Alessio Turetta, 2008069

Scarabottolo Samuel, 2012435

1 Abstract

KeplerSoftware è un'azienda italiana di fama internazionale che si occupa di sviluppare software aziendali e consulenze informatiche per clienti in tutto il mondo. Nel territorio tricolore vi sono diverse sedi, ove in ognuna delle quali vi lavorano dei dipendenti, ciascuno di essi con un ruolo specifico e fondamentale nell'organismo aziendale. Troviamo i direttori, elemento cruciale di ogni sede, i quali oltre a dirigere la sede di appartenenza, sono incaricati di redigere contratti il più proficui possibili sia per il cliente che per l'azienda stessa. Troviamo inoltre i Manager Generali che accettano o rifiutano proposte di progetti, Manager di Progetti che sono i veri e propri leader dei progetti che hanno passato positivamente la selezione, Sviluppatori che vengono assegnati ai moduli dei progetti assecondati dai Tester che man mano controllano se il codice sviluppato non genera errori. Infine ma non per importanza troviamo gli Addetti al Marketing che sono i dipendenti che principalmente si interfacciano al Cliente per trovare un accordo finanziario e successivamente firmare un contratto.

2 Analisi dei Requisiti

2.1 Descrizione testuale

Si vuole realizzare una base di dati per l'organizzazione di una Software House. Essa è composta da varie *sedi* sparse in tutto il mondo, identificate da un codice univoco e possedenti un indirizzo con: città, provincia, via, cap e numero civico.
In ogni sede ci lavorano dei *dipendenti*, ognuno di essi identificato da un codice univoco, per il riconoscimento del singolo, inoltre vengono salvati degli estremi di identificazione come nome, cognome e la data di nascita e quelli della sede per cui lavora.
Ognuno dei dipendenti è assegnato ad un *progetto*, quest'ultimo identificato da un codice univoco e possedenti un nome, descrizione e la data di inizio. Ogni progetto viene valutato dai *manager generali* e possono venire bocciati o approvati, inoltre ognuno di essi che ha passato la selezione ha un *leader* di riferimento (Manager) e diversi *sviluppatori* che ci lavorano.

A sua volta i progetti sono suddivisi in vari *moduli*, i quali sono identificati dal codice del progetto a cui fanno riferimento e del proprio codice di modulo. Ad ogni modulo poi vengono assegnati più Manager e sviluppatori.
L'insieme totale di dipendenti è formato da varie categorie:

- **Direttore:** il direttore è uno dei responsabili (assieme ad altri) della sede per cui lavora. Il suo impiego oltre a quello di dirigere appunto l'andamento della sede è anche quello di redigere contratti il più proficui per le due parti con il cliente.
- **Manager Generale:** è uno dei manager incaricati nel controllo dei progetti, decidono se è conveniente l'approvazione di esso e la successiva implementazione o meno
- **Manager Progetti:** i manager dei progetti, come dice il nome stesso, sono i leader incaricati nella direzione di essi. Oltretutto, essendo anche sviluppatori, oltre a partecipare attivamente ai progetti, partecipano nello sviluppo dei moduli stessi.
- **Sviluppatore:** è il dipendente base e chiave della software house, vengono assegnati ai moduli che verranno implementati da loro.
- **Tester:** sono i dipendenti incaricati nel testare se i programmi scritti dagli altri suoi colleghi svolgono le attività di cui ci si aspettava lo svolgimento
- **Addetto Marketing:** sono responsabili di tutti gli accordi con vari clienti.

Ogni *cliente* viene identificato dalla partita IVA e si interfaccia ad un *addetto marketing* per stringere un accordo. Successivamente viene firmato il *contratto* redatto dal *direttore*, ognuno di essi (contratti) viene identificato dalle parti firmanti, ovvero il cliente e il direttore, possiede una descrizione e la data della firma. Al cliente poi viene emessa una *fattura* come documento legale.
Nel database tutte le *entrate* e *uscite* vengono segnalate tramite delle *transazioni*, ognuna di esse identificate unicamente da un codice e dal codice della sede in cui è stata effettuata. Ognuna di esse è caratterizzata da un saldo (quantità di soldi impiegati nella transazione) e data della transazione.
L'insieme di tutte le transazioni inoltre si suddivide in:

- **Uscita:** vengono tenute conto di tutte le uscite per ogni sede
- **Entrata:** tutto ciò che ha a che fare con transazioni in entrata, come dei compensi o pagamenti di alcuni clienti
- **Fattura:** l'insieme di tutti i documenti legali che segnano un bene o servizio offerto al cliente richiedente
- **Stipendio:** compendio economico mensile dato al singolo lavoratore mensilmente

2.2 Glossario dei termini

Termine	Descrizione	Collegamenti
Sede	Luogo di lavoro dei dipendenti della Software House	Dipendente, Transazione
Dipendente	Lavoratore per la software house	Entità padre di Direttore, Manager Generale, Manager Progetto, Sviluppatore, Tester e Addetto Marketing. Collegato a Sede e Stipendio
Direttore	Parte dirigente della software house	Entità figlia di Dipendente ed è collegata a Contratto
Manager Generali	Dipendenti che sono effettivamente supervisori dell'andamento generale dell'azienda e giudici per la valutazione dei progetti	Entità figlia di Dipendente, collegata a Progetto

Manager Progetto	Amministrano i progetti software (e sono anche sviluppatori)	Entità figlia di Dipendente, collegata a Progetto e Modulo
Tester	Eseguono i test del software	Entità figlia di Dipendente, collegata a Test
Sviluppatore	Lavorano direttamente sui moduli del progetto	Entità figlia di Dipendente, collegata a Modulo
Addetto al Marketing	Dipendenti assegnati alla ricezione di possibili clienti	Entità figlia di Dipendente, collegata a Cliente
Progetto	Il software da sviluppare a cui partecipano diversi tipi di dipendenti all'interno della sede	Entità collegata a Modulo, Manager Progetti e Manager Generale
Modulo	Una sezione di software che andrà ad implementare il progetto	Entità collegata a Progetto, Manager Progetti, Sviluppatore e Test
Test	Un singolo test da eseguire su un modulo	Entità collegata a Tester e Modulo
Transazione	Tutte le azioni monetarie effettuate da una sede	Entità padre di Uscita, Entrata, Fattura, Stipendio, collegata a Sede.
Uscita	Tutte quelle transazioni in uscita diverse dagli stipendi dati ai dipendenti	Entità figlia di Transazione
Entrata	Entrata monetaria per una sede	Entità figlia di Transazione
Fattura	Documento legale emesso al cliente	Entità figlia di Transazione, collegata a Cliente
Stipendio	Compenso economico mensile dato al dipendenti	Entità figlia di Transazione, collegata a Dipendente
Contratto	Contratto redatto da un Direttore per un Cliente	Entità collegata a Direttore e Cliente

2.3 Operazioni tipiche

Le operazioni descritte di seguito sono rappresentate nell'arco di un mese di lavoro

Operazione	**Frequenza**	**Tipo**
Inserimento dei test svolti nel database	400 al giorno	Scrittura
Inserimento fatture emesse	300 al giorno	Scrittura
Calcolo del guadagno giornaliero	300 al giorno	Lettura
Retribuzione dei dipendenti	1000 volte al mese	Scrittura
Inserimento clienti	100 volte al giorno	Scrittura
Inserimento dei contratti firmati	150 volte al giorno	Scrittura

Inserimento transazioni in entrata	500 volte al giorno	Scrittura
Inserimento transazioni in uscita	300 volte al giorno	Scrittura
Stampare l'elenco di partecipanti per progetto	200 volte al giorno	Lettura
Controllo dei guadagni totali per tutte le sedi	600 volte al giorno	Lettura

3 Progettazione Concettuale

3.1 Lista delle entità

Il database è formato dalle entità descritte in seguito, i cui attributi sottolineati rappresentano le *chiavi primarie,* se non è specificato l'attributo può essere *NULL:*

SEDE			
id_sede	INT	NOT NULL	PK
n_civico	CHAR(5)	NOT NULL	
via	VARCHAR(20)	NOT NULL	
città	VARCHAR(20)	NOT NULL	
provincia	CHAR(2)	NOT NULL	
cap	CHAR(5)	NOT NULL	

DIPENDENTE			
id_dip	INT	NOT NULL	PK
nome	VARCHAR(20)	NOT NULL	
cognome	VARCHAR(20)	NOT NULL	
data_nascita	DATE	NOT NULL	
sede_via	VARCHAR(20)	NOT NULL	
sede_citta	VARCHAR(20)	NOT NULL	
sede_provincia	CHAR(2)	NOT NULL	
sede_cap	CHAR(5)	NOT NULL	

L'entità **Dipendente** a sua volta si può distinguere in:

- **Direttore:** è uno dei dipendenti più fondamentali della software house, il suo compito è quello di dirigere l'intera sede e soprattutto quella di interfacciarsi con il cliente e di redigere dei contratti.
- **Manager Generali:** sono incaricati nel revisionare i progetti proposti e di darne una valutazione del tipo "approvato" o "non approvato"
- **Manager Progetto:** il manager dei progetti sono appunto oltre ad essere partecipanti attivi come sviluppatori "speciali" sono anche dei Leader
- **Sviluppatore:** sono coloro che partecipano ai moduli dei progetti assegnati e sviluppano i codici implementativi
- **Tester:** svolgono il compito di visionare i codici implementati dagli sviluppatori e trovare vari bug
- **Addetto al Marketing:** si interfacciano con il cliente per stringere accordi di valore economico

PROGETTO			
id_prog	INT	NOT NULL	PK, entità figlia di Dipendente
nome_prog	VARCHAR(20)	NOT NULL	
descrizione	VARCHAR(100)	/	
data_inizio	DATE	/	

MODULO			
id_modulo	INT	NOT NULL	PK composta con Progetto

TEST			
id_test	INT	NOT NULL	PK composta con Modulo

CLIENTE			
p_iva	INT	NOT NULL	PK
nome_azienda	VARCHAR(50)	NOT NULL	

CONTRATTO			
id_contratto	INT	NOT NULL	PK composta con Direttore e Cliente

descrizione	VARCHAR(200)	NOT NULL
data_firma	DATE	NOT NULL

TRANSAZIONE			
id_trz	INT	NOT NULL	PK composta con Sede
descrizione	VARCHAR(300)	/	
data_trz	DATE	NOT NULL	
saldo	INT	NOT NULL	

L'entità **Transazione** a sua volta si può distinguere in:

- **Uscita:** tutte le transazioni in negativo per l'azienda che non siano stipendi
  - Per le transazioni in uscita il saldo deve essere obbligatoriamente <0
- **Entrata:** storico di tutte le entrate per una sede
  - Per le transazioni in entrata il saldo deve essere obbligatoriamente >0
- **Fattura:** documento legale rilasciato al cliente rappresentante il bene/servizio offerto con tutti gli identificativi del caso
  - Essendo appunto delle fatture in seguito ad una vendita ad un cliente il saldo deve essere per forza >0
- **Stipendio:** compenso economico dato al dipendente mensilmente
  - La retribuzione deve essere <0

3.2 Lista delle relazioni

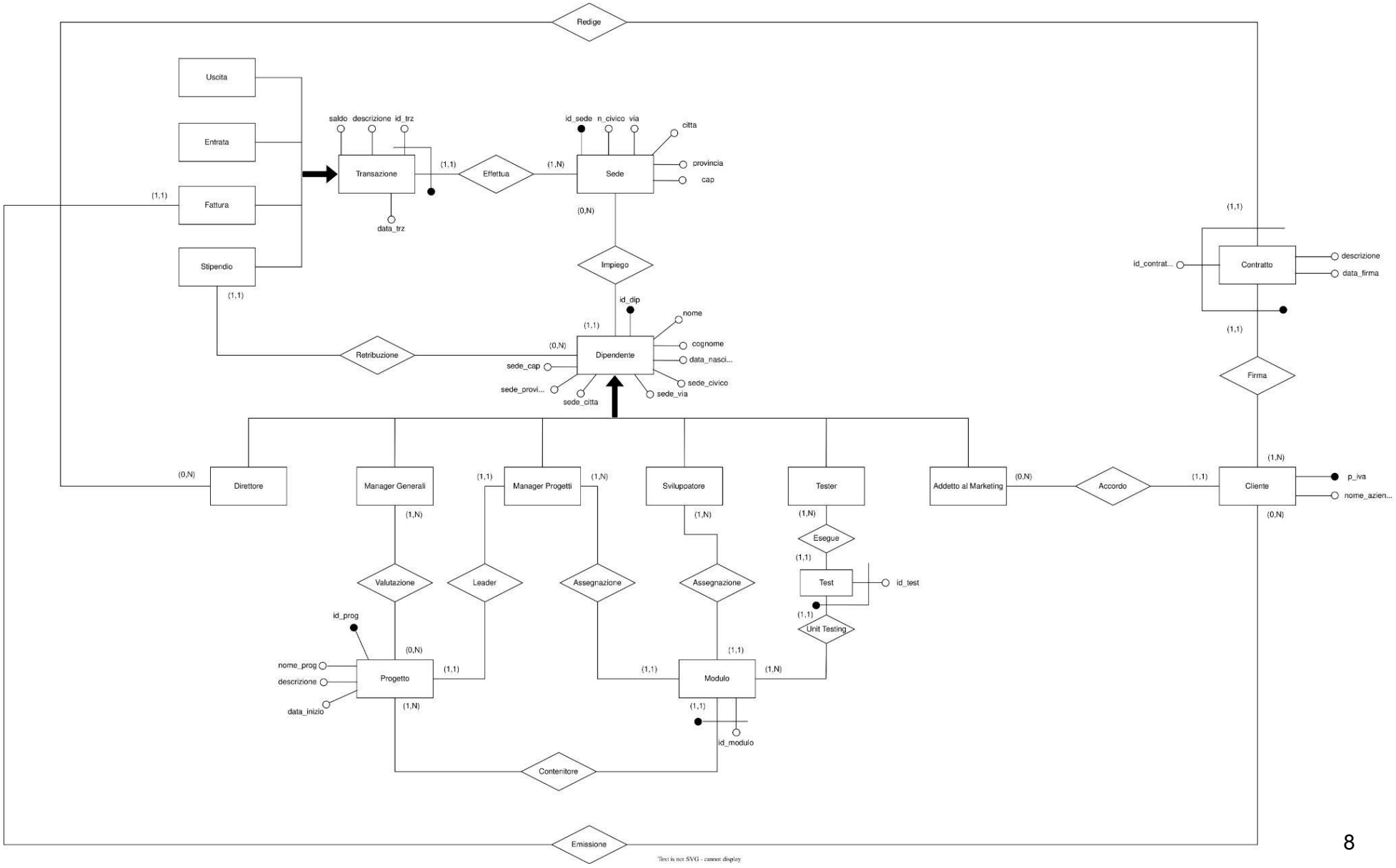
- Sede - Dipendente: **IMPIEGO**
  - Una sede se appena aperta non ha alcun dipendente che effettivamente ci lavori, altrimenti ne avrà molti (0,N)
  - Un dipendente lavora in un'unica sede (1,1)
- Direttore - Contratto: **REDIGE**
  - Un direttore può avere nessun contratto o molti da redigere (0,N)
  - Un contratto può essere redatto da unicamente da un solo direttore (1,1)
- Manager Generali - Progetto: **VALUTAZIONE**
  - L'amministratore analizzerà e approverà diversi progetti e ne boccerà altrettanti (1,N)
  - Diversi progetti verranno analizzati (0,N)
- Manager Progetti - Progetto: **LEADER**
  - Un manager è unicamente il leader di un progetto (1,1)
  - Allo stesso modo un progetto ha un solo leader (1,1)
- Manager Progetti - Modulo: **ASSEGNAZIONE**
  - Un manager di progetti può sviluppare uno o più moduli (1,N)
  - Un modulo può essere preso in carico da un solo manager (1,1)
- Sviluppatore - Modulo: **ASSEGNAZIONE**
  - Uno sviluppatore può incaricarsi di uno o più moduli (1,N)
  - Un modulo può essere preso in carico da un solo sviluppatore (1,1)
- Tester - Test: **ESEGUE**
  - Un tester può testare l'esecuzione di diversi moduli (1,N)
  - Un modulo è testato unicamente da un solo Tester (1,1)

- Test - Modulo: **UNIT TESTING**
    - Un test è eseguito su un solo modulo (1,1)
    - Un modulo contiene almeno un test (1,N)
- Addetto al Marketing - Cliente: **ACCORDO**
    - L'addetto al marketing può non avere clienti con cui accordarsi o al più stringe diversi accordi con diversi clienti (0,N)
    - Un clienti si accorda con un unico Addetto al marketing(1,1)
- Cliente - Contratto: **FIRMA**
    - Un cliente può firmare diversi contratti diversi (1,N)
    - Il contratto è unico e firmato da un solo cliente (1,1)
- Progetto - Modulo: **CONTENITORE**
    - Un progetto contiene uno o più moduli (1,N)
    - Un modulo appartiene ad un solo progetto (1,1)
- Transazione - Sede: **EFFETTUA**
    - La transazione è un'unica operazione (1,1)
    - Le sedi possono compiere diverse transazioni (1,N)
- Stipendio - Dipendente: **RETRIBUZIONE**
    - Lo stipendio viene erogato unicamente ad un unico dipendente (1,1)
    - I dipendenti possono ricevere più stipendi (0,N)
- Fattura - Cliente: **EMISSIONE**
    - La fattura è un documento emesso unicamente ad un solo cliente (1,1)
    - I clienti possono ricevere diverse fatture (0,N)

Vincoli non rappresentabili nel schema E-R:

- **VALUTAZIONE**:
    - Se un progetto è stato rifiutato, allora il progetto corrispondente alla tabella avrà come valore NULL per id_manager e data_inizio
- **CLIENTI:**
    - I clienti con cui si interfaccia la software house sono aziende aventi partita iva e dunque non privati

Schema concettuale



4 Progettazione Logica

4.1 Analisi delle ridondanze

Assegnazione

La relazione Assegnazione è ripetuta sia per Sviluppatori che per Manager di Progetti in quanto questi ultimi sono anch'essi degli Sviluppatori, seppur "speciali", in quanto appunto svolgono sia il compito di coders che di amministratori. Per ovviare a questa ridondanza viene fatta un'unica relazione "Assegnazione".

Dipendente

La tabella Dipendente presenta diverse ridondanze, in quanto sono presenti le colonne che rappresentano l'indirizzo della sede in cui il dipendente è situato. La risoluzione consiste nell'eliminare suddette colonne da Dipendente, mantenendo la relazione Impiego con chiave esterna su id_sede. Attuando questa soluzione si vanno a diminuire i tempi d'accesso per le varie query.

4.2 Eliminazione delle Generalizzazioni

Dipendente

La generalizzazione viene risolta accorpando tutte le entità figlie in quella padre e successivamente aggiungendo un attributo *"tipologia"* per identificare i diversi tipi di dipendenti. Per quanto riguarda le relazioni che prima erano collegate alle entità figlie ora sono collegate a quella padre e ognuna di esse, *"Valutazione", "Leader", "Assegnazione"* ed *"Esegue"* avranno come cardinalità minima 0. Si è scelta questa soluzione poiché gli accessi al padre sono contestuali.

Transazione

Questa generalizzazione è stata risolta incorporando le entità figlie in quella padre e aggiungendo un attributo *"tipo_trz"* per distinguere i vari tipi di transazioni. Le relazioni collegate a quelle figlie sono collegate ora a quella padre con cardinalità minima 0 in quanto gli accessi al padre sono anch'essi contestuali.

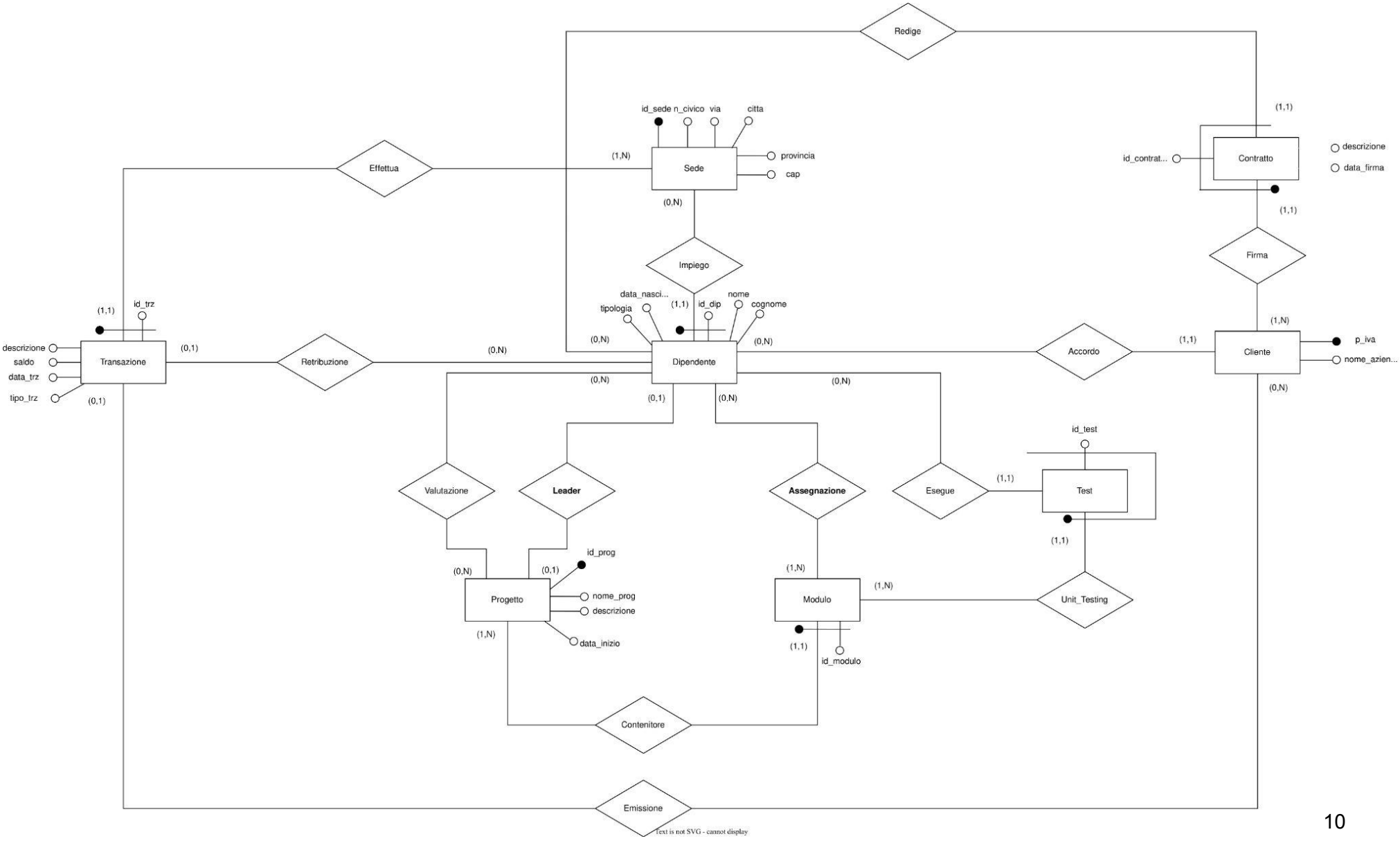
4.3 Scelta degli identificatori primari

La chiave dell'entità **Dipendente** è composta da un identificatore esterno verso l'entità "Sede". Facendo ciò però si va ad aumentare la complessità dello schema relazionale, per risolvere questa complicanza abbiamo affiancato questo identificatore esterno pure un attributo "id_dip" scelto come identificatore primario. Questo id è un intero di 3 cifre, esempio: 281, 699, 611…

Per quanto riguarda l'entità **Transazione**, il discorso è molto simile a quello precedente con "Dipendente", la chiave è composta da un identificatore esterno verso "Sede", affiancato pure da un attributo "id_trz" scelto come identificatore primario. Questo è un id composto da 3 cifre, esempio: 379, 384, 108…

Per quanto riguarda le entità rimanenti invece le chiavi sono rimaste invariate.

Schema concettuale risolto



4.3 Schema relazionale e vincoli di integrità relazionale

Sede(id_sede, n_civico, via, citta, provincia, cap)
Dipendente(id_dip, id_sede, nome, cognome, tipo_dip, data_nascita)
Dipendente.id_sede → Sede.id_sede
Progetto(id_prog, id_manager, nome_prog, descrizione, data_inizio)
Progetto.id_manager → Dipendente.id_dip
Valutazione(id_dip, id_prog, esito)
Valutazione.id_dip → Dipendente.id_dip
Valutazione.id_prog → Progetto.id_prog
Modulo(id_modulo, id_prog)
Modulo.id_prog → Progetto.id_prog
Assegnazione(id_dip, id_modulo)
Assegnazione.id_dip → Dipendente.id_dip
Assegnazione.id_modulo → Modulo.id_modulo
Test(id_test, id_dip, id_modulo)
Test.id_modulo → Modulo.id_modulo
Cliente(p_iva, nome_azienda)
Contratto(id_contratto, p_iva, id_dip, descrizione, data_firma)
Contratto.p_iva → Cliente.p_iva
Contratto.id_dip → Dipendente.id_dip
Accordo(id_dip, p_iva)
Accordo.id_dip → Dipendente.id_dip
Accordo.p_iva → Cliente.p_iva
Transazione(id_trz, id_sede, tipo_trz, saldo, descrizione, data_trz)
Transazione.id_sede → Sede.id_sede
Retribuzioni(id_trz, id_dip)
Retribuzioni.id_trz → Transazione.id_trz
Retribuzioni.id_dip → Dipendente.id_dip

5 Query e indici

5.1 Query

1.Conteggio dei moduli assegati a tutti i dipendenti di una provincia (nell'esempio la provincia è 'PO')

```
SELECT COUNT(*) as moduli, dipendente.id_dip, cognome, nome, citta, provincia
FROM assegnazione JOIN dipendente ON assegnazione.id_dip = dipendente.id_dip
JOIN sede ON dipendente.id_sede = sede.id_sede
WHERE provincia = 'PO'
GROUP BY dipendente.id_dip, cognome, nome, citta, provincia
ORDER BY moduli DESC;
```

	moduli bigint	id_dip integer	cognome character varying (20)	nome character varying (20)	citta character varying (20)	provincia character (2)
1	6	124	Navazzotti	Iole	Prato	PO
2	4	543	Formenti	Telemaco	Prato	PO
3	4	652	Hercolani	Saveria	Prato	PO
4	3	559	Ducci	Urania	Prato	PO
5	3	764	Semilla	Enrica	Prato	PO
6	2	158	Isaurini	Elena	Prato	PO
7	2	444	Camesaschi	Tizio	Prato	PO

2.Conteggio dei contratti stipulati, raggruppati per sede, prima di una certa data (nell'esempio abbiamo preso tutti i contratti prima del 2020)

```
SELECT COUNT (*) as Contratti, S.id_sede, C.data_firma
FROM Contratto as C JOIN Dipendente as D ON C.id_dip = D.id_dip
       JOIN Sede as S ON D.id_sede = S.id_sede
GROUP BY S.id_sede, C.data_firma
HAVING C.data_firma < '2020-1-1';
```

	contratti bigint	id_sede integer	data_firma date
1	1	594	2019-11-04
2	1	239	2019-06-17
3	1	239	2019-05-15
4	1	389	2009-04-12
5	1	239	2018-04-19

3.Calcolo della media degli stipendi suddivisi per ruolo

```
SELECT d.tipologia, -1* ROUND(AVG(t.saldo), 2) AS "Stipendio medio per ruolo"
FROM Dipendente as d JOIN Retribuzioni as r ON d.id_dip = r.id_dip
   JOIN Transazione as t ON t.id_trz = r.id_trz
GROUP BY d.tipologia;
```

	tipologia tipo_dip	Stipendio medio per ruolo numeric
1	Addetto Marketing	2125.00
2	Manager Generale	2266.67
3	Direttore	2950.00
4	Manager Progetto	1971.75
5	Sviluppatore	1513.90
6	Tester	1611.80

4.Selezione dei dipendenti che hanno valutato negativamente più di 3 progetti oppure positivamente più di uno

```
SELECT v.esito , d.id_dip, d.nome, d.cognome
FROM Dipendente as d JOIN Valutazione as v ON d.id_dip = v.id_dip
```

```
GROUP BY v.esito, d.id_dip, d.nome, d.cognome
HAVING v.esito = 'FALSE' AND COUNT(*) > 3
UNION
SELECT v.esito, d.id_dip, d.nome, d.cognome
FROM Dipendente as d JOIN Valutazione as v ON d.id_dip = v.id_dip
GROUP BY v.esito,d.id_dip, d.nome, d.cognome
HAVING v.esito = 'TRUE' AND COUNT(*) > 1;
```

	esito boolean	id_dip integer	nome character varying (20)	cognome character varying (20)
1	true	865	Eusebia	Sammaritano
2	true	677	Stella	Giangiuli
3	true	423	Leonardo	Gallenzi
4	true	989	Alessio	Rigli
5	false	423	Leonardo	Gallenzi
6	true	422	Rebecca	Mazzolai
7	true	234	Dionigi	Neffati
8	true	699	Melania	Pagliarello
9	true	955	Rocco	Modello

5.Classifica delle sedi identificate per id_sede, citta e via ordinate per il totale delle loro uscite. Inoltre per ognuna di esse si identifica lo stipendio più alto.

```
CREATE VIEW max_stipendio_per_sede as
   SELECT t.id_sede, MIN(t.saldo) as massimo_saldo
   FROM Transazione as t JOIN Retribuzioni as r ON t.id_trz = r.id_trz
   GROUP BY t.id_sede, t.tipo_trz
   HAVING t.tipo_trz = 'Stipendio';

CREATE VIEW stipendi_uscite as
   SELECT t.id_sede, s.via, s.citta, SUM(t.saldo) as Uscite_Totali,
          -1*ms.massimo_saldo as Stipendio_Massimo
   FROM max_stipendio_per_sede as ms JOIN Transazione as t
          ON ms.id_sede = t.id_sede
       JOIN Sede as s ON s.id_sede = t.id_sede
   GROUP BY t.id_sede, s.via, s.citta, t.tipo_trz, ms.massimo_saldo
   HAVING t.tipo_trz = 'Uscita'
   ORDER BY SUM(t.saldo);
```

	id_sede integer	via character varying (20)	citta character varying (20)	uscite_totali bigint	stipendio_massimo integer
1	239	Via Torelli	Prato	-1109727	2300
2	312	Via Delle Gagliarde	Arezzo	-1109538	3500
3	389	Via Ospedale Civile	Padova	-1061556	3500
4	594	Via Delle Pianazze	La Spezia	-1021784	3437
5	437	Via Tumedei Casalis	Carmagnola	-1014437	3500

6.Nome dei progetti in cui relativi moduli vi lavorano 3 o più dipendenti

```
SELECT p.nome_prog, a.id_modulo
FROM Progetto as p JOIN Modulo as m ON p.id_prog = m.id_prog
   JOIN Assegnazione as a ON a.id_modulo = m.id_modulo
GROUP BY p.nome_prog, a.id_modulo
HAVING COUNT(*) >= 3;
```

	nome_prog character varying (20)	id_modulo integer
1	ProjectSigma	65628
2	ProjectSigma	25401
3	MathSolver	34745
4	LWJGL	92610
5	ATM Intesa	30592
6	ProjectAlpha	50558
7	MathSolver	63351

5.2 Indice

Le transazioni delle varie sedi sono dati che vengono utilizzati molto spesso in lettura, basti pensare agli accessi giornalieri per controllare l'andamento dei profitti delle varie sedi. Quindi essendo che la software house lavora a livello globale, si ipotizza che la tabella delle transazioni si sviluppi su larga scala e dunque per ottimizzarne gli accessi si è deciso di aggiungere un indice "*indice_transazioni*" che indicizzi gli attributi Tipologia e Saldo.

```
CREATE INDEX indice_transazioni ON Transazione (tipologia, saldo);
```

6 Codice C++

6.1 Compilazione e Descrizione

Windows: Nella cartella del codice sorgente deve essere inclusa la cartella `dependencies` con all'interno altre due cartelle, `include` e `lib`. In `include` devono essere presenti `libpq-fe.h`, `pg_config_ext.h` e `postgres_ext.h`, mentre in lib devono esserci `libpq.dll` e `libpq.lib`. Per la compilazione e l'esecuzione è necessario eseguire i seguenti comandi:

```
g++ main.cpp -L dependencies\lib -lpq -o program
.\program [HOST] [USERNAME] [DBNAME] [PASSWORD] [PORT]
```

dove `[HOST]` è l'indirizzo dell'host, `[USERNAME]` è il nome utente, `[DBNAME]` è il nome del database, `[PASSWORD]` è la password del database, `[PORT]` è il numero di porta del server.

Su Linux non è necessaria alcuno particolare configurazione (eccetto l'installazione di PostgreSQL). Comandi per la compilazione ed esecuzione:

```
g++ main.cpp -o program -I /usr/include/postgresql -lpq
./program [HOST] [USERNAME] [DBNAME] [PASSWORD] [PORT]
```

In alcune distribuzioni non è necessaria la flag `-I [path]` essendo che riconoscono direttamente qualsiasi libreria installata in `/usr/`.

All'inizio dell'esecuzione il programma si assicura innanzitutto che vi siano i 6 argomenti necessari (in caso contrario previene la connessione al database). Successivamente tenta la connessione con il server database, e se non riesce a connettersi si rifiuta di eseguire ulteriormente. Dichiara un array di stringhe rappresentanti le query, e introduce un menù da cui selezionare una query; se la query è parametrica (query 1, 2 e 6), viene specificato il parametro da inserire; nello specifico:

- La query 1 richiede l'inserimento di una provincia (sigla a due caratteri) nel `WHERE`, ad esempio PO oppure PD.
- La query 2 richiede l'inserimento di una data nell'`HAVING`, con formato AAAA-MM-GG (Anno, Mese, Giorno)
- La query 6 richiede l'inserimento di un valore intero nell'`HAVING`, possibilmente ad una cifra

Dopo l'interrogazione del database, viene riproposto il menu, in cui l'ultima opzione permette di uscire dal programma (chiudendo la connessione).

6.2 Documentazione

```
PGconn* tryConn(const char* host, const char* user, const char* db, const char*
pass, const char* port)
```

Tenta la connessione con il server del database, prendendo come parametri le credenziali. Se la connessione viene eseguita con successo, ritorna la connessione stessa, altrimenti chiude il programma con un messaggio d'errore.

```
PGresult* paramExec(PGconn* conn, string queries[], int input)
```

Esegue una query parametrica, prendendo come parametri la connessione al database, un array di query e l'indice corrispondente alla query scelta nel menù. Ritorna il risultato della query.

```
void checkResults(PGresult* res, const PGconn* conn)
```

Controlla il risultato di una query, prendendo come parametri il risultato e la connessione. In caso di risultati inconsistenti, ripulisce il risultato e termina il programma, altrimenti finisce l'esecuzione della funzione.

```
void separateLines(int fields, int* maxLen)
```

Funzione per stampare separatori, prendendo come parametri il numero di colonne e un array di lunghezze massime per colonna. Da usare esclusivamente con la funzione `prettyPrint`.

```
void prettyPrint(PGresult* res)
```

Funzione per stampare l'intero risultato di una query con opportuna formattazione, prendendo come parametri il risultato di una query.