Progettazione di una base di dati relazionale per un software di planning aziendale

Autori: Andrea Lemmo N86003262, Davide Soldatini N86002862, Michele Zurlo N86003219

Data Creazione: 13/12/2020

Indice

1 Introduzione

Di seguito intendiamo introdurre brevemente il problema posto e lo scopo della base che progetteremo attraverso l'intero documento.

1.1 Descrizione del problema

La traccia ricevuta *(n.2)* era testualmente la seguente:

> *Si sviluppi un sistema informativo, composto da una base di dati relazionale e un applicativo Java dotato di GUI (Swing o JavaFX), per la gestione di progetti in un'azienda. Si tenga traccia dei partecipanti al progetto, identificando i ruoli per ognuno di essi (per ogni progetto ci sarà solo un project manager). Ad ogni progetto è associato una tipologia ("Ricerca di base", "Ricerca Industriale", "Ricerca sperimentale", "Sviluppo Sperimentale", ...) ed uno o più ambiti (Economia, Medicina, …). Il sistema dovrà permettere anche l'organizzazione di meeting fisicamente, in sale riunioni, o telematicamente su una piattaforma di videoconferenza. Si dovrà tenere traccia delle partecipazioni ai progetti ed ai meeting, ai fini della valutazione del singolo partecipante. In fase di creazione di un nuovo progetto, i partecipanti dovranno essere selezionati in base a criteri di ricerca che includono anche il salario medio e la valutazione aziendale del partecipante, oltre alla tipologia di progetti cui ha preso parte. Ad ogni partecipante sarà associata una lista di skill. Inoltre, in fase di creazione di un nuovo progetto, i partecipanti potranno essere scelti in funzione anche delle loro skill. In fase di registrazione di un partecipante, inserire le skill e se non presente nel DB, crearne una nuova.*

Da questo testo possiamo immediatamente individuare le entità chiave del problema che andranno in qualche modo a costituire in seguito le tabelle della nostra base di dati. Sono abbastanza evidenti anche le relazioni che intercorrono tra le principali entità del problema. Ad esempio l'entità **dipendente** deve essere in una qualche relazione con l'entità **progetto**, dato che ogni dipendente può partecipare a dei progetti.

Questo era solo un accenno dell'analisi che verrà fatta di seguito nella fase di progettazione concettuale della base, dove avremo lo scopo di schematizzare in modo ancora abbastanza astratto ma molto intuitivo lo scheletro della nostra base.

2 Progettazione concettuale

In fase di progettazione concettuale dobbiamo analizzare nel dettaglio il testo per individuare tutte le componenti fondamentali della nostra futura base e le loro associazioni allo scopo di creare un Class Diagram iniziale completo che la descriva. A questo primo diagramma seguirà una fase di ristrutturazione dello stesso allo scopo di trasformare uno schema astratto come quello in uno schema sempre abbastanza astratto ma anche compatibile con il modello di dati relazionale. Infine per descrivere quelle informazioni che non compaiono per loro natura nel Class Diagram, come ad esempio i vincoli sulla formattazione di un certo dato, stileremo anche i dizionari delle classi, delle associazioni e dei vincoli.

Se si intende saltare tutta la parte discorsiva sull'individuazione di entità e associazioni e analizzare direttamente il class diagram risultante da questa prima analisi, si può andare al capitolo 2.3 Class Diagram.

2.1 Individuazione delle entità

Di seguito individueremo le entità fondamentali che compaiono nel problema e di cui è importante tenere traccia nella base. Ognuna di esse sarà definita da degli attributi che renderanno ogni loro istanza unica.

2.1.1 Classe Progetto

Una prima entità importantissima è quella del **Progetto**. Un progetto aziendale sarà ovviamente caratterizzato da un nome *(NomeProgetto)* e da una sua breve descrizione *(DescrizioneProgetto)*. Inoltre, come suggerisce la traccia, ogni progetto appartiene a uno o più ambiti possibili *(AmbitoProgetto)* ed è di un tipo specifico di ricerca *(TipoProgetto)*. Trattandosi di un software per planning aziendale, supponiamo sia importante avere anche delle funzionalità aggiuntive come la possibilità di creare nuovi progetti, di terminarli e magari anche di definire una data di scadenza degli stessi. Per questo motivo possiamo aggiungere altri tre attributi alla classe Progetto rappresentanti la data di creazione del progetto *(DataCreazione)*, la data di scadenza del progetto *(Scadenza)* e la data di eventuale terminazione del progetto *(DataTerminazione)*. Sebbene questi attributi siano sufficienti a definire una specifica istanza di un progetto, preferiamo individuare le singole istanze attraverso un ulteriore attributo unico che sarà un codice identificativo specifico del progetto *(CodProgetto)*.

Dopo questa analisi, possiamo schematizzare la classe Progetto nel seguente modo:

Progetto
CodProgetto (codice identificativo)
NomeProgetto (nome del progetto)
DescrizioneProgetto (descrizione breve del progetto)
TipoProgetto (tipo di ricerca che rappresenta il progetto)
AmbitoProgetto (ambiti di applicazione del progetto)
DataCreazione (data di creazione del nuovo progetto)
Scadenza (data di eventuale scadenza prevista del progetto)
DataTerminazione (data di terminazione del progetto)

2.1.2 Classe Dipendente

Un'altra entità essenziale del nostro problema è il dipendente dell'azienda che rappresenteremo con la classe **Dipendente**. Un dipendente sarà chiaramente dotato di alcuni dati anagrafici essenziali come nome *(Nome)*, cognome *(Cognome)*, sesso *(Sesso)*, data di nascita *(DataNascita)* e luogo di nascita *(LuogoNascita)*. Questi ci saranno in seguito necessari per generare automaticamente il corretto codice fiscale *(CF)* del dipendente, che sarà utilizzato anche come identificativo di ogni singola istanza della classe. Abbiamo supposto che al fine di una migliore pianificazione dei meeting fosse utile tracciare anche l'indirizzo di residenza attuale del dipendente *(Indirizzo)*, i contatti telefonici da lui forniti *(Telefono)* e l'email aziendale a lui assegnata con cui contattarlo e anche con cui può accedere al software *(Email)*. In seguito come richiesto dalla traccia ricevuta bisogna tenere conto anche del salario attuale del dipendente *(Salario)* e di

una sua valutazione generale *(Valutazione)*. Infine affinchè il dipendente sia in grado di accedere alla base e di operare con il software, è necessario che sia dotato di alcune credenziali. In funzione dell'username verrà usata la stessa email aziendale già fornita e come password *(Password)* una a sua scelta in fase di creazione dell'account.

In conclusione possiamo riassumere la classe Dipendente nel seguente modo:

Dipendente
CF (codice fiscale generato a partire dai dati anagrafici forniti)
Nome (nome del dipendente)
Cognome (cognome del dipendente)
Sesso (sesso del dipendente)
DataNascita (data di nascita del dipendente)
LuogoNascita (luogo di nascita del dipendente)
Indirizzo (indirizzo di residenza attuale del dipendente)
Telefono (contatti telefonici forniti dal dipendente)
Email (email aziendale a cui è collegato l'account del dipendente)
Password (password di accesso al software per il dipendente)
Salario (salario attuale del dipendente)
Valutazione (valutazione aziendale del dipendente)

2.1.3 Classe Meeting

Il software ha come altro scopo principale quello di pianificare meeting aziendali e quindi risulta chiaro che un'altra importantissima entità da descrivere è quella del **Meeting**. Un meeting sarà ovviamente caratterizzato da una data di inizio *(DataInizio)*, un orario di inizio *(OrarioInizio)*, una data di fine *(DataFine)* e un orario di fine *(OrarioFine)*. La traccia specifica in seguito che un meeting può essere considerato sia fisico che telematico attraverso varie piattaforme. Di conseguenza un ulteriore attributo che definisce il tipo di meeting *(Modalità)* è necessario per distinguere le riunioni fisiche da quelle telematiche e un altro serve invece a tenere traccia dell'eventuale piattaforma *(Piattaforma)* usata per i meeting telematici. Non abbiamo specificato un altro attributo per tracciare anche le sale riunioni occupate dai meeting fisici, perché per una migliore gestione di tali informazioni abbiamo preferito introdurre una entità a parte per esse. Infine per individuare le singole istanze dei meeting abbiamo preferito introdurre un attributo ulteriore "artificiale" che funga da indice identificativo del meeting *(IDMeeting)*.

Meeting
IDMeeting (codice identificativo del meeting)
DataInizio (data di inizio del meeting)
OrarioInizio (orario dell'inizio del meeting)
DataFine (data della fine del meeting)
OrarioFine (orario della fine del meeting)
Modalità (modalità di incontro del meeting, fisico o telematico)
Piattaforma (eventuale piattaforma su cui avviene il meeting telematico)

2.1.4 Classe Sala Riunione

Come anticipato per una migliore gestione delle prenotazioni delle sale riunioni per i meeting fisici è utile introdurre un'ulteriore entità **SalaRiunione**. Una sala riunione è univocamente determinata da un suo codice identificativo *(CodSala)*. Una sala riunione ha in genere una capienza limitata *(Capienza)* che non può essere superata, quindi terremo traccia anche di questo fattore. Data la possibilità che l'azienda preveda più sedi sparse, una sala si troverà in una specifica sede con un proprio indirizzo *(Indirizzo)* di cui terremo traccia. Infine supponendo che l'azienda abbia sedi dotate di almeno due piani, torna utile tenere traccia anche del piano *(Piano)* in cui si trova la sala riunione.

In conclusione possiamo riassumere la classe SalaRiunione nel seguente modo:

SalaRiunione
CodSala (codice identificativo della sala)
Capienza (limite di posti disponibili della sala)
Indirizzo (indirizzo della sede in cui si trova la sala)
Piano (piano della sede in cui si trova la sala)

2.1.5 Classe Skill

Quasi alla fine della traccia ricevuta viene aggiunta un'ulteriore informazione di cui tenere traccia che sarà strettamente collegata concettualmente al dipendente, ovvero le sue skill lavorative che rappresenteremo come entità **Skill**. Preferiamo che siano rappresentate come classe associata al dipendente piuttosto che come suo attributo per avere un maggior controllo sulle skill di tutti i dipendenti senza perdere la possibilità di associare a ciascun dipendente le sue skill.

Una skill è chiaramente definita da un nome breve *(NomeSkill)* e da un identificativo (*IDSkill)* che ci permetta di distinguere univocamente le istanze di ciascuna di esse.

In conclusione possiamo riassumere la classe Skill come segue:

Skill
IDSkill (codice identificativo della skill)
NomeSkill (nome della skill)

2.2 Individuazione delle associazioni

Abbiamo individuato e analizzato le singole entità che compongono il problema e a partire da esse abbiamo definito delle classi che le rappresentassero. Adesso vogliamo individuare le associazioni che intercorrono tra di esse.

2.2.1 Associazione Progetto-Dipendente

Dalla traccia si evince chiaramente che i dipendenti sono assegnati a dei progetti aziendali e che quindi ogni progetto aziendale prevede dei partecipanti. Di conseguenza definiamo un'associazione tra queste due classi **Dipendente** e **Progetto** che chiameremo **Partecipazione**. Questa associazione quindi descrive la partecipazione dei dipendenti ai progetti aziendali. Il ruolo del dipendente è quindi quello di partecipare al progetto (*partecipa*), mentre quello del progetto è di comprendere diversi dipendenti che ci lavorano su *(comprende)*.

Dato che un'azienda può avere anche dipendenti temporaneamente inattivi, ovvero che non partecipano ad alcun progetto, il numero minimo di progetti a cui un dipendente partecipa è 0. Ovviamente un progetto può prevedere (anzi di norma ci si aspetta sia così) più dipendenti, quindi non è previsto un tetto massimo di dipendenti partecipanti ad un progetto.

Ovviamente un'azienda può anche avere temporaneamente progetti in corso a cui nessuno sta lavorando, un esempio è un progetto appena creato. Tuttavia anche in questo caso almeno il creatore del progetto risulterà partecipante al progetto. Questo vuol dire che anche il numero minimo di dipendenti occupati in un singolo progetto è 1. Siccome poi non è stabilito alcun limite sul numero di dipendenti occupati in un progetto, non esiste un tetto massimo di dipendenti compresi in un progetto.

In conclusione l'associazione Partecipazione tra Dipendente e Progetto è un'associazione *molti-a-molti*.

Un'ulteriore nota su questa associazione segue da una specifica fatta nella traccia, che prevede che ogni dipendente che partecipa a un progetto deve assumere un ruolo specifico, tra cui quello di project manager che deve essere unico per ogni progetto. Per descrivere questa informazione, abbiamo optato per una classe d'associazione **RuoloDipendente** con singolo attributo il nome del ruolo *(RuoloDip)* che vincola l'associazione a specificare un ruolo per il dipendente che partecipa al progetto.

2.2.2 Associazione Dipendente-Meeting

Ogni dipendente può chiaramente presenziare a un meeting, dunque risulta inevitabile definire un'associazione anche tra le classi definite prima **Dipendente** e **Meeting**. La chiameremo **Presenza**. Questa associazione descrive quindi la partecipazione di un dipendente al meeting *(presente)* e il gruppo di dipendenti inclusi in un meeting *(include)*.

Data l'istanza di uno specifico meeting, è richiesto che almeno un dipendente sia presente affinchè il meeting abbia motivo di essere organizzato, ad esempio il suo organizzatore. Di conseguenza il numero

minimo di dipendenti inclusi in un meeting deve essere 1. Siccome poi non c'è un limite al numero di dipendenti che possono essere inclusi nel meeting (se non per ulteriori vincoli esterni dettati dalla capienza di un'eventuale sala riunione), il tetto massimo di dipendenti inclusi nel meeting non è specificato.

Data invece l'istanza di uno specifico dipendente, questo può presumibilmente sia non essere presente in alcun meeting sia essere richiesto in più meeting esistenti. Quindi il numero minimo di meeting a cui un dipendente può essere legato è 0, mentre non esiste un tetto massimo di meeting in cui egli possa essere richiesto.

Inoltre per tenere traccia dell'effettiva presenza avvenuta del dipendente al meeting, abbiamo aggiunto una classe d'associazione **Frequenza** con un attributo boolean che ci dice se tale dipendente è stato presente o meno al meeting di riferimento. Questo ci permetterà in seguito di calcolare la valutazione del dipendente.

Invece per tenere traccia del dipendente che ha organizzato il meeting la classe d'associazione **Frequenza** ha un altro attributo boolean che sarà vero se il dipendente partecipante al meeting è l'organizzatore oppure falso se è un semplice invitato.

In conclusione anche questo associazione Presenza risulta un'associazione *molti-a-molti*.

2.2.3 Associazione Progetto-Meeting

Ogni meeting viene chiaramente organizzato allo scopo di discutere di uno specifico progetto. Di conseguenza risulta evidente la necessità di definire un'associazione anche tra le classi **Progetto** e **Meeting**. Questa associazione la chiameremo **Discussione**. In pratica questa associazione descrive il progetto discusso nel meeting *(discusso)* e anche i meeting relativi ad uno specifico progetto *(relativo a)*.

Data una specifica istanza di un progetto, è chiaro che il numero di meeting relativi ad essa può essere tanto 0 (ovvero non esistono meeting che ne discutono) quanto una moltitudine. Quindi non è definito un tetto massimo di possibili meeting relativi a uno specifico progetto.

Data una specifica istanza di un meeting, è ovvio che il numero di progetti discussi in essa deve essere sempre 1.

Di conseguenza quest'associazione così definita è un'associazione *uno-a-molti*.

2.2.4 Associazione Meeting-SalaRiunione

I meeting fisici vengono ospitati in sale riunioni disparse tra le varie sedi aziendali. Questo vuol dire che terremo traccia di questo legame tra sale e meeting attraverso un'altra associazione tra le classi **Meeting** e **SalaRiunione**. La chiameremo **Riunione**. Questa associazione in pratica descrive la sala riunioni che ospita il meeting *(in)* e i meeting che sono ospitati nelle varie sale riunioni *(ospita)*.

Data una specifica istanza di un meeting, questa può essere ospitata al più da una sola sala riunione qualora si trattasse di un meeting fisico. Di conseguenza il numero minimo di sale che ospitano uno specifico meeting è 0 (ad esempio quando il meeting è telematico), mentre il numero massimo è 1.

Data una specifica istanza di sala riunione, questa può sia essere libera e quindi ospitare 0 meeting sia ospitare più riunioni. Di conseguenza non è possibile definire un tetto massimo di meeting ospitati da una sala riunione.

In conclusione quest'associazione Riunione è un'associazione *uno-a-molti*.

2.2.5 Associazione Dipendente-Skill

Come specificato nell'ultima sezione della traccia ricevuta, ogni dipendente può avere delle skill lavorative di cui tenere traccia. Per tenere traccia delle skill associate a ciascun dipendente introduciamo allora un'ulteriore associazione tra **Dipendente** e **Skill**. La chiameremo **Abilità**. Questa associazione descrive in pratica le skill possedute da un dipendente *(ha)* e i dipendenti che hanno una specifica skill *(posseduta)*.

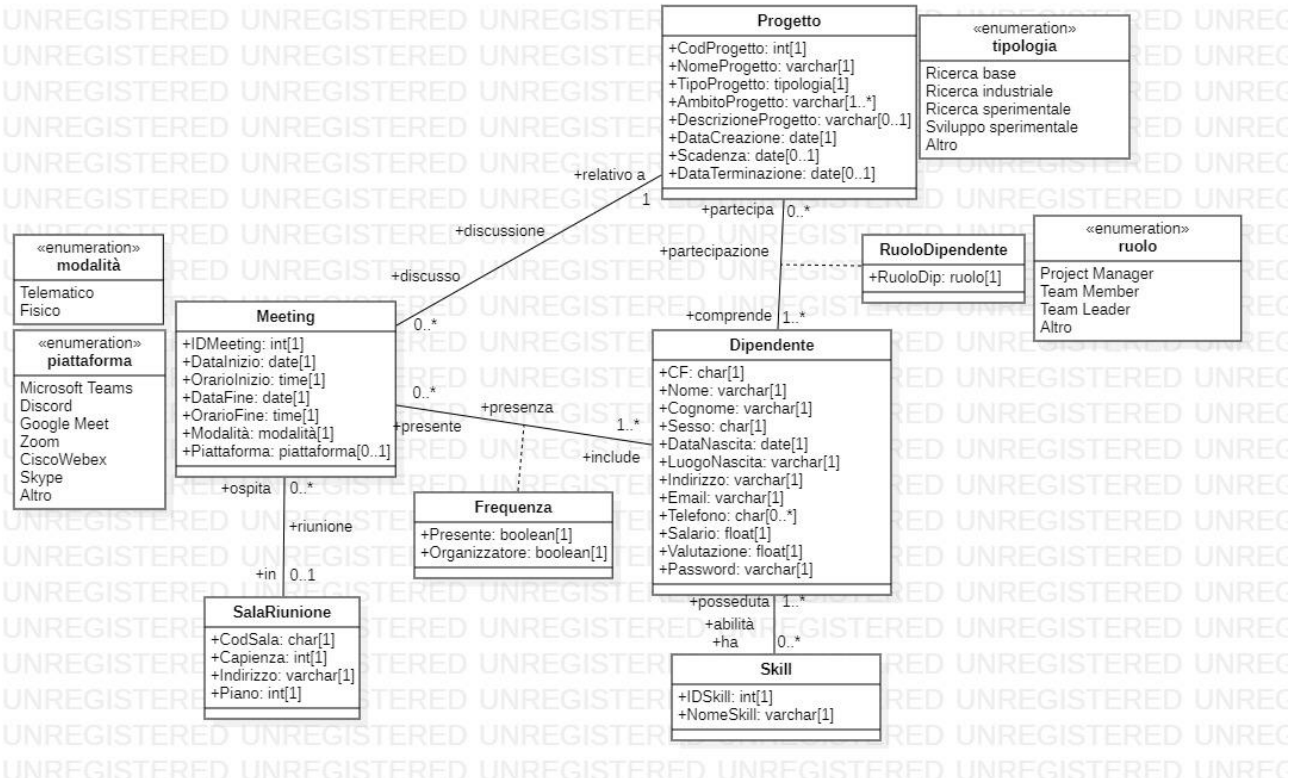
Data una specifica istanza di skill lavorativa, questa deve essere posseduta da almeno 1 dipendente affinchè abbia senso tenerne conto nella base di dati. Ovviamente può essere posseduta anche da una moltitudine di dipendenti, quindi il tetto massimo di dipendenti associati a una skill non è definito.

Data una specifica istanza di dipendente, questa può tanto essere priva di skill quanto averne una moltitudine. Di conseguenza il numero minimo di skill associate a un dipendente è 0, mentre il numero massimo non è definito.

In conclusione l'associazione Abilità è un'associazione *molti-a-molti*.

2.3 Class Diagram

Alla fine di tutte queste analisi possiamo allora stilare un class diagram che schematizzi tutte le osservazioni fatte fino ad ora.



2.4 Ristrutturazione del Class Diagram

In questa fase della progettazione intendiamo ristrutturare il class diagram risultato dalle prime osservazioni per renderlo compatibile con le restrizioni del modello di dati relazionale, che sarà quello adottato in questo caso per la creazione della base di dati. Dovremo evitare inutili ridondanze di informazioni tra associazioni e attributi di classe, cercare di definire chiavi primarie semplici e chiare per

rendere più efficiente la base e la sua interrogazione, dovremo gestire eventuali attributi a valori multipli che non sono previsti nello schema relazionale, dovremo eliminare qualsiasi specializzazione e definire un modo di memorizzazione dei dati strutturati.

Se si intende saltare tutte le analisi svolte e passare direttamente al class diagram risultante dalle modifiche svolte, si passi al capitolo 2.4.6 Class Diagram ristrutturato.

2.4.1 Analisi delle ridondanze

Nel class diagram l'unica ridondanza evidente è quella relativa alla valutazione del dipendente. Abbiamo infatti sia un attributo di dipendente **Valutazione** che conserva tale dato, sia un sistema per calcolare lo stesso dato attraverso un'interrogazione combinata tra **Meeting** e **Dipendente** con **Frequenza** e **Progetto** con **Dipendente**. Per evitare problemi di inconsistenza di tale informazione abbiamo preferito eliminare l'attributo e calcolare la valutazione del dipendente ogni volta che è richiesta.

2.4.2 Analisi delle chiavi primarie

Tutte le entità definite nel class diagram presentano già chiavi primarie semplici ed efficienti. Infatti di seguito elencheremo per ogni classe la loro chiave:

- **Dipendente** (CF)
- **Progetto** (CodProgetto)
- **Meeting** (IDMeeting)
- **SalaRiunione** (CodSala)
- **Skill** (IDSkill)

Come si può osservare tutte le chiavi primarie sono composte da singoli attributi di tipo integer o char, quindi dai valori anche rapidamente confrontabili.

2.4.3 Analisi degli attributi multipli

La classe **Progetto** prevede un attributo a valori multipli **AmbitoProgetto**. Per eliminare questo attributo senza perdere l'informazione che ci interessa, abbiamo optato per la definizione di un'ulteriore classe **AmbitoProgetto**. La classe ha come attributi un codice identificativo che individua univocamente ogni istanza di ambito *(IDAmbito)* e il suo nome *(NomeAmbito)*. Ovviamente dalla classe Progetto è stato eliminato l'attributo AmbitoProgetto e abbiamo introdotto un'associazione tra **AmbitoProgetto** e **Progetto**. Questa descrive gli ambiti di appartenenza di uno specifico progetto *(di)* e l'ambito dei vari progetti *(ha)*. L'associazione l'abbiamo chiamata **Ambito** ed è un'associazione *molti-a-molti*. Infatti data una specifica istanza di ambito di un progetto, questa potrebbe essere associata a 0 progetti oppure a una moltitudine di loro. Viceversa, data una specifica istanza di un progetto, questo deve avere almeno un ambito, ma può averne più di uno.

La classe **Dipendente** anche prevede un attributo a valori multipli, ovvero **Telefono**. Questo rappresenta i vari contatti telefonici che un dipendente può inserire nelle sue informazioni. Mentre prima avere una classe a sé per l'attributo a valori multipli era la scelta migliore per mantenere il controllo sugli ambiti trattati dall'azienda, in questo caso risulta più efficiente (a nostro dire) la scomposizione dell'attributo in due attributi singoli. Questo perché mediamente una persona offre come contatti telefonici il proprio numero di cellulare privato e il numero di casa. Allora abbiamo eliminato l'attributo **Telefono** e al suo posto abbiamo messo due attributi a valori singoli **Cellulare** e **TelefonoCasa**.

2.4.4 Analisi delle gerarchie

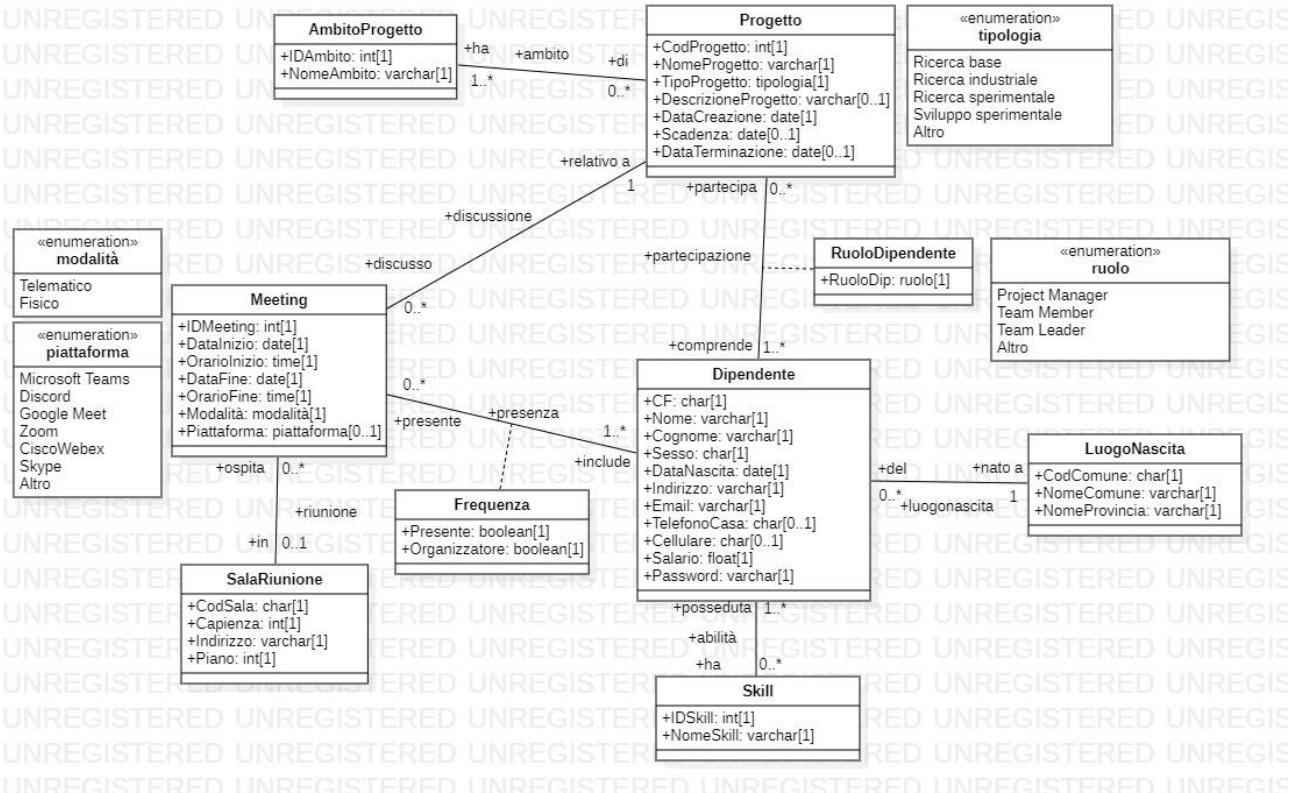
Nel class diagram non compaiono specializzazioni e generalizzazioni, quindi non sono avvenute modifiche di alcun tipo a riguardo.

2.4.5 Analisi dei dati strutturati

L'unica classe che presenta un attributo strutturato è **Dipendente**. Il dato in questione è **LuogoNascita**. Questo perché, affinchè si possa generare il codice fiscale del dipendente, è necessario distinguere comune e provincia in cui egli è nato. Dato però che per questa generazione del codice fiscale è necessario nella pratica avere il codice identificativo del comune, piuttosto che scomporre semplicemente LuogoNascita in due attributi semplici, abbiamo preferito definireuna classe aggiuntiva **LuogoNascita** con attributi il codice del comune *(CodComune)*, il nome del comune in cui è nato *(NomeComune)* e il nome della provincia a cui tale comune appartiene *(NomeProvincia)*. Abbiamo poi definito un'associazione **Dipendente-LuogoNascita** che descrive il luogo di nascita di ciascun dipendente *(del)* e tutti dipendenti nati in uno specifico comune *(nato a)*. Abbiamo chiamato quest'associazione **LuogoNascita**. Questa è un'associazione *uno-a-molti*, infatti ogni dipendente può essere nato in un solo comune e ogni comune può essere il luogo di nascita di più dipendenti.

2.4.6 Class Diagram ristrutturato

Al termine di queste osservazione e delle relative modifiche, mostriamo di seguito il class diagram risultante.



2.5 Dizionari

Una volta stabilito che questo è il class diagram di riferimento per la futura progettazione logica della base di dati, è opportuno definire anche nel dettaglio le classi, le associazioni e i vincoli non visibili dal diagramma in opportuni dizionari riassuntivi. Definiremo quindi dizionari di classi, di associazioni e di vincoli qui di seguito.

2.5.1 Dizionario delle classi

Classe	Descrizione	Attributi
Progetto	Descrittore di ogni possibile progetto dell'azienda.	**CodProgetto** *(int)* codice identificativo del progetto. **NomeProgetto** *(varchar)* nome del progetto. **TipoProgetto** ***(tipologia)*** tipo di ricerca svolta dal progetto. **DescrizioneProgetto** *(varchar, opzionale)* descrizione breve degli obiettivi del progetto. **DataCreazione** *(date)* data di creazione del progetto **Scadenza** *(date, opzionale)* data prevista per la terminazione del progetto. **DataTerminazione** *(date, opzionale)* data in cui viene effettivamente terminato il progetto.
Dipendente	Descrittore di ogni dipendente aziendale.	**CF** *(char)* codice fiscale del dipendente. **Nome** *(varchar)* nome del dipendente. **Cognome** *(varchar)* cognome del dipendente. **Sesso** *(char)* sesso del dipendente. **DataNascita** *(date)* data di nascita del dipendente. **Indirizzo** *(varchar)* indirizzo di residenza del dipendente. **Email** *(varchar)* Email aziendale del dipendente. **TelefonoCasa** *(char, opzionale)* contatto telefonico di casa del dipendente. **Cellulare** *(char, opzionale)* contatto telefonico privato del dipendente. **Salario** *(float)* salario del dipendente. **Password** *(varchar)* password del dipendente per accedere al sistema.
Skill	Descrittore delle skill associate a un dipendente.	**IDSkill** *(int)* chiave identificativa delle skill. **NomeSkill** *(varchar)* nome della skill.
Meeting	Descrittore dei meeting tenuti in azienda o in videoconferenza per ciascun progetto.	**IDMeeting** *(int)* Chiave identificativa del meeting. **DataInizio** *(date)* Data in cui si terrà il meeting. **OrarioInizio** *(time)* Orario in cui comincia il meeting. **DataFine** *(date)* Data in cui è prevista la fine del meeting. **OrarioFine** *(time)* Orario in cui è prevista la fine del meeting. **Modalità** ***(modalità)*** modalità di incontro del meeting. **Piattaforma** ***(piattaforma**, opzionale)* eventuale piattaforma digitale dove è tenuta la videoconferenza.
SalaRiunione	Descrittore delle sale riunioni a disposizione in azienda.	**CodSala** *(char)* sigla identificativa della sala. **Capienza** *(int)* numero di posti disponibili in sala. **Indirizzo** *(varchar)* indirizzo dell'edificio in cui si trova la sala. **Piano** *(int)* piano in cui si trova la sala riunioni.
AmbitoProgetto	Descrittore degli ambiti di applicazione dei progetti aziendali.	**IDAmbito** *(int)* identificativo dell'ambito. **NomeAmbito** *(varchar)* nome dell'ambito.
LuogoNascita	Descrittore del luogo di nascita dei dipendenti.	**CodComune** *(char)* identificativo del comune. **NomeComune** *(varchar)* nome del comune.

		NomeProvincia *(varchar)* nome della provincia di appartenenza del comune.

2.5.2 Dizionario delle associazioni

Associazione	Descrizione	Classi coinvolte
Partecipazione	Descrive la partecipazione dei dipendenti ai progetti aziendali.	**Dipendente [1..*]** ruolo **comprende**: indica i dipendenti partecipanti ad un progetto. **Progetto [0..*]** ruolo **partecipa**: indica i progetti di uno specifico dipendente.
Abilità	Descrive le skill lavorative possedute da ciascun dipendente.	**Dipendente [1..*]** ruolo **posseduta**: indica i dipendenti che possiedono una specifica skill. **Skill [0..*]** ruolo **ha**: indica le skill lavorative che un dipendente possiede.
Discussione	Descrive i progetti discussi nei meeting aziendali.	**Progetto [1]** ruolo **relativo a**: indica il progetto discusso nel meeting. **Meeting [0..*]** ruolo **discusso**: indica i meeting il cui argomento è uno specifico progetto.
Riunione	Descrive le sale riunioni occupate per ogni meeting fisica.	**Meeting [0..*]** ruolo **ospita**: indica il meeting ospitato in una specifica sala riunioni. **SalaRiunione [0..1]** ruolo **in**: indica la sala riunioni in cui viene tenuto un meeting se questo non è in modalità telematica.
Presenza	Descrive la partecipazione dei dipendenti ai meeting organizzati.	**Dipendente [1..*]** ruolo **include**: indica i dipendenti presenti in un meeting. **Meeting [0..*]** ruolo **presente**: indica i meeting in cui è presente un dipendente.
Ambito	Descrive gli ambiti di applicazione dei vari progetti.	**Progetto [0..*]** ruolo **di**: indica i progetti che hanno uno specifico ambito d'applicazione. **AmbitoProgetto [1..*]** ruolo **ha**: indica gli ambiti di uno specifico progetto.
LuogoNascita	Descrive i luoghi di nascita dei dipendenti.	**LuogoNascita [1]** ruolo **nato a**: indica il comune e la provincia in cui è nato un dato dipendente. **Dipendente [0..*]** ruolo **del**: indica i dipendenti nati in uno specifico comune.

2.5.3 Dizionario dei vincoli

Nome vincolo	Descrizione
CFLegit **CfPartecipazione** **CfAbilità** **CfPresenza**	I codici fiscali di ciascun dipendente devono essere compatibili con i dati anagrafici inseriti. Oltre quindi alla richiesta che il codice fiscale sia una stringa esattamente di 16 caratteri alfanumerici, è richiesto che i campi relativi a Cognome, Nome, Data di nascita, Giorno di nascita, Comune di nascita e Carattere di Controllo del codice fiscale siano coerenti con i medesimi dati anagrafici del dipendente.
EmailLegit	L'email aziendale legata al dipendente deve essere formalmente corretta e quindi avere almeno un carattere prima del carattere "@", almeno un altro prima del carattere "." e almeno due caratteri dopo.
NomeLegit	Il nome non deve contenere numeri
CognomeLegit	Il cognome non deve contenere numeri
SalarioPositivo	Il salario di un dipendente deve necessariamente essere una quantità non negativa.

ValutazioneLimitata	Il valore della valutazione, basandosi su un sistema di punti decimale, deve sempre essere un valore appartenente all'intervallo [0,10].
DataCreazioneValida	La data di creazione di un progetto deve essere precedente o uguale alla data di scadenza e alla data di terminazione
DataTerminazioneValida	La data di terminazione di un progetto deve essere uguale o precedente alla data di scadenza.
Composizione Skill	Se una skill non è associata a nessun dipendente, non ha motivo di esistere nel database e va pertanto eliminata.
NoOnnipresenza	Un dipendente non deve essere in grado di partecipare a due o più meeting che avvengono alla stessa ora in luoghi diversi.
NoAccavallamenti	Due meeting fisici non possono avvenire nella medesima data e contemporaneamente nella stessa sala riunioni. Ciò non è richiesto per le videoconferenze.
DataValidaMeeting	Ogni meeting (fisico o telematico che sia) deve terminare in una data uguale o successiva a quella di inizio
OrarioValidoMeeting	Ogni meeting (fisico o telematico che sia) deve terminare in un orario che sia successivo a quello di inizio o deve terminare in un giorno successivo a quello di inizio.
PianoEsistente	Il piano in cui è situata una sala riunioni deve essere un numero non negativo.
ProjectManager	Ogni progetto può avere al più un dipendente nel ruolo di project manager.
CapienzaEsistente	La capienza di una sala riunione deve essere un numero non negativo, dato che non ha senso avere un numero negativo di posti a disposizione.
CapienzaRispettata	Ogni meeting fisico in una sala riunione deve rispettare la capienza della sala. In altri termini il numero di dipendenti presenti al meeting non possono superare la capienza della sala.
AmbitoLegit	I nomi degli ambiti progetto non possono contenere numeri.
LuogoNascita Esistente	Il luogo di nascita deve essere univocamente determinato dal codice del comune, nome del comune e nome della provincia di appartenenza.
TelefonoLegit **CellulareLegit**	I numeri di telefono e di cellulare del dipendente devono contenere solo numeri.
SessoLegit	Il sesso del dipendente può essere solo maschile(M) o femminile(F).
DataNascitaValida	La data di nascita del dipendente deve essere tale che quest'ultimo sia almeno maggiorenne.
ModalitàRiunione	La modalità del meeting può essere solo o telematica o fisica e se è telematica allora la sala riunione deve essere null e viceversa se è fisica allora la piattaforma deve essere null.
AmbitoProgettoEsistente	Un progetto non può avere associati più volte lo stesso ambito.
PartecipazioneEsistente	Un dipendente può partecipare al massimo una volta al progetto.
SkillEsistente	Una skill può essere associata allo stesso dipendente massimo una sola volta.
PresenzaEsistente	Un dipendente può presenziare a un meeting al massimo una volta.

3 Progettazione logica

In questa fase della progettazione stiliamo gli schemi relazionali a partire dal class diagram precedente, definendo anche le chiavi primarie, le chiavi esterne e gli eventuali ulteriori schemi relazionali necessari a tradurre le associazioni.

3.1 Schemi relazionali

Progetto (CodProgetto, NomeProgetto, TipoProgetto, DescrizioneProgetto, DataCreazione, Scadenza, DataTerminazione)

Partecipazione (CodProgetto, CF, RuoloDip)

CodProgetto → CodProgetto (Progetto), CF → CF (Dipendente)

Dipendente (CF, Nome, Cognome, Sesso, DataNascita, Indirizzo, Email, TelefonoCasa, Cellulare, Salario, Valutazione, Password, CodComune)

CodComune → CodComune (LuogoNascita)

LuogoNascita (CodComune, NomeComune, NomeProvincia)

Skill (IDSkill, NomeSkill)

Abilità (IDSkill, CF)

IDSkill → IDSkill (Skill), CF → CF (Dipendente)

AmbitoProgetto (IDAmbito, NomeAmbito)

AmbitoProgettoLink (IDAmbito, CodProgetto)

IDAmbito → IDAmbito (AmbitoProgetto), CodProgetto → CodProgetto (Progetto)

Meeting (IDMeeting, DataInizio, OrarioInizio, DataFine, OrarioFine, Modalità, Piattaforma, CodSala, CodProgetto)

CodSala → CodSala(SalaRiunione), CodProgetto → CodProgetto(Progetto)

Presenza (IDMeeting, CF, Presente, Organizzatore)

IDMeeting → IDMeeting (Meeting), CF → CF (Dipendente)

SalaRiunione (CodSala, Capienza, IndirizzoSede, Piano)

4 Descrizione Trigger e Funzioni

All'interno del database abbiamo definito una serie di trigger e funzioni per garantire la correttezza dei dati.

Si noti che in postgreSQL non è possibile definire direttamente funzioni all'interno del corpo di un trigger. Bisogna quindi definire prima la funzione per fare in modo che questa possa essere eseguita quando il trigger viene attivato.

Quando descriviamo il compito che svolge un trigger parliamo quindi sia della funzione ad esso associata che del trigger stesso.

Funzione Accavallamento: Questa funzione calcola un eventuale accavallamento di due eventi prendendo in input le loro date di inizio/fine ed i loro orari di inizio/fine.
Restituisce un boolean : true quando si verifica un accavallamento, false quando non si verifica.

Funzione ValutazioneMeeting: Questa funzione attribuisce una valutazione ad un dipendente in base ai meeting, compresa tra un valore di 0 e 10.
Per calcolare questo valore fa il rapporto tra i meeting in cui il dipendente si è effettivamente presentato e quelli in cui è stato invitato.
Prende in input il codice fiscale del dipendente e restituisce un float.

Funzione ValutazioneProgetti: Questa funzione attribuisce una valutazione ad un dipendente in base ai progetti, compresa tra un valore di 0 e 10.
Per calcolare questo valore fa il rapporto tra i progetti a cui il dipendente ha effettivamente collaborato e tutti i progetti aziendali.
Prende in input il codice fiscale del dipendente e restituisce un float.

Funzione Valutazione Totale: Questa funzione calcola la valutazione totale di un dipendente basandosi sulla sua partecipazione ai progetti ed al numero di presenze nei meeting, con un valore compreso tra 0 e 10.
Richiama a sua volta le funzioni ValutazioneMeeting e ValutazioneProgetto e successivamente fa la media tra i valori ritornati da queste due.
Prende in input il codice fiscale del dipendente e ritorna un float che sarà la valutazione.

Trigger AccavallamentoSale: Questo trigger garantisce che, prima di ogni insert o update di un meeting fisico, non ci siano meeting nella stessa sala che finiscano per accavallarsi con i tempi.
Nel caso in cui questo si verifichi lancia un eccezione e non autorizza l'operazione.

Trigger Capienza: Questo trigger controlla che , quando viene fatto un insert o un update nella tabella Presenza o quando avviene un insert o un update nella tabella Meeting, il numero di invitati nel meeting fisico sia minore o uguale alla capienza della sala in cui avviene. Nel caso in cui questo si verifichi invia un'eccezione che avvisa del superamento della capienza massima e consiglia di cambiare sala.

Trigger ComposizioneSkill: Questo trigger controlla che, quando avviene un delete nella tabella Abilità (quindi quando un dipendente perde una skill), ci siano altri dipendenti che abbiano come abilità quella determinata skill.
Se ciò non avviene vuol dire che nel database è presente una skill non posseduta da nessuno, questa quindi non ha più senso di esistere e viene eliminata.

Trigger Onnipresenza: Questo trigger garantisce che un dipendente non sia presente in più meeting contemporaneamente.
Quando viene fatto un insert o un update in Presenza o in Meeting controlla che i meeting di quel dipendente non si accavallino con il nuovo inserito.
Nel caso in cui questo si verifichi lancia un eccezione e non autorizza l'operazione.

Trigger UnicitàProjectManager: Questo trigger garantisce che per ogni progetto sia presente un unico project manager.
Prima di fare un insert o un update di un project manager in Partecipazione, controlla che non ci sia già un altro partecipante con lo stesso ruolo, in tal caso lancia un eccezione e non autorizza l'operazione.

Triggers Uppercase CodiceFiscale: Sono presenti una serie di Trigger che semplicemente convertono il codice fiscale del dipendente in lettere maiuscole, prima di fare un insert o un update.
Operano su tutte le tabelle in cui troviamo l'attributo codice fiscale, quindi in Abilità, Presenza, Partecipazione e Dipendente