inserimento, se la data di licenziamento non è già presente (perché magari si vuole inserire l'assunzione per motivi di storicizzazione), si incrementa il numero dei dipendenti dell'attività.

```
CREATE OR REPLACE FUNCTION ins_dipendenza()
  RETURNS trigger AS
$BODY$
BEGIN
-- Se il dipendente non è già licenziato
IF (NEW.data_licenziamento IS NULL) THEN
  -- Incremento il numero dei dipendenti
  UPDATE attivita
  SET num_dip = num_dip + 1
  WHERE attivita.id_attivita = NEW.id_attivita;
END IF;
RETURN NEW;
```

```
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;

CREATE TRIGGER ins_dipendenza
  AFTER INSERT
  ON dipendenza
  FOR EACH ROW
  EXECUTE PROCEDURE ins_dipendenza();
```

4.3.1.2 Cancellazione

Se una dipendenza che viene cancellata non aveva la data di licenziamento impostata, ovvero se il dipendente era ancora assunto, si decrementa il numero dei dipendenti dell'attività.

```
CREATE OR REPLACE FUNCTION del_dipendenza()
  RETURNS trigger AS
$BODY$
BEGIN
-- Se il dipendente non era stato licenziato
IF (OLD.data_licenziamento IS NULL) THEN
  -- Decremento il numero dei dipendenti
  UPDATE attivita
  SET num_dip = num_dip - 1
  WHERE attivita.id_attivita = OLD.id_attivita;
END IF;
RETURN OLD;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;

CREATE TRIGGER del_dipendenza
  AFTER DELETE
  ON dipendenza
  FOR EACH ROW
  EXECUTE PROCEDURE del_dipendenza();
```

4.3.1.3 Aggiornamento

In caso di aggiornamento di una assunzione, se la data di licenziamento è stata modificata, allora si incrementa il numero dei dipendenti se la nuova data è nulla e si decrementa se la nuova data non è nulla.

```
CREATE OR REPLACE FUNCTION upd_dipendenza_licenziamento()
  RETURNS trigger AS
$BODY$
BEGIN
-- Se è cambiata la data di licenziamento
IF (NEW.data_licenziamento IS DISTINCT FROM OLD.data_licenziamento) THEN
  -- Se la nuova data di licenziamento è impostata
  IF (NEW.data_licenziamento IS NOT NULL) THEN
    RAISE INFO 'Decremento il numero dei dipendenti.';
    -- Decremento il numero dei dipendenti dell'attività
    UPDATE attivita
    SET num_dip = num_dip - 1
    WHERE attivita.id_attivita = NEW.id_attivita;
```

```
  ELSE
    RAISE INFO 'Incremento il numero dei dipendenti.';
    -- Altrimenti incremento il numero dei dipendenti dell'attività
    UPDATE attivita
    SET num_dip = num_dip + 1
    WHERE attivita.id_attivita = NEW.id_attivita;
  END IF;
END IF;

RETURN NEW;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;

CREATE TRIGGER upd_dipendenza_licenziamento
  BEFORE UPDATE
  ON dipendenza
  FOR EACH ROW
  EXECUTE PROCEDURE upd_dipendenza_licenziamento();
```

4.3.2 Modifica di una attività o di un dipendente in una dipendenza

Al fine di mantenere registrate tutte le modifiche alle assunzioni (per ragioni di controllo che possono esistere nel centro commerciale), ogni qualvolta che l'attività o il dipendente di una assunzione vengono modificati, viene creata una nuova dipendenza copiando i nuovi valori, e si termina la dipendenza modificata modificando solo la data di licenziamento, prevenendo quindi la modifica sulla tupla di altri valori.

4.3.2.1 Aggiornamento di una dipendenza

```
CREATE OR REPLACE FUNCTION upd_dipendenza()
  RETURNS trigger AS
$BODY$
BEGIN
-- Se l'attività o il dipendente sono cambiati
IF (NEW.id_attivita != OLD.id_attivita OR NEW.id_dipendente !=
OLD.id_dipendente) THEN
  RAISE INFO 'Attività o dipendente modificati.';
  -- Inserisco una nuova dipendenza
  INSERT INTO dipendenza (id_dipendente, id_attivita, data_assunzione,
data_licenziamento)
  VALUES (NEW.id_dipendente, NEW.id_attivita, NEW.data_assunzione,
NEW.data_licenziamento);
  -- Termino la dipendenza corrente impostando la data di licenziamento
  -- SOLO SE la data di licenziamento è nulla
  IF (OLD.data_licenziamento IS NULL) THEN
    UPDATE dipendenza
    SET data_licenziamento = NOW()
    WHERE id_dipendente = OLD.id_dipendente AND id_attivita = OLD.id_attivita;
  END IF;
  -- Restituisco la vecchia riga, che non deve essere modificata
  RETURN OLD;
END IF;
RAISE INFO 'Attività e dipendente non modificati.';
RETURN NEW;
```

```
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;

CREATE TRIGGER upd_dipendenza
  BEFORE UPDATE
  ON dipendenza
  FOR EACH ROW
  EXECUTE PROCEDURE upd_dipendenza();
```

4.4 Funzioni

4.4.1 Cerca dipendente

La funzione riceve in ingresso una stringa che può essere la combinazione, in qualsiasi ordine, di nome e cognome del dipendente, o di una sua parte. Ad esempio, per ricercare Mario Rossi si potrebbero passare in input le stringhe “mar”, “rossi ma”, “mario r”, ecc. Tale funzione si rivela essenziale per gestire in maniera ottimale l'auto-complete di una eventuale ricerca.

In input vengono anche richiesti un valore massimo di record da restituire ed il valore di record da saltare dall'inizio della ricerca (utile per il paging).

```
CREATE OR REPLACE FUNCTION cerca_dipendente(nomecognome character varying, lim
bigint, offs bigint)
  RETURNS SETOF dipendente AS
$BODY$
SELECT * FROM dipendente
WHERE (LOWER(nome || ' ' || cognome) LIKE LOWER('%' || nomecognome || '%'))
OR (LOWER(cognome || ' ' || nome) LIKE LOWER('%' || nomecognome || '%'))
ORDER BY cognome, nome
LIMIT lim OFFSET offs;
$BODY$
  LANGUAGE sql VOLATILE
  COST 100
  ROWS 1000;
```

4.4.2 Conteggio ricerca dipendenti

Ai fini di creare un sistema di paging sull'applicazione Web, si rende necessario ottenere il numero totale dei dipendenti che rispondono al criterio di ricerca.

```
CREATE OR REPLACE FUNCTION conta_cerca_dipendenti(nomecognome character
varying)
  RETURNS integer AS
$BODY$
DECLARE c int;
BEGIN
SELECT COUNT(*) into c FROM dipendente
WHERE (LOWER(nome || ' ' || cognome) LIKE LOWER('%' || nomecognome || '%'))
OR (LOWER(cognome || ' ' || nome) LIKE LOWER('%' || nomecognome || '%'));
RETURN c;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

4.4.3 Esistenza di sessione

La funzione riceva in ingresso una stringa che rappresenta un possibile authcode. Se l'authcode viene ritrovato, la funzione restituisce `true`, altrimenti restituisce `false`.

```
CREATE OR REPLACE FUNCTION esiste_sessione(auth character varying)
  RETURNS boolean AS
$BODY$
DECLARE m_session RECORD;
BEGIN
  SELECT * INTO m_session
  FROM user_session
  WHERE authcode = auth;
  IF NOT found THEN RETURN FALSE;
  ELSE RETURN TRUE;
  END IF;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

5 Servizi Web RESTful

Prima di realizzare l'applicazione Web vera e propria è stato realizzato uno strato di servizi Web di tipo RESTful, basati quindi sul paradigma REST e su un basso accoppiamento fra dati e presentazione. I servizi RESTful sono stati realizzati tramite diverse componenti a più livelli di integrazione:

- **Apache Tomcat** come servlet container;
- **jOOQ** come generatore di entità POJO persistenti su database, per la comunicazione con il database in maniera sicura a compile-time e più efficiente.
- **Jersey** come framework che implementa le specifiche JAX-RS, per la realizzazione dello strato di comunicazione con i client ed esposizione dell'interfaccia del servizio Web.

Il progetto del servizio Web è stato realizzato con Eclipse Juno e JRE 1.6. IL DBMS utilizzato è PostgreSQL 9.2.

5.1 Autenticazione e sicurezza

5.1.1 Login

Sebbene si sia adottato il paradigma REST per la creazione del servizio Web, è stato necessario gestire l'autenticazione memorizzando sul server l'avvenuto login. Il processo di login segue i seguenti passi:

- Il client invia username e password al server, come opzioni per la creazione di una `UserSession`. Per richiedere la creazione di una `UserSession` non serve autorizzazione.
- Il server valida le credenziali e
  - genera un codice di autorizzazione, detto `authcode`, e valido solo per lo specifico utente. L'`authcode` va a comporre la `UserSession`, che contiene i dati dell'utente e parametri di tempo. La sessione viene restituita al client, che risulta loggato e può utilizzare il codice di autorizzazione in tutte le seguenti chiamate.
  - risponde con `401 Unauthorized` se le credenziali non sono valide.

5.1.2 Richieste

L'autenticazione degli utenti è basata su richieste al server e risposte al client:

- Il client invia un codice di autorizzazione al server.
- Il server verifica l'esistenza del codice di autorizzazione e
  - può negare la richiesta, restituendo un codice di errore `401 Unauthorized`;
  - può accettare la richiesta, e proseguire normalmente.

Dal punto di vista implementativo, la classe che si occupa di gestire l'autorizzazione è `AuthenticationResourceFilter`, basata su due classi di Jersey (`ResourceFilter`, `ContainerRequestFilter`), che filtra tutte le richieste prima che queste vengano processate dall'apposito modulo. Dalla richiesta viene estratto il parametro authcode che, se non presente o non valido, genera immediatamente un'eccezione, che scatena la restituzione del codice `401`.

Il filtro è localizzato nel package `net.frapontillo.uni.db2.project.filter`.

5.2 Cross Origin Resource Sharing

Al fine di prevedere l'utilizzo dello strato di servizi da qualsiasi server (e da qualsiasi porta), è stato necessario abilitare il CORS (Cross Origin Resource Sharing), che permette a qualsiasi server di richiedere e modificare risorse esposte dal RESTful Service.

La funzionalità è abilitata inserendo, ad ogni richiesta ricevuta da un client, l'header `Access-Control-Allow-Origin` impostato su `*`, in modo tale da permettere richieste da qualsiasi origine (combinazione di indirizzo e porta). Della gestione CORS si occupa la classe `CorsResponseFilter`, che eredita da `ContainerResponseFilter` e aggiunge gli appositi header ad ogni risposta.

Il filtro è localizzato nel package `net.frapontillo.uni.db2.project.filter`.

5.3 Gestione dei valori nulli

È stata infine creata una classe per la gestione delle risposte vuote: secondo il paradigma REST e l'architettura generale del Web, se una risorsa non esiste, il server che ha ricevuto la richiesta deve restituire come risposta `404 Not Found`. Poiché Jersey non gestisce nella propria implementazione standard tale comportamento, è stato necessario realizzarlo tramite la classe `NullResponseFilter` che, come `ContainerResponseFilter`, gestisce la restituzione di una risposta al client.

Se la risposta ha un corpo nullo, e se la chiamata originale aveva `GET` come metodo, viene rigettata un'eccezione che provoca la restituzione del codice `404`. È possibile, infatti, che alcune risposte generino valori nulli ma che tali valori debbano essere nulli; ad esempio, chiamate di tipo `OPTIONS` o `DELETE` generano necessariamente valori nulli di risposta.

Il filtro è localizzato nel package `net.frapontillo.uni.db2.project.filter`.

5.4 Eccezioni

Come è stato detto, le varie parti dell'applicazione possono rigettare eccezioni in base a diverse condizioni. Queste eccezioni vengono gestite da appositi `ExceptionMapper`, che generano e restituiscono la risposta corretta per il client. Sono stati realizzati diversi mapper che gestiscono eccezioni create ad hoc.

Le eccezioni sono contenute nel package `net.frapontillo.uni.db2.project.exception`, mentre i relativi mapper sono contenuti nel package `net.frapontillo.uni.db2.project.exception.mapper`.

5.4.1 NotFoundException e NotFoundMapper

Una eccezione di classe `NotFoundException` viene rigettata dal modulo di gestione dei valori nulli e viene gestita dal `NotFoundMapper`, che costruisce ed invia al client una risposta di tipo `404 Not Found`.

5.4.2 UnauthorizedException e UnauthorizedMapper

Una eccezione di classe `UnauthorizedException` viene rigettata dal modulo di gestione dell'autenticazione quando l'authcode non è presente o non è valido e dal modulo di gestione della sessione utente quando non sono presenti o non sono validi i dati di login. La risposta restituita al client è di tipo `401 Unauthorized`.

5.4.3 BadRequestException e BadRequestMapper

Una eccezione di classe `BadRequestException` viene rigettata da qualsiasi modulo quando uno dei parametri in ingresso non soddisfa le condizioni attese, ad esempio è obbligatorio o ci si attende un altro formato. La risposta che viene rigettata al client è di tipo `400 Bad Request`.

5.4.4 Altre eccezioni

Le eccezioni non gestite ma rigettate comunque dalle diverse componenti dell'applicazione generano una risposta di tipo `500`, che include nel corpo della risposta lo stack trace della chiamata che ha generato l'eccezione.

5.5 Risorse esposte

Di seguito si elencano le risorse esposte dal servizio Web, con relativi metodi accettati. Le risorse sono contenute nel package `net.frapontillo.uni.db2.project.resource`.

5.5.1 AttivitaResource

`AttivitaResource` (`/api/attivita`) gestisce le attività, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	Attivita a	Preso in ingresso un identificativo, restituisce la corrispondente attività.
GET	String nome String struttura Double page	AttivitaList l	Cerca le attività che hanno un certo nome e che sono in una certa struttura. Gestisce il paging.
POST	Attivita a	Attivita a	Crea l'attività in input e la restituisce al client.
PUT	Integer id Attivita a	Attivita a	Aggiorna l'attività in input e la restituisce al client.
DELETE	Integer id	-	Elimina l'attività identificata dal valore in input.

5.5.2 DipendenteResource

`DipendenteResource` (`/api/dipendente`) gestisce i dipendenti, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	Dipendente d	Preso in ingresso un identificativo, restituisce il dipendente corrispondente.
GET	String nome Double page	DipendenteList l	Cerca i dipendenti che hanno la permutazione di nome e cognome simile al valore in input. Gestisce il paging.
POST	Dipendente d	Dipendente d	Crea il dipendente in input e lo restituisce al client.
PUT	Integer id Dipendente d	Dipendente d	Aggiorna il dipendente in input e lo restituisce al client.
DELETE	Integer id	-	Elimina il dipendente identificato dal valore in input.

5.5.3 DipendenzaResource

`DipendenzaResource` (`/api/dipendenza`) gestisce le assunzioni (dipendenze), esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	Dipendenza z	Preso in ingresso un identificativo, restituisce la dipendenza corrispondente.
GET	Integer d Integer a Double page	DipendenzaList l	Cerca le dipendenze di un dipendente in una struttura (entrambi sono opzionali). Gestisce il paging.
POST	Dipendenza z	Dipendenza z	Crea la dipendenza in input e la restituisce al client.
PUT	Integer id Dipendenza z	Dipendenza z	Aggiorna la dipendenza in input e la restituisce al client.

DELETE	Integer id	-	Elimina la dipendenza identificata dal valore in input.

5.5.4 StrutturaResource

`StrutturaResource` (`/api/struttura`) gestisce le strutture, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	Struttura s	Preso in ingresso un identificativo, restituisce la struttura corrispondente.
GET	String codice Double page	StrutturaList l	Cerca le strutture a partire da un codice in ingresso. Gestisce il paging.
POST	Struttura s	Struttura s	Crea la struttura in input e la restituisce al client.
PUT	Integer id Struttura s	Struttura s	Aggiorna la struttura in input e la restituisce al client.
DELETE	Integer id	-	Elimina la struttura identificata dal valore in input.

5.5.5 TipoAttivitaResource

`TipoAttivitaResource` (`/api/tipoattivita`) gestisce le tipologia di attività, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	TipoAttivita t	Preso in ingresso un identificativo, restituisce la tipologia di attività corrispondente.
GET	String descrizione Integer skip Integer top	TipoAttivitaList l	Cerca le tipologie di attività a partire da una descrizione in ingresso. Gestisce il paging tramite skip (`OFFSET`) e top (`LIMIT`).

5.5.6 TipoStrutturaResource

`TipoStrutturaResource` (`/api/tipostruttura`) gestisce le tipologia di attività, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	TipoStruttura t	Preso in ingresso un identificativo, restituisce la tipologia di struttura corrispondente.
GET	String descrizione Integer skip Integer top	TipoStrutturaList l	Cerca le tipologie di struttura a partire da una descrizione in ingresso. Gestisce il paging tramite skip (`OFFSET`) e top (`LIMIT`).

5.5.7 UserResource

`UserResource` (`/api/user`) gestisce gli utenti, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	Integer id	User u	Preso in ingresso un identificativo, restituisce l'utente corrispondente.

5.5.8 UserSessionResource

`UserSessionResource` (`/api/usersession`) gestisce le sessioni utente, esponendo i seguenti metodi:

Metodo	Input	Output	Descrizione
GET	String authcode	UserSession s	Preso in ingresso l'authcode, restituisce la

			sessione corrispondente.
POST	String username String password	UserSession s	Esegue il login e restituisce la sessione creata.
DELETE	String authcode	-	Esegue il logout.

5.6 Utilizzo del database

Il database è stato utilizzato tramite le classi create da jOOQ. Il progetto rest-gen contiene un file di configurazione e la strategia personalizzata per la creazione delle classi e la gestione dell'ORM.

Le classi ed i metodi creati da jOOQ sono stati utilizzati dalle risorse per l'ottenimento e la modifica dei dati sulla base di dati.

Tutte le classi generate da jOOQ sono contenute nel package `net.frapontillo.uni.db2.project.jooq` e relativi package in esso contenuti.

5.7 Entità e Converter

Una volta ottenute le entità mappate sulle classi di jOOQ, si è reso necessario adottare strategie particolari per restituire tali oggetti al motore di serializzazione in maniera opportuna ed in modo tale che il client abbia tutte le informazioni a disposizione. Infatti, se fossero state restituite le entità così come ottenute dal database avremmo avuto una semplice serializzazione delle tuple, senza possibilità di innestamento di oggetti all'interno di altri.

Sono state create, pertanto, alcune classi entità che sono popolate, a partire da oggetti gestiti da jOOQ, tramite appositi Converter. Questi Converter, in base ad un livello di conversione, generano gerarchie di oggetti in modo tale da rendere disponibile la maggior quantità possibile di informazione utile ad ogni chiamata.

Ad esempio, ritrovando un dipendente si può essere interessati a conoscere tutte le assunzioni che tale dipendente ha. Allo stesso modo, si vorrebbe conoscere anche l'attività in cui è stato assunto per ogni dipendente. Ma una attività mantiene riferimenti anche alle tipologie di attività, e a tutte le altre assunzioni! Per ovviare a possibili e molto probabili eccezioni di stack overflow durante la serializzazione, si è fatto uso dei livelli di conversione. La conversione di un oggetto ad un livello X comporta la serializzazione degli oggetti in esso innestati a livelli inferiori a X, in modo tale da non avere mai loop di conversione.

Tutti i Converter sono contenuti nel package `net.frapontillo.uni.db2.project.converter`. Le entità che vengono serializzate e deserializzate, invece, sono contenute nel package `net.frapontillo.uni.db2.project.entity`.

6 Web Application

Per realizzare l'applicazione Web sono stati utilizzati diversi strumenti e tecnologie:

- Come base dell'applicazione è stato utilizzato un emergente framework MVC lato client, **AngularJS** (di Google). Fortemente basato sul concetto di Dependency Injection, esso permette di utilizzare moduli di codice "al volo", definiti in un qualsiasi altro punto dell'applicazione. Possiede un linguaggio di templating, funzioni automatiche di two-way data binding, moduli di rete, sistemi per la definizione di componenti HTML riutilizzabili, filtri sui dati, direttive HTML, servizi e factories.
- Per la parte grafica si è utilizzato il pacchetto grafico **Bootstrap** (di Twitter), che mette a disposizione stili CSS predefiniti e plugin JavaScript che favoriscono una presentazione di tipo "responsive".
- **bootstrap-datepicker** è un plugin per Bootstrap che permette di utilizzare un datepicker su un tag `input`.
- **angular-resource** è una parte opzionale di AngularJS che espone un service per la definizione e l'utilizzo di risorse REST in maniera immediata, con pieno supporto al CORS e al caching automatico (opzionale).
- **angular-cookies** è un altro modulo opzionale di AngularJS che permette di gestire i cookie nello stesso modo su tutti i browser, eliminando le incongruenze e fornendo un accesso top-level ai cookies tramite un oggetto associativo
- **angular-ui** è un pacchetto aggiuntivo di AngularJS che fornisce ulteriori direttive, filtri e componenti.
- **angular-bootstrap** e **angular-strap** forniscono direttive specifiche per i componenti di Twitter Bootstrap in AngularJS.
- **moment** è una libreria per la gestione delle date in Javascript, che fornisce wrapper di accesso alla data equivalenti per tutti i browser.
- **yeoman** è un tool per la gestione del progetto a livello di dipendenze fra pacchetti, building e testing.
- **jQuery** viene usato (opzionalmente nella versione lite) per la manipolazione del DOM da AngularJS.
- L'IDE scelto per lo sviluppo delle diverse componenti dell'applicazione Web è **JetBrains WebStorm** 5.0.4.

6.1 Componenti MVC

6.1.1 Router

Il Router utilizzato è quello integrato in AngularJS, `$routeProvider`, che permette di far corrispondere, a determinati template URL, un Controller ed una View, che vengono automaticamente iniettati nel contesto applicativo e nel DOM.

Seguendo il paradigma REST anche per l'applicazione Web, ogni URL identifica una particolare risorsa ed azione relativa.

Route template	Controller
/	MainCtrl
/struttura	StrutturaListPageCtrl
/struttura/new	StrutturaNewEditCtrl
/struttura/:id	StrutturaDetailCtrl
/struttura/:id/edit	StrutturaNewEditCtrl
/dipendente	DipendenteListPageCtrl
/dipendente/new	DipendenteNewEditCtrl

/dipendente/:ID	DipendenteDetailCtrl
/dipendente/:ID/edit	DipendenteNewEditCtrl
/attivita	AttivitaListPageCtrl
/attivita/new	AttivitaNewEditCtrl
/attivita/:ID	AttivitaDetailCtrl
/attivita/:ID/edit	AttivitaNewEditCtrl
/assunzione/new	AssunzioneNewEditCtrl
/assunzione/:id	AssunzioneDetailCtrl
/assunzione/:id/edit	AssunzioneNewEditCtrl
/login	LoginCtrl
/logout	LogoutCtrl
/analisi	AnalisiCtrl

6.1.2 Controller

In conformità al paradigma MVC, ogni Controller è associato ad una View e viene iniettato automaticamente in corrispondenza di una View che lo dichiara o di una template URL risolta dal `$routeProvider`.

Ogni Controller ha un proprio oggetto `$scope`, che delimita un "confine" entro il quale i componenti della View possono controllare e modificare gli oggetti. I Controller, comunque, possono essere innestati uno dentro l'altro se le View sono annidate fra di loro. Esiste anche un super-contesto a livello padre, chiamato `$rootScope`.

Ogni Controller ha un modello implicito, definito dagli oggetti nel proprio `$scope`.

6.1.3 View

Le View sono, banalmente, pagine parziali che possono essere iniettate in qualsiasi punto dell'applicazione e che sono fortemente accoppiate ad un Controller che gestisce i dati che esse presentano.

Le view sono presenti nella cartella `views`.

View	Descrizione del contenuto
_header.html	Contiene l'header del sito, costruendo automaticamente il menu in base alla URL corrente.
_buttons_entity_edit.html	Contiene una serie di pulsanti per la modifica di qualsiasi entità. Il Controller che contiene ad un livello superiore questi pulsanti può definire i metodi di implementazione della loro pressione.
_buttons_entity_new.html	Contiene un pulsante per la creazione di qualsiasi entità. Il Controller che contiene ad un livello superiore questo pulsante può definire l'implementazione della sua pressione.
_buttons_entity_save.html	Contiene una serie di pulsanti per il salvataggio di qualsiasi entità. Il Controller che contiene ad un livello superiore questi pulsanti può definire i metodi di implementazione della loro pressione.
_dialog_attivita.html	Definisce una dialog modale che contiene un riferimento alla pagina parziale di ricerca delle attività. La ricerca, quindi, può essere fatta anche all'interno di una dialog.
_dialog_dipendente.html	Definisce una dialog modale che contiene un riferimento alla pagina parziale di ricerca dei dipendenti. La ricerca, quindi, può essere fatta anche all'interno di una dialog.
_dialog_struttura.html	Definisce una dialog modale che contiene un riferimento alla pagina parziale di ricerca delle strutture. La ricerca, quindi, può essere fatta anche all'interno di una dialog.
_select_assunzione.html	Pagina parziale di lista delle assunzioni di un dipendente e/o di una

	attività. Quando viene istanziata si aspetta alcuni parametri a livello superiore, sui quali avvia la ricerca.
_select_attivita.html	Pagina parziale di ricerca e selezione delle attività.
_select_dipendente.html	Pagina parziale di ricerca e selezione dei dipendenti.
_select_struttura.html	Pagina parziale di ricerca e selezione delle strutture.
analisi.html	Pagina di primo livello che contiene un iframe che punta alla pagina di analisi realizzata tramite jPivot.
assunzione_detail.html	Pagina di dettaglio dell'assunzione.
assunzione_edit.html	Pagina di modifica dell'assunzione.
assunzione_list.html	Pagina di lista delle assunzioni. Al momento non è implementata.
assunzione_new.html	Pagina di creazione assunzione.
assunzione_new_edit.html	Pagina parziale che viene utilizzata sia per la creazione che per la modifica di un'assunzione.
attivita_detail.html	Pagina di dettaglio dell'attività.
attivita_edit.html	Pagina di modifica dell'attività.
attivita_list.html	Pagina di ricerca delle attività.
attivita_new.html	Pagina di creazione attività.
attivita_new_edit.html	Pagina parziale che viene utilizzata sia per la creazione che per la modifica di un'attività.
dipendente_detail.html	Pagina di dettaglio del dipendente.
dipendente_edit.html	Pagina di modifica del dipendente.
dipendente_list.html	Pagina di ricerca dei dipendenti.
dipendente_new.html	Pagina di creazione dipendente.
dipendente_new_edit.html	Pagina parziale che viene utilizzata sia per la creazione che per la modifica di un dipendente.
login.html	Pagina di login.
logout.html	Pagina di logout.
main.html	Home page.
struttura_detail.html	Pagina di dettaglio della struttura.
struttura_edit.html	Pagina di modifica della struttura.
struttura_list.html	Pagina di ricerca delle strutture.
struttura_new.html	Pagina di creazione struttura.
struttura_new_edit.html	Pagina parziale che viene utilizzata sia per la creazione che per la modifica di una struttura.

6.2 Comunicazione con il servizio RESTful

La comunicazione con il servizio RESTful è stata realizzata facendo affidamento al modulo `ngResource`: si è definita una Factory per ogni risorsa sul servizio (in `/scripts/services/`), che espone e media la comunicazione fra i Controller ed il modulo `$resource` di AngularJS.

In un certo senso, una risorsa implementata tramite la componente `$resource` può essere vista come una classe di modelli del paradigma MVC.

Le factories create in questo modo sono:

- Attivita
- Dipendente
- Dipendenza
- Struttura
- TipoAttivita
- TipoStruttura
- UserSession

6.3 Factory e servizi aggiuntivi

Per gestire in maniera opportuna il flusso delle informazioni all'interno dell'applicativo Web sono state realizzate altre componenti:

- `AuthHttpInterceptor`, è una tipologia di `$httpInterceptor` che si pone dopo ogni ricezione di risposte dal modulo `$http`, su cui anche `$resource` è basato. Esso controlla che ogni risposta abbia un codice valido. Se il codice di risposta è `401 Unauthorized`, invece, l'interceptor considera l'utente che correntemente sta usando il sistema come non autorizzato, lo disconnette cancellando eventuali cookies e lo reindirizza alla pagina di login.
- `AuthHandler`, è una factory che gestisce le richieste relative alla sessione locale: cancella ed imposta cookie, memorizza e restituisce l'eventuale authcode.
- `Config` è una factory che memorizza i dati relativi alla configurazione del sito Web, ovvero tutte le risorse esterne a cui l'applicativo deve accedere, in modo tale da poterle modificare facilmente in altre situazioni e ambienti. Le configurazioni al momento gestite sono la base URL delle API RESTful e l'indirizzo della pagina di analisi realizzata con jPivot.
- `Menu` è una factory per la gestione degli elementi di menu, in modo tale da costruire un menu semplicemente annidando figli e genitori. Il menu è utilizzato nella vista parziale `__header.html` e in `main.html`.

7 Progettazione del Data Warehouse

Si può supporre di voler integrare i dati amministrativi del centro commerciale, gestiti dall'applicativo Web realizzato, con altri sistemi di gestione della contabilità e degli incassi, magari pre-esistenti, al fine di fornire uno strumento di analisi per gli analisti dei dati e per gli analisti di business.

A causa dell'eterogeneità delle tipologie di attività, e quindi di prodotti venduti e servizi offerti, si potrebbe immaginare di richiedere a tutte le entità del centro commerciale di fornire periodicamente, tramite opportuni sistemi di acquisizione dati, informazioni generiche relative a:

- Incasso
- Attività che ha generato l'incasso
- Data
- Dipendente che ha portato a termine l'acquisto (dove applicabile)

Il sistema di acquisizione a partire dalle informazioni ricevute le integra con i dati presenti nel database di gestione del centro commerciale già realizzato, andando ad aggiungere le seguenti informazioni:

- Tipologia dell'attività
- Struttura nella quale l'attività si trova al momento dell'incasso
- Eventuale manager o proprietario dell'attività, detto "responsabile".

7.1 Progettazione concettuale

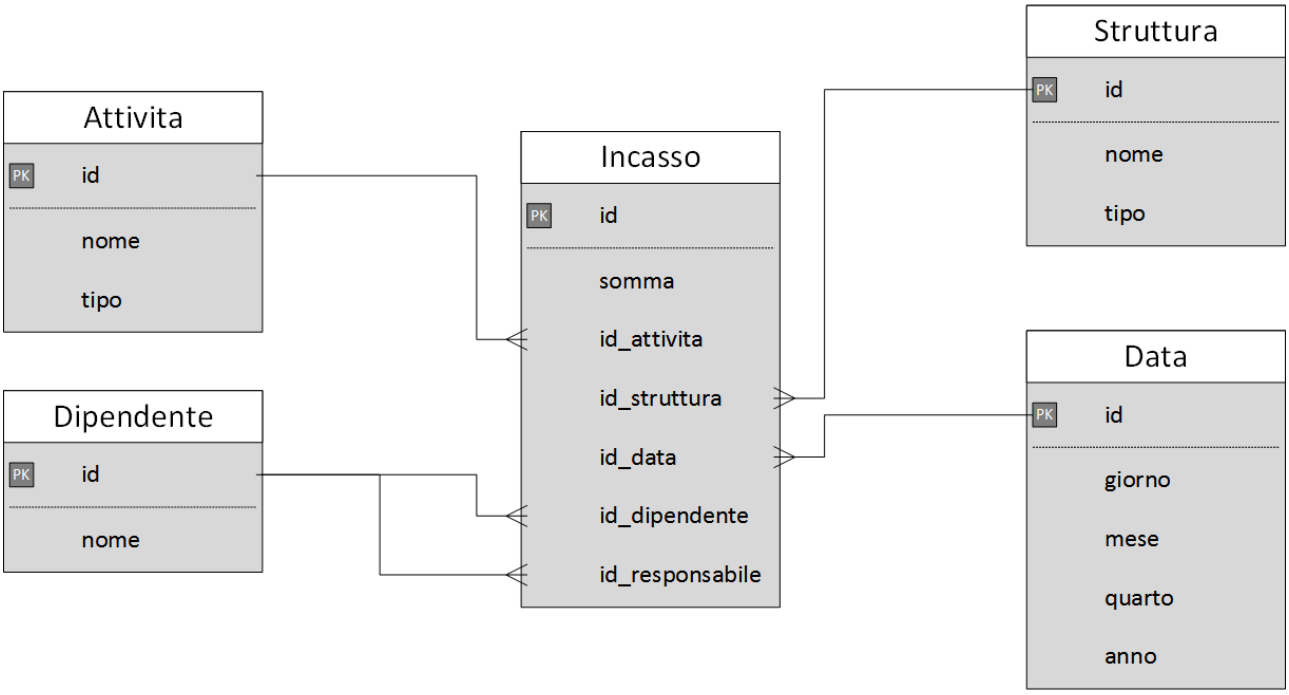
Andiamo a definire lo schema concettuale del Data Warehouse che si vuole creare.

Alla base del DW vi è un cubo multidimensionale così definito:

- Fatto: incasso
- Misure: ammontare dell'incasso
- Dimensioni:
    - Attività
    - Data
    - Dipendente
    - Struttura
    - Responsabile

La tabella dei fatti rappresenta la struttura centrale nello schema a stella, dove le entità connesse da opportune relazioni rappresentano le dimensioni di analisi:

- Attività, contiene le informazioni sulle attività che generano gli incassi.
- Data, contiene le date relative agli incassi, con possibilità di realizzare gerarchie su giorno, mese, trimestre, anno.
- Dipendente, contiene le informazioni sui dipendenti che hanno concluso una vendita o realizzato un servizio, dove applicabile.
- Struttura, contiene la struttura in cui si trova l'attività al momento dell'incasso. Una attività, infatti, potrebbe cambiare struttura.
- Responsabile, contiene il manager o il proprietario (informazioni opzionali) dell'attività al momento dell'incasso.



Sarebbe stato possibile introdurre un'ulteriore collegamento fra Attività e tipo di attività. Tuttavia si è scelto di optare verso una soluzione che garantisca migliori prestazioni a discapito del piccolo spreco di spazio.

7.2 Progettazione logica

Lo schema a stella/a costellazione è stato tradotto in un opportuno schema logico relazionale in modo tale da poterlo implementare tramite il DBMS di destinazione scelto, PostgreSQL 9.2.

- attivita(**id**, nome, tipo)
- dipendente(**id**, nome)
- struttura(**id**, nome, tipo)
- data(**id**, giorno, mese, quarto, anno)
- incasso(**id**, somma, id_attivita, id_struttura, id_data, id_dipendente, id_responsabile)

I vincoli di chiave esterna sono i seguenti:

- incasso.id_attivita è chiave esterna di attivita.id
- incasso.id_struttura è chiave esterna di struttura.id
- incasso.id_data è chiave esterna di data.id
- incasso.id_dipendente è chiave esterna di dipendente.id
- incasso.id_responsabile è chiave esterna di dipendente.id

7.3 Implementazione

Si definiscono gli script SQL utilizzati per la creazione del DW, presenti nel file `dw/script.sql` (il file contiene già molti dati di esempio generati in maniera casuale). È stato utilizzato il pacchetto open source Mondrian per la definizione e l'utilizzo del cubo multidimensionale realizzato tramite il DW qui presentato.

```
CREATE TABLE attivita
(
  id bigserial NOT NULL,
  nome character varying(20) NOT NULL,
  tipo character varying(20),
```

```
  CONSTRAINT pk_attivita PRIMARY KEY (id)
)

CREATE TABLE data
(
  id bigserial NOT NULL,
  giorno character varying(2),
  mese character varying(2),
  quarto character varying(1),
  anno character varying(4),
  CONSTRAINT pk_data PRIMARY KEY (id)
)

CREATE TABLE dipendente
(
  id bigserial NOT NULL,
  nome character varying(20) NOT NULL,
  CONSTRAINT pk_dipendente PRIMARY KEY (id)
)

CREATE TABLE struttura
(
  id bigserial NOT NULL,
  nome character varying(20) NOT NULL,
  tipo character varying(20),
  CONSTRAINT pk_struttura PRIMARY KEY (id)
)

CREATE TABLE incasso
(
  id bigserial NOT NULL,
  somma real NOT NULL DEFAULT 0,
  id_attivita bigint NOT NULL,
  id_struttura bigint NOT NULL,
  id_data bigint NOT NULL,
  id_dipendente bigint,
  id_responsabile bigint,
  CONSTRAINT pk_incasso PRIMARY KEY (id),
  CONSTRAINT fk_attivita FOREIGN KEY (id_attivita)
      REFERENCES attivita (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_data FOREIGN KEY (id_data)
      REFERENCES data (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_dipendente FOREIGN KEY (id_dipendente)
      REFERENCES dipendente (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_responsabile FOREIGN KEY (id_responsabile)
      REFERENCES dipendente (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_struttura FOREIGN KEY (id_struttura)
      REFERENCES struttura (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

7.4 Schema e analisi dei dati

È stato utilizzato il software Schema Workbench per creare lo schema multidimensionale a partire dal DW di tipo relazionale. È stato creato un file XML che definisce il cubo, la tabella dei fatti con misure, le dimensioni e le relative gerarchie. È possibile ritrovare l'XML al percorso `dw/mallSchema.xml`.

È possibile esplorare ed analizzare il cubo tramite la pagina `mallAnalysis.jsp`. Tale pagina presenta, nel tag `jp:mondrianQuery`, la seguente query MDX:

```
SELECT {[Measures].[incasso]} ON COLUMNS,
{([data],[attivita],[struttura],[dipendente],[responsabile])} ON ROWS
FROM [incassi]
```

La visualizzazione della tabella di jPivot è possibile copiando la pagina fra le query di jPivot, e passando il parametro alla `testPage.jsp`. Alternativamente, è possibile visualizzare il grafico dall'applicativo Web, che contiene un `iframe` che punta alla stessa pagina.