Corso Basi di Dati
Anno Accademico 2015/16

PROGETTAZIONE DI UNA BASE DI DATI PER LA GESTIONE DI UN'AZIENDA DI ASSISTENZA INFORMATICA

Jordan Gottardo - 1070703 - jgottard
Prete Giovanni - 1097588 - gprete

Database caricato su jgottard

Homepage interfaccia web:
http://basidati.studenti.math.unipd.it/~jgottard/

Dati di accesso:
Nome utente: username
Password: password

Indice

1. Abstract

Il progetto si pone lo scopo di modellare la realtà di un'azienda di assistenza informatica che si occupi di interventi di installazione e manutenzione presso sistemi esterni come datacenter o uffici. Nello specifico si vuole rappresentare la gestione delle lavorazioni eseguite all'interno di un dato incarico individuando il tipo di mansione svolto e il dipendente che ha eseguito il lavoro, si vuole inoltre tenere traccia dei costi relativi alle lavorazioni, delle fatture emesse per ogni incarico e dei dati relativi ai clienti che li richiedono.

2. Descrizione dei Requisiti

L'entità principale della base di dati è rappresentata dalla **lavorazione**, l'azienda utilizza questa classe per salvare le informazioni relative ai singoli lavori eseguiti che poi verrano utilizzate anche per la compilazione della **fattura**. Una lavorazione consiste in un'istanza di una specifica **mansione** che l'azienda mette a disposizione, i dati propri della lavorazione combinati con quelli generici della mansione permettono il calcolo del costo totale del lavoro.
Le mansioni disponbili possono essere divise in due categorie principali, ovvero mansioni **software** o **hardware**, entrambe presentano un costo orario ma le seconde hanno inoltre un prezzo aggiuntivo relativo al pezzo a cui fanno riferimento, all'interno della lavorazione sarà quindi necessario indicare il totale delle ore lavorate ed, eventualmente, la quantità di pezzi hardware usati. Gli altri dati salvati all'interno della lavorazione sono le date di inizio e fine della stessa, il **dipedente** a cui è stata affidata ed infine l'**incarico** di cui fa parte.
L'azienda mantiene una lista dei dipendenti correnti con i rispettivi nominativi e le date di assunzione, ogni dipendente inoltre può essere **abilitato** allo svolgimento di alcune mansioni a decorrere da una certa data ed è quindi necessario che ad ogni lavorazione venga assegnato un dipendente abilitato alla mansione relativa.
La classe incarico viene utilizzata dall'azienda per raccogliere più lavorazioni che sono state richieste durante lo svolgimento di una singola commissione richiesta da un dato **cliente**, all'interno dell'incarico vengono salvate le date di inizio e fine ed il cliente che lo ha richiesto. La data di fine incarico viene lasciata vuota fintanto che l'incarico è considerato aperto, una volta concluso la data di chiusura viene impostata ed è possibile cacolarne la fattura.
All'interno della fattura, oltre che la data di emissione, vengono registrati il totale imponibile, l'aliquota da applicare ad esso e l'incarico a cui fa riferimento.
Infine l'azienda mantiene una lista dei proprio clienti, questi sono divisi in due tipologie, **privati** e **imprese**, che differiscono per il codice identificativo associato, codice fiscale nel primo caso e partita iva per i secondi.

3. Progettazione Concettuale

3.1 Entità e Attributi

- Cliente: entità che identifica un cliente.
  - ID: INTEGER(11)
  - Nominativo: VARCHAR(50)
  - Indirizzo: VARCHAR(50)
  - Città: VARCHAR(30)
  - Telefono: VARCHAR(30)

    È presente la seguente generalizzazione totale:
    - Privato: entità che identifica un cliente privato.
      - CodiceFiscale: VARCHAR(16)
    - Impresa: entità che identifica un'impresa.
      - PartitaIVA(VARCHAR(11)

- Incarico: entità che identifica un incarico richiesto da parte di un cliente.
  - ID: INTEGER(11)
  - DataInizio: DATE
  - DataFine*: DATE

- Fattura: entità che identifica le fatture emesse.
  - ID: INTEGER(11)
  - DataEmissione: DATE
  - Imponibile: DECIMAL(12,2)
  - Aliquota: DECIMAL(4,2)

- Mansione: entità che identifica le mansioni che possono essere eseguite.
  - ID: INTEGER(11)
  - Descrizione: VARCHAR(50)
  - PrezzoOrario: DECIMAL(12,2)

    È quindi presente la seguente generalizzazione totale:
    - Software: entità che identifica le mansioni software
    - Hardware: entità che identifica le mansioni hardware
      - CostoHW: DECIMAL(12,2)

- Lavorazione: entità che identifica le mansioni realmente eseguite
  - ID: INTEGER(11)
  - DataInizio: DATE
  - DataFine: DATE
  - Ore: INTEGER(2)
  - Costo: DECIMAL(12,2)
  - QuantitàHW: INTEGER(3)

- Dipendente: entità che identifica i dipendenti attualmente assunti
    - ID: INTEGER(11)
    - Nome: VARCHAR(50)
    - Cognome: VARCHAR(50)
    - DataAssunzione: DATE

3.2 Relazioni

- Cliente-Acquisto: Ordinazione.
  Indica che il cliente ha ordinato un incarico.
    - Ogni cliente può ordinare da 1 a N incarichi. Un incarico è ordinato da un solo cliente.
    - Cardinalità **N:1**

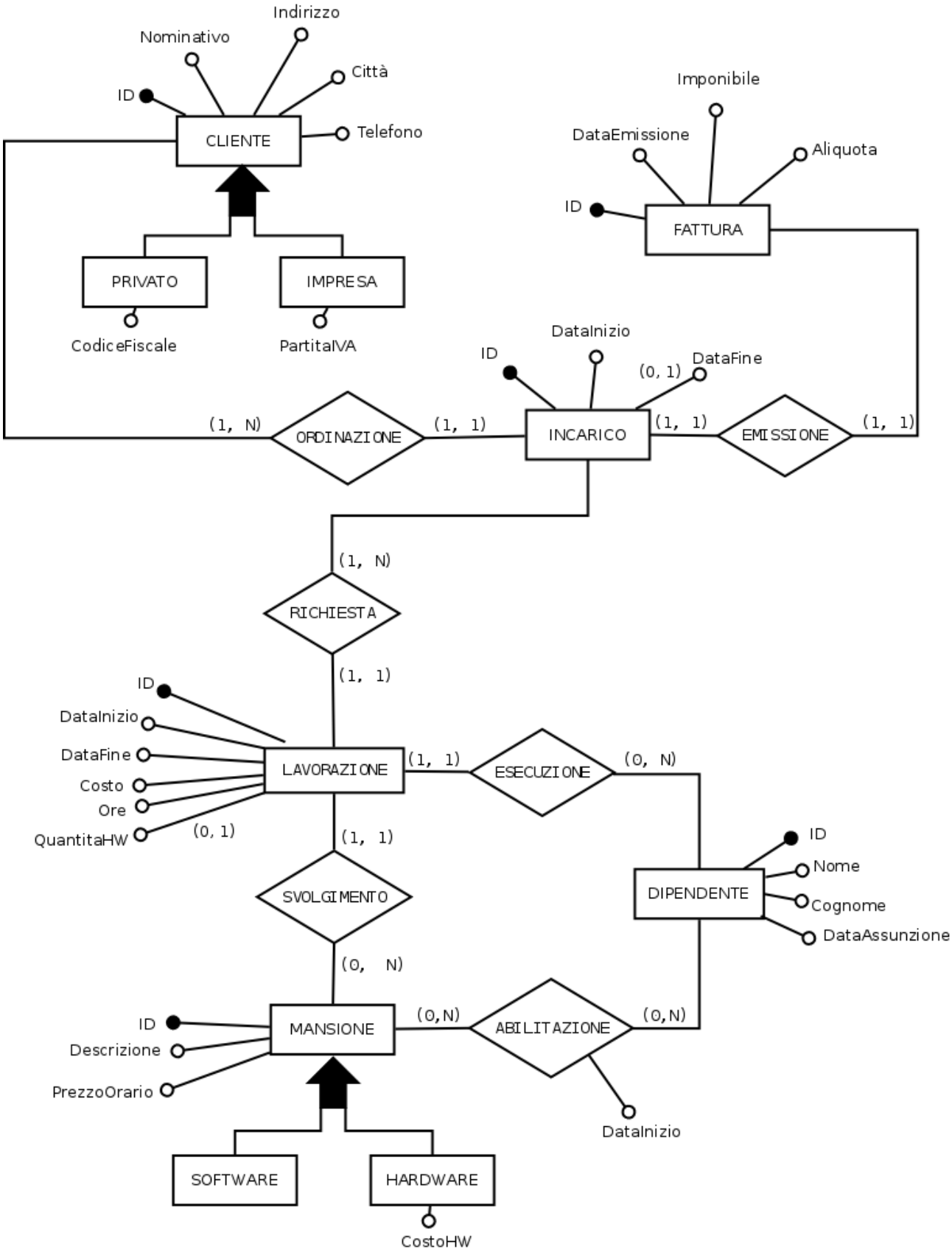
- Incarico-Fattura: Emissione.
  Indica che è stata emessa una fattura relativa ad un incarico.
    - Ogni fattura è emessa per un solo incarico. Un incarico può avere una sola fattura
    - Cardinalità **1:1**

- Incarico-Lavorazione: Richiesta.
  Indica che per un incarico viene richiesta una certa lavorazione.
    - Un incarico richiede da 1 a N lavorazioni. Una lavorazione è relativa ad un solo incarico.
    - Cardinalità: **N:1**
- Lavorazione-Mansione: Svolgimento.
  Indica che una mansione viene eseguita all'interno di una lavorazione.
    - Una mansione può essere svolta all'interno di 0 o N lavorazioni. In una lavorazione viene svolta una sola mansione.
    - Cardinalità: **1:N**

- Mansione-Dipendente: Abilitazione.
  Indica che un dipendente è abilitato a svolgere una data mansione.
    - Per una mansione possono essere abilitati 0 o N dipendenti. Un dipendente può essere abilitato a 0 o N dipendenti.
    - Cardinalità: **N:N**

- Lavorazione-Dipendente: Esecuzione.
  Indica che un dipendente ha eseguito una certa lavorazione
    - Una lavorazione viene eseguita da un solo dipendente. Un dipendente può svolgere da 0 a N lavorazioni.
    - Cardinalità: **1:N**

3.3 Schema E-R


Indirizzo
Nominativo
Città
ID
CLIENTE
Telefono
Imponibile
DataEmissione
Aliquota
ID
FATTURA
PRIVATO
IMPRESA
CodiceFiscale
PartitaIVA
DataInizio
ID
DataFine
(0, 1)
(1, N)
ORDINAZIONE
(1, 1)
INCARICO
(1, 1)
EMISSIONE
(1, 1)
(1, N)
RICHIESTA
(1, 1)
ID
DataInizio
DataFine
Costo
Ore
QuantitaHW
(0, 1)
LAVORAZIONE
(1, 1)
ESECUZIONE
(0, N)
(1, 1)
SVOLGIMENTO
ID
Nome
Cognome
DataAssunzione
DIPENDENTE
(0, N)
ID
Descrizione
PrezzoOrario
MANSIONE
(0,N)
ABILITAZIONE
(0,N)
DataInizio
SOFTWARE
HARDWARE
CostoHW

3.3.1 Vincoli non Esprimibili

Durante la progettazione sono stai indentificati i seguenti vincoli non esprimibili tramite schema ER

Incarico:

- dataFine deve essere maggiore di dataInizio
- dataFine deve essere maggiore o uguale del massimo tra lavorazione.dataFine

Fattura:

- l’imponibile deve essere la somma di lavorazione.costo dove lavorazione.incarico è uguale a incarico di fattura
- dataEmissione deve essere maggiore o uguale di incarico.dataFine

Lavorazione:

- dataInizio deve essere maggiore o uguale di incarico.dataInizio
- dataFine deve essere maggior o uguale di dataInizio
- quantitaHW deve essere diverso da NULL solo se la mansione svolta è hardware
- costo è dato da mansione.prezzoOrario moltiplicato per le ore; se la mansione è hardware va aggiunto anche mansione.costoHW per quantitaHW
- dipendente deve essere abilitato allo svolgimento della mansione

Abilitazione:

- dataInizio deve essere maggiore o uguale di dipendente.dataAssunzione

4. Progettazione Logica

4.1 Ristrutturazione

4.1.1 Analisi Ridondanze

Con i dati a disposizione è stata compilata la seguente tabella dei volumi di dati attesi per il database:

TABELLA	VOLUME	INCREMENTO MENSILE ATTESO
CLIENTE	100	+2
INCARICO	600	+10
FATTURA	600	+10
LAVORAZIONE	3000	+50
MANSIONE	25	+0
DIPENDENTE	10	+0
ABILITAZIONE	60	+2

In base a quest'analisi, ed in particolar modo all'osservazione che la tabella **fattura** è quella su cui si prevede il maggior volume di dati e il maggior incremento nel tempo, si è deciso di aggiungere, per motivi di efficienza nelle interrogazioni del database, due casi di ridondanza correlati tra loro; si tratta dell'attributo **costo** nell'entità **lavorazione** e dell'attributo **imponibile** nell'entità **fattura**.
Dato che verosimilmente questi due dati saranno utilizzati spesso è stato scelto di mantenerli ridondati per evitare che le molte operazioni che li coinvolgano vadano ad influire sulle prestazioni della base di dati.

- **Costo:** l'attributo è derivabile dalla relazione tra lavorazione e mansione ma il suo calcolo richiede molti accessi alle tabelle coinvolte. Inoltre la rindondanza potrebbe cessare di esistere nel caso in cui una mansione subisca modifiche o la cancellazione.

- **Imponibile:** in maniera simile l'attributo imponibile è derivabile dal costo delle lavorazioni coinvolte, il suo calcolo non risulta troppo oneroso in termini di accessi alle tabelle (dando per scontato che l'attributo costo sia presente) ma si è deciso comunque di ridondarlo data l'alta cardinalità prevista per la tabella a cui si dovrebbe accedere.

Di seguito una query esemplificativa per il calcolo dell'imponibile delle fatture nel caso in cui non ci fossero ridondanze:

```
SELECT f.*, SUM(
    (SELECT m1.prezzoOrario FROM mansione as m1 JOIN lavorazione AS l1
       ON m1.ID = l1.mansione JOIN fattura AS f1 ON l1.incarico = f1.incarico) *
    (SELECT l2.ore FROM lavorazione AS l2 JOIN fattura AS f2
       ON l2.incarico = f2.incarico) +
    IFNULL((SELECT m3.costoHW FROM mansione as m3 JOIN lavorazione AS l3
       ON m3.ID = l3.mansione JOIN fattura AS f3 ON l3.incarico = f3.incarico), 0) *
    IFNULL((SELECT l4.quantitaHW FROM lavorazione AS l4 JOIN fattura AS f4
       ON l4.incarico = f4.incarico), 0)
) AS imponibileCalc
FROM fattura AS f
```

Come si può notare all'interno del SELECT vengono eseguiti i seguenti accessi:
2x mansione
4x lavorazione
4x fattura
ognuno dei quali moltiplicato per la cardinalità della tabella su cui avviene.

Con l'aggiunta della ridondanze questi accessi vengono eseguiti unicamente durante gli inserimenti e gli aggiornamenti di una tupla ma non durante le interrogazioni.

4.1.2 Eliminazione Generalizzazioni

Le generalizzazioni delle entità Cliente e Mansione sono state eliminate portando gli attributi delle entità figlie nei rispettivi padri. È stato deciso di aggiungere un attributo di “tipo” alle entità per avere un discriminante preciso e non annullabile (a differenza di partita IVA e codice fiscale) sulla tipologia di cliente e di mansione a cui si vuole fare riferimento

4.1.3 Scelta Identificatori Principali

Data la natura dei dati stessi e l'intrinseca inaffidabilità nell'inserimento (fattore umano) si è optato per un sistema di chiavi primarie basato su ID univoci ed assegnati in maniera automatica dal DBMS per quasi tutte le entità presenti, fa eccezione l'entità abiltazione

Nello specifico:

- Per le entità **fattura** e **mansione** non sono stati forniti altri attributi che avrebbero potuto svolgere la funzione di identificatori.
- Per **dipendente** si sarebbe potuta scegliere la coppia (Nome, Cognome) ma questo avrebbe dato problemi per casi di omonimia.
- Nel caso di **cliente** si sarebbe potuto utilizzare o la partita IVA o il codice fiscale, possibilmente accopiati al nominativo, ma dato che MySQL non permette di creare un vincolo per cui due attributi che possano essere nulli abbiano, presi assieme come attributo unico, una valore non nullo si è optato anche in questo caso per un ID. Inoltre in rari casi può accadere che il codice fiscale non sia univoco.

- Per quanto riguarda **incarico** e **lavorazione** si sarebbero potute scegliere alcune combinazioni di attributi già presenti, per esempio (cliente, dataInizio) in incarico e (incarico, mansione, dipendente, dataInizio) in lavorazione. Questi insiemi di attributi però avrebbero lasciato spazio a possibili (per quanto rari) casi di due o più tuple con il medesimo identificatore primario, in particolare per il fatto che entrambi hanno una data al loro interno che in questo caso non è sembrato un attributo particolarmente affidabile da usare come chiave.
- Per l'entità **abilitazione** invece è stato possibile scegliere come identificatore primario la coppia (dipendente, mansione), dato che un dipendente può essere abilitato ad eseguire una mansione al più una sola volta questa coppia di valori sarà sempre univoca.

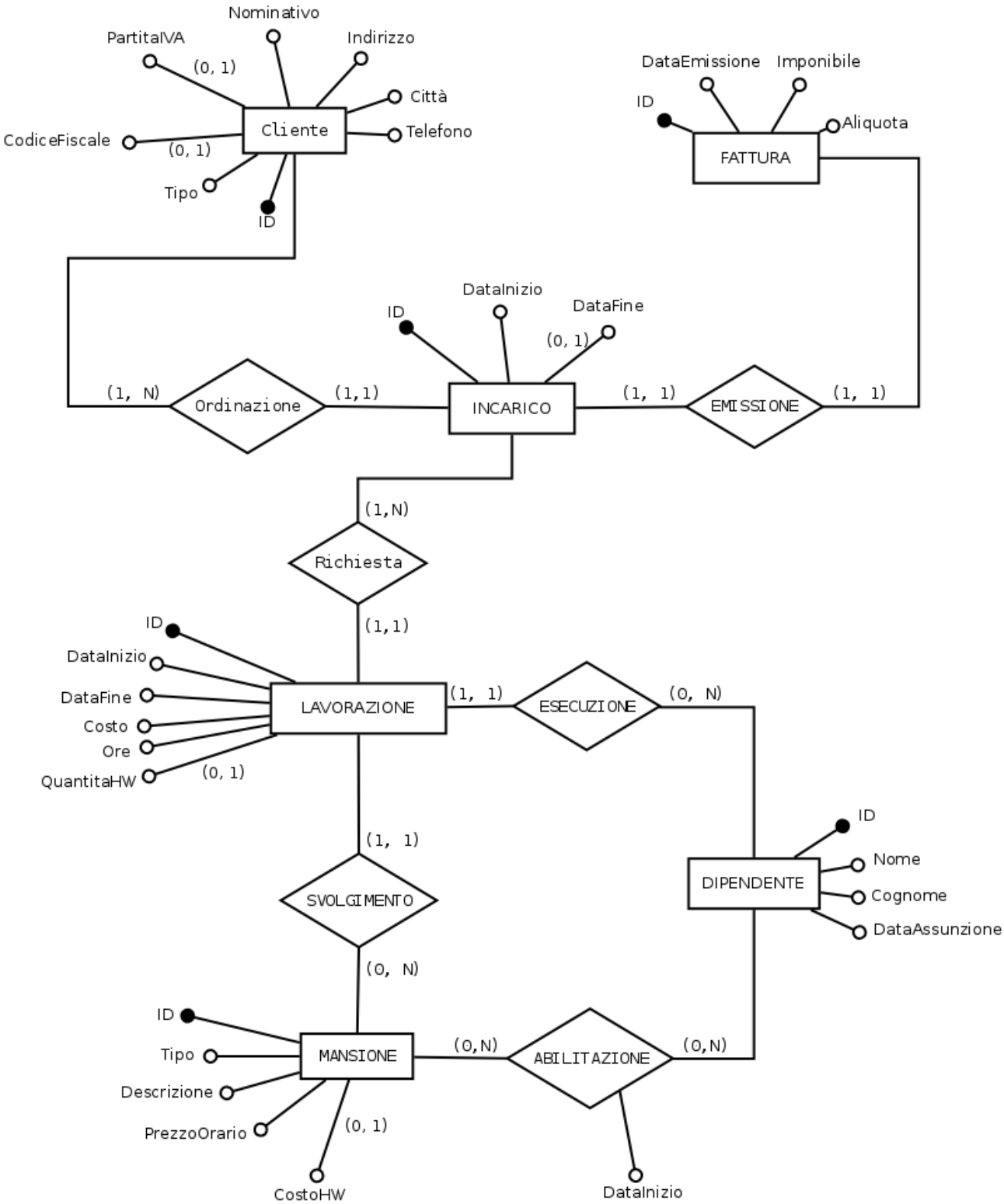
4.2 Modello Relazionale

La relazione 1:1 è stata eliminata ed è stato deciso di inserire una chiave esterna relativa all'incarico in fattura.
Le relazioni 1:N sono state eliminate e sono state inserite chiavi esterne nell'entità lato "1" della relazione.
La relazione N:N (Abilitazione) è stata resa entità e sono state inserite due chiavi esterne, che servono anche come chiave primaria.

NOTA: **PK**, *FK,* NULLABLE*

- **Cliente**
  (**ID**, Tipo, Nominativo, CodiceFiscale*, PartitaIva*, Indirizzo, Citta, Telefono)
- **Incarico**
  (**ID**, DataInizio, DataFine*, *Cliente*)
- **Fattura**
  (**ID**, DataEmissione, Imponibile, Aliquota, *Incarico*)
- **Mansione**
  (**ID**, Tipo, Descrizione, PrezzoOrario, CostoHW*)
- **Lavorazione**
  (**ID**, DataInizio, DataFine, Ore, Costo, QuantitaHW*, *Incarico*, *Mansione*, *Dipendente*)
- **Abilitazione**
  (***Dipendente***, ***Mansione***, DataInizio)
- **Dipendente**
  (**ID**, Nome, Cognome, DataAssunzione)

4.3 Schema E-R Ristrutturato



4.3.1 Vincoli non Esprimibili

In seguito alla ristrutturazione dello schema sono stati identificati ulteriori vincoli non esprimibili.

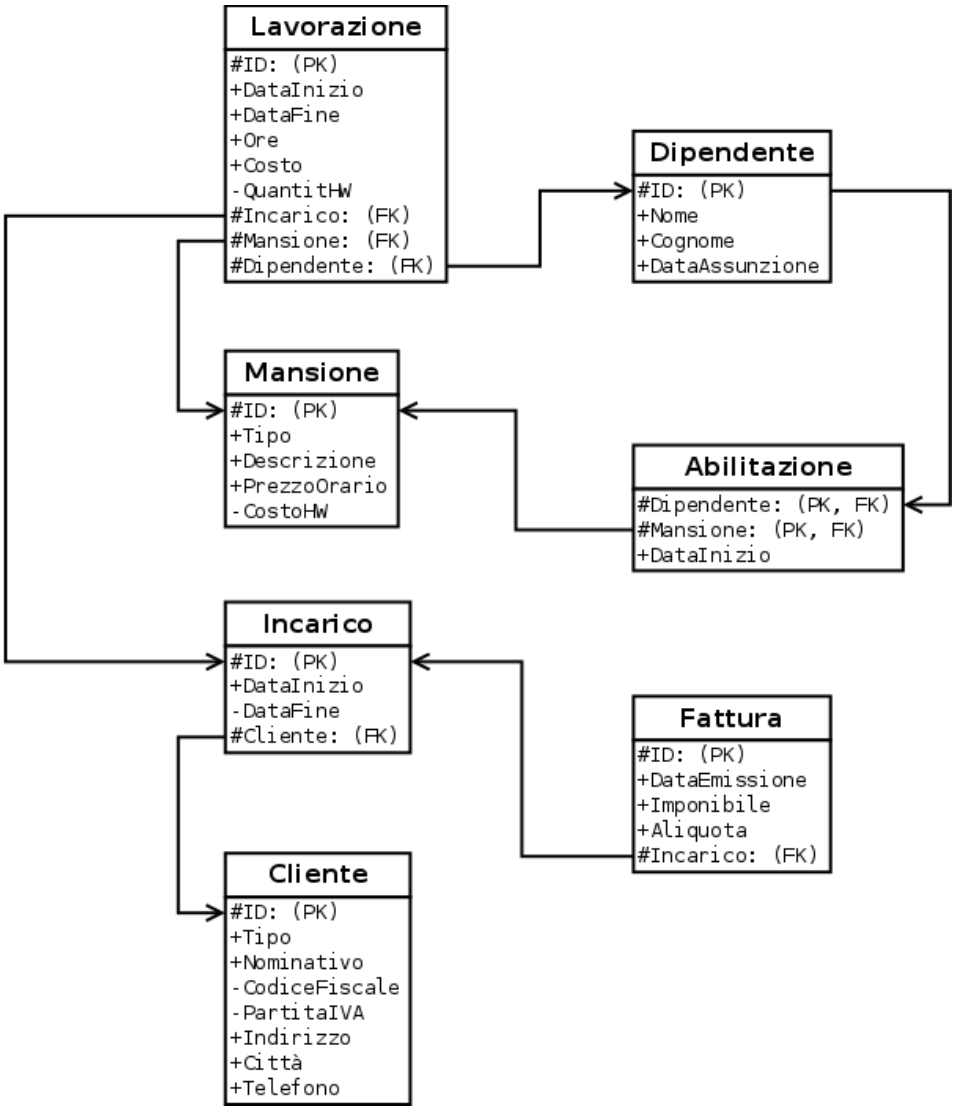
Cliente:

- se tipo è uguale a P(rivato) codiceFiscale deve essere diverso da NULL e partitaIVA deve essere NULL
- se tipo è uguale a I(mpresa) partitaIVA deve essere diverso da NULL e codiceFiscale deve essere NULL

Mansione:

- se tipo è H(ardware) costoHW deve essere diverso da NULL
- se tipo è S(oftware) costHW deve essere NULL

4.4 Schema UML



5. Implementazione Schema Logico

5.1 Lista delle Tabelle

- **Cliente**
  - **ID** *int(11)* <<PK>>
  - tipo *enum('P', 'I')*
  - nominativo *varchar(50)*
  - codiceFiscale *varchar(16)*
  - partitaIVA *varchar(11)*
  - indirizzo *varchar(30)*
  - citta *varchar(30)*
  - telefono *varchar(11)*

- **Incarico**
  - **ID** *int(11)* <<PK>>
  - dataInizio *date*
  - dataFine *date*
  - cliente *int(11)* <<FK (Cliente)>>

- **Fattura**
  - **ID** *int(11)* <<PK>>
  - dataEmissione *date*
  - imponibile *decimal(12,2)*
  - aliquota *decimal(4,2)*
  - incarico *int(11)* <<FK (Incarico)>>

- **Dipendente**
  - **ID** *int(11)* <<PK>>
  - nome *varchar(50)*
  - cognome *varchar(50)*
  - dataAssunzione *date*

- **Mansione**
  - **ID** *int(11)* <<PK>>
  - tipo *enum('S', 'H')*
  - descrizione *varchar(50)*
  - costoOrario *decimal(12,2)*
  - costoHW *decimal(12,2)*

- **Lavorazione**
  - **ID** *int(11)* <<PK>>
  - dataInizio *date*
  - dataFine *date*
  - ore *int(2)*
  - costo *decimal(12,2)*
  - quantitaHW *int(3)*
  - incarico *int(11)* <<FK (Incarico)>>
  - mansione *int(11)* <<FK (Mansione)>>
  - dipendente *int(11)* <<FK (Dipendente)>>

- **Abilitazione**
  - **dipendente** *int(11)* <<PK, FK (Dipendente)>>
  - **mansione** *int(11)* <<PK, FK (Mansione)>>
  - dataInizio *date*

6. Implementazione della Base di Dati

6.1 Creazione Tabelle

```
CREATE TABLE Cliente (
  ID int(11) NOT NULL AUTO_INCREMENT,
  tipo ENUM('P', 'I') DEFAULT 'P',
  nominativo varchar(50) NOT NULL,
  codiceFiscale varchar(16) UNIQUE DEFAULT NULL,
  partitaIVA varchar(11) UNIQUE DEFAULT NULL,
  indirizzo varchar(50) NOT NULL,
  citta varchar(30) NOT NULL,
  telefono varchar(11) NOT NULL,
  PRIMARY KEY (ID)
) ENGINE=InnoDB;

CREATE TABLE Incarico (
  ID int(11) NOT NULL AUTO_INCREMENT,
  dataInizio date NOT NULL,
  dataFine date DEFAULT NULL,
  cliente int(11) NOT NULL,
  PRIMARY KEY (ID),
  FOREIGN KEY (cliente) REFERENCES Cliente(ID)
) ENGINE=InnoDB;

CREATE TABLE Fattura (
  ID int(11) NOT NULL AUTO_INCREMENT,
  dataEmissione date NOT NULL,
  imponibile decimal(12,2) NOT NULL DEFAULT '0',
  aliquota decimal(4,2) NOT NULL DEFAULT '22',
  incarico int(11) NOT NULL,
  PRIMARY KEY (ID),
  FOREIGN KEY (incarico) REFERENCES Incarico(ID)
) ENGINE=InnoDB;
```

```sql
CREATE TABLE Dipendente (
  ID int(11) NOT NULL AUTO_INCREMENT,
  nome varchar(50) NOT NULL,
  cognome varchar(50) NOT NULL,
  dataAssunzione date NOT NULL,
  PRIMARY KEY (ID)
) ENGINE=InnoDB;

CREATE TABLE Mansione (
  ID int(11) NOT NULL AUTO_INCREMENT,
  tipo ENUM('S', 'H') NOT NULL DEFAULT 'S',
  descrizione varchar(50) NOT NULL,
  prezzoOrario decimal(12,2) NOT NULL DEFAULT '0',
  costoHW decimal(12,2) DEFAULT NULL,
  PRIMARY KEY (ID)
) ENGINE=InnoDB;

CREATE TABLE Lavorazione (
  ID int(11) NOT NULL AUTO_INCREMENT,
  dataInizio date NOT NULL,
  dataFine date NOT NULL,
  ore int(2) NOT NULL DEFAULT '0',
  costo decimal(12,2) NOT NULL DEFAULT '0',
  quantitaHW int(3) DEFAULT NULL,
  incarico int(11) NOT NULL,
  mansione int(11),
  dipendente int(11),
  PRIMARY KEY (ID),
  FOREIGN KEY (incarico) REFERENCES Incarico(ID),
  FOREIGN KEY (mansione) REFERENCES Mansione(ID) ON DELETE SET NULL,
  FOREIGN KEY (dipendente) REFERENCES Dipendente(ID) ON DELETE SET NULL
) ENGINE=InnoDB;


CREATE TABLE Abilitazione (
  dipendente int(11) NOT NULL,
  mansione int(11) NOT NULL,
  dataInizio date NOT NULL,
  PRIMARY KEY (dipendente, mansione),
  FOREIGN KEY (dipendente) REFERENCES Dipendente(ID) ON DELETE CASCADE,
  FOREIGN KEY (mansione) REFERENCES Mansione(ID) ON DELETE CASCADE
) ENGINE=InnoDB;
```

6.2 Creazione delle Viste

```sql
CREATE VIEW IncarichiNonTerminati AS
SELECT ID, DataInizio, DataFine, Cliente
FROM Incarico
WHERE DataFine IS NULL;

CREATE VIEW IncarichiSenzaFattura AS
SELECT I.ID, I.DataInizio, I.DataFine, I.Cliente
FROM Incarico AS I
WHERE I.ID NOT IN (SELECT I2.ID FROM Incarico AS I2 join Fattura AS F1 ON
I2.ID=F1.Incarico)
  AND I.ID NOT IN (SELECT ID FROM IncarichiNonTerminati);
```

6.3 Query

1. Lista dei clienti con incarichi in corso.

```
SELECT C.Id AS IdCliente, C.Nominativo, I.Id AS IdIncarico, I.DataInizio
FROM Cliente AS C JOIN IncarichiNonTerminati AS I ON C.Id=I.Cliente;
```

IdCliente	Nominativo	IdIncarico	DataInizio
5	Ludovica Sal	1	2016-04-20
1	Mario Bianchi	2	2016-05-10
1	Mario Bianchi	3	2016-05-15
9	Lazzarel SNC	8	2016-03-11
2	Matteo Gallo	9	2016-01-24
7	WeBreakIt SRL	10	2016-04-25
5	Ludovica Sal	14	2016-04-15
3	Tommaso Sagese	15	2016-05-26
9	Lazzarel SNC	16	2016-06-01
10	Caramel SRL	17	2016-06-05
8	Pneus SPA	18	2016-05-26
3	Tommaso Sagese	19	2016-06-02
7	WeBreakIt SRL	20	2016-06-06

2. Lista di tutte le mansioni eseguite con i rispettivi incarichi e le relative fatture.

```
SELECT DISTINCT M.ID AS IdMansione, M.Descrizione, F.ID AS IdFattura
FROM Mansione AS M JOIN Lavorazione AS L ON M.ID=L.Mansione JOIN Incarico AS I
ON L.Incarico=I.ID JOIN Fattura AS F ON I.ID=F.Incarico;
```

IdMansione	Descrizione	IdFattura
6	Installazione ventole	7
4	Aggiornamento OS	7
5	Installazione CPU	7
1	Installazione antivirus	7
7	Sostituzione pasta termica	7
2	Deframmentazione hard disk	6
5	Installazione CPU	6
1	Installazione antivirus	6
8	Pulizia PC	6
7	Sostituzione pasta termica	6
3	Formattazione disco	6
1	Installazione antivirus	5
7	Sostituzione pasta termica	5

4	Aggiornamento OS	5
8	Pulizia PC	4
6	Installazione ventole	4
5	Installazione CPU	4
3	Formattazione disco	4
2	Deframmentazione hard disk	4
7	Sostituzione pasta termica	4
1	Installazione antivirus	4
4	Aggiornamento OS	4
1	Installazione antivirus	3
3	Formattazione disco	3
6	Installazione ventole	3
4	Aggiornamento OS	3
5	Installazione CPU	3
2	Deframmentazione hard disk	3
7	Sostituzione pasta termica	3
8	Pulizia PC	3
6	Installazione ventole	2
2	Deframmentazione hard disk	2
5	Installazione CPU	2
3	Formattazione disco	2
1	Installazione antivirus	2
2	Deframmentazione hard disk	1
3	Formattazione disco	1
7	Sostituzione pasta termica	1

3. Clienti che hanno richiesto almeno 1 incarico nel mese corrente.

```
SELECT C.ID AS IdCliente, count(*) AS NumeroIncarichi
FROM Cliente AS C JOIN Incarico AS I  ON C.ID=I.Cliente
WHERE YEAR(I.DataInizio)=YEAR(NOW()) AND MONTH(I.DataInizio)=MONTH(NOW())
GROUP BY IdCliente
HAVING NumeroIncarichi>=1;
```

IdCliente	NumeroIncarichi
3	1
7	1
9	1
10	1

4. Totale e media del fatturato annuo.

```
SELECT YEAR(DataEmissione) AS Anno, SUM(Imponibile) AS Totale, AVG(Imponibile)
AS MediaFatturato
FROM Fattura
GROUP BY Anno;
```

Anno	Totale	MediaFatturato
2015	45312.00	6473.142857

5. ID e data d'inizio degli incarichi nei quali sono state eseguite almeno una lavorazione software e due lavorazioni hardware per le quali, in tutto, siano stati utilizzati più di 50 pezzi.

```
SELECT INC.ID AS IdIncarico, INC.DataInizio, SUM(LAV.quantitaHW) AS TotalePezzi
FROM Incarico AS INC JOIN Lavorazione AS LAV ON INC.ID=LAV.Incarico
WHERE INC.ID IN
  (
  SELECT I.ID
  FROM Incarico AS I JOIN Lavorazione as L ON I.ID=L.Incarico JOIN Mansione AS M
ON M.ID=L.Mansione WHERE M.Tipo='S'
  ) AND INC.ID IN
    (
    SELECT I.ID FROM Incarico as I JOIN Lavorazione as L ON I.ID=L.Incarico
    JOIN Mansione as M on M.ID=L.Mansione  WHERE M.Tipo='H'
    GROUP BY I.ID
    HAVING count(*)>=2 AND SUM(quantitaHW)>=50
    )
GROUP BY IdIncarico, INC.DataInizio;
```

IdIncarico	DataInizio	TotalePezzi
3	2016-05-15	77
4	2015-10-20	50
8	2016-03-11	86
11	2015-09-19	59
12	2015-05-27	65
15	2016-05-26	62
18	2016-05-26	76
19	2016-06-02	60
20	2016-06-06	65

6. ID e nominativo dei clienti privati per i quali sono state emesse più di 2 fatture che abbiano un imponibile medio maggiore di 500 euro ma che non contengano mansioni che siano state eseguite all'iterno di un incarico richiesto da un'impresa e non ancora terminato.

```
SELECT Cli.ID AS IdCliente, Cli.nominativo, AVG(Imponibile) AS MediaImponibile
  FROM Cliente AS Cli JOIN Incarico AS Inc ON Cli.ID=Inc.Cliente JOIN Fattura AS
Fat ON Fat.Incarico=Inc.ID
WHERE NOT EXISTS
  (
  SELECT Lav.Mansione FROM Lavorazione AS Lav WHERE Lav.Incarico=Inc.ID AND
EXISTS
    (
    SELECT L.Mansione FROM Lavorazione AS L JOIN IncarichiNonTerminati AS I on
I.ID=L.Incarico JOIN Cliente AS C ON I.Cliente=C.ID
    )
    AND Inc.ID NOT IN (SELECT ID FROM IncarichiNonTerminati)
    GROUP BY Cli.ID, Cli.Nominativo
    HAVING count(*)>=2 AND MediaImponibile>'500';
```

IdCliente	Nominativo	MediaFatture
1	Mario Bianchi	750.000000

7. Dati degli incarichi che hanno richiesto ogni mansione esattamente una sola volta.

```
SELECT * FROM Incarico AS i1 WHERE
  (
  SELECT COUNT(*) FROM Lavorazione AS l3 WHERE l3.incarico = i1.ID
  AND l3.mansione NOT IN
    (
    SELECT l4.mansione FROM Lavorazione AS l4 WHERE l4.incarico = i1.ID AND
l4.ID <> l3.ID
    )
  ) = (SELECT COUNT(*) FROM Mansione);
```

IdIncarico	DataInizio	DataFine	Cliente
7	2015-02-20	2015-03-22	6
11	2015-09-19	2015-10-23	5
18	2016-05-26	---	8

8. Totale delle fatture emesse per incarichi che comprendono lavorazioni che sono state terminate oltre una settimana dopo la data di inizio incarico.

```
SELECT SUM(f1.imponibile) FROM Fattura AS f1 JOIN Incarico AS i1 ON
i1.ID=f1.incarico JOIN Lavorazione AS l1 ON l1.incarico=i1.ID
WHERE (i1.dataInizio + INTERVAL 7 DAY) <= (l1.dataFine);
```

Totale
308530.00

9. ID degli incarichi nei quali uno o più dipendenti hanno partecipato lavorando in totale più di 100 ore ed in cui le lavorazioni prese in considerazione siano iniziate non oltre un anno fa a partire dalla data corrente, per ogni ID vengono visualizzati nome e cognome dei dipendenti che soddisfano il vincolo.

```
SELECT DISTINCT i1.ID AS idIncarico, d1.Nome, d1.Cognome FROM Incarico AS i1
JOIN Lavorazione AS l1 ON l1.incarico=i1.ID
JOIN Dipendente AS d1 ON l1.dipendente=d1.ID
WHERE (l1.dataInizio > (CURDATE() - INTERVAL 365 DAY))
AND EXISTS
  (
  SELECT * FROM Lavorazione AS l2 JOIN Abilitazione AS a2 ON l2.mansione =
a2.mansione
  WHERE l2.incarico = i1.ID AND a2.dipendente <> d1.ID
  )
  AND
  (
  SELECT SUM(l3.ore) FROM Lavorazione AS l3 WHERE l3.dipendente = d1.ID AND
l3.incarico = i1.ID
  ) > 100
ORDER BY idIncarico;
```

IdCliente	Nominativo	MediaFatture
2	Walter	Lucciano
14	Fiammetta	Nucci
15	Carla	Schiavone

6.4 Procedure e Funzioni

1. Procedura che prende come input un valore ENUM(H, S, A) ad indicare un tipo di mansione o entrambe, una percentuale (negativa o positiva) espressa come decimale ed un valore limite anche'esso decimale. La procedura seleziona tutte le lavorazioni della tipologia fornita e calcola la media del loro costo. Se la percentuale fornita è negativa ed il risultato della media dei costi è minore del limite fornito allora il prezzo delle mansioni relative viene abbassato della percentuale fornita. Se la percentuale è positiva invece e la media maggiore del limite il prezzo viene aumentato della percentuale in input.

```
CREATE PROCEDURE modPrezzoOrario(IN tipoMans ENUM('S', 'H', 'A'), IN per
decimal(4,2), IN lim decimal(12,2))
  BEGIN
    DECLARE mSign tinyint(1);

    IF (per < 0) THEN SET mSign = -1;
    ELSE SET mSign = 1;
    END IF;

    UPDATE Mansione AS m1
    SET m1.prezzoOrario = m1.prezzoOrario + (m1.prezzoOrario * (per/100))
    WHERE (m1.tipo = tipoMans OR IF(tipoMans like 'A', true, false))
    AND (mSign*lim) > mSign*(
       SELECT AVG(l2.costo) FROM Lavorazione AS l2 WHERE l2.mansione = m1.ID
    );
  END //
```

2. Funzione che riceve in input due decimali ed un booleano. Se il booleano è false ritorna la lista degli incarichi per cui è stata emessa una fattura con imponibile compreso tra i due valori forniti, se il booleano è true la verifica dei limiti è fatta tenendo conto anche dell'aliquota.

```
CREATE FUNCTION cercaFattura(FID int(11), impMin decimal(12,2), impMax
decimal(12,2), IVA bool) RETURNS decimal(12,2)
BEGIN
  DECLARE IMP decimal(12,2);
  SET IMP = (SELECT f.imponibile FROM Fattura AS f WHERE f.ID=FID);

  IF (impMin IS NULL) THEN SET impMin = 0;
  END IF;
  IF (impMax = 0) THEN SET impMax = NULL;
  END IF;

  IF (IVA = 0) THEN
    IF ((IMP >= impMin) AND (IFNULL(IMP <= impMax, true))) THEN
      RETURN IMP;
    END IF;
    RETURN NULL;
  ELSE
    SET IVA = (SELECT f.aliquota FROM Fattura AS f WHERE f.ID = FID);
    SET IMP = IMP + (IMP * (IVA/100));
    IF ((IMP >= impMin) AND (IFNULL(IMP <= impMax, true))) THEN
      RETURN IMP;
    END IF;
```

```
    RETURN NULL;
  END IF;
END //
```

3. Funzione che prende in input una città e restituisce la quantità di lavorazioni eseguite da ogni dipendente per clienti che risiedono nella città indicata.

```
CREATE FUNCTION lavorazioniDipCitta(luogo varchar(255)) RETURNS int(11)
BEGIN
  DECLARE LAV int(11);

  SELECT COUNT(*) INTO LAV
  FROM Lavorazione AS l1 JOIN Dipendente AS d1 ON l1.dipendente = d1.ID
    JOIN Incarico AS i1 ON l1.incarico = i1.ID
    JOIN Cliente AS c1 ON i1.cliente = c1.ID
  WHERE c1.citta = luogo
  GROUP BY d1.ID;

  RETURN LAV;
END //
```

6.5 Trigger

Lo sviluppo dei trigger si è concentrato in particolar modo nell'assicurare il mantenimento e l'integrità dei dati rispetto ai vincoli non esprimibili identificati durante la progettazione.

Inoltre sono utilizzati anche per automatizzare il calcolo di alcuni dati, come il costo delle lavorazioni e l'imponibile delle fatture.

```
CREATE TRIGGER before_mansione_insert BEFORE INSERT ON Mansione
  FOR EACH ROW BEGIN
    IF (NEW.tipo = "S") THEN
      SET NEW.costoHW = NULL;
    ELSEIF (NEW.tipo = "H" AND NEW.costoHW IS NULL) THEN
      SET NEW.costoHW = 0;
    END IF;
  END//


CREATE TRIGGER before_mansione_update BEFORE UPDATE ON Mansione
  FOR EACH ROW BEGIN
    IF (NEW.tipo = "S") THEN
      SET NEW.costoHW = NULL;
    ELSEIF (NEW.tipo = "H" AND NEW.costoHW IS NULL) THEN
      SET NEW.costoHW = OLD.costoHW;
    END IF;
  END//


CREATE TRIGGER before_lavorazione_insert BEFORE INSERT ON Lavorazione
  FOR EACH ROW BEGIN
    DECLARE IDM int(11);
    DECLARE TM ENUM('S', 'H');
    DECLARE POM decimal(12,2);
    DECLARE PHM decimal(12,2);
```

```
    DECLARE IDI int(11);
    DECLARE DII date;
    DECLARE IDD int(11);

    SET IDM = NEW.mansione;
    SET TM = (SELECT m.tipo FROM Mansione as m WHERE m.ID = IDM LIMIT 1);

    IF (TM = 'S' AND NEW.quantitaHW IS NOT NULL) THEN
      SET NEW.quantitaHW = NULL;
    ELSEIF (TM = 'H' AND NEW.quantitaHW IS NULL) THEN
      SET NEW.quantitaHW = 1;
    END IF;

    SET POM = (SELECT m.prezzoOrario FROM Mansione as m WHERE m.ID = IDM LIMIT
1);
    SET PHM = (SELECT m.costoHW FROM Mansione as m WHERE m.ID = IDM LIMIT 1);
    SET NEW.costo = (POM * NEW.ore) + (IFNULL(PHM, 0) * IFNULL(NEW.quantitaHW,
0));

    SET IDI = NEW.incarico;
    SET DII = (SELECT i.dataInizio FROM Incarico as i WHERE i.ID = IDI LIMIT 1);

    IF (NEW.dataInizio > NEW.dataFine) THEN
      SET NEW.dataFine = NEW.dataInizio;
    END IF;

    IF (NEW.dataInizio < DII) THEN
      SET NEW.dataInizio = DII;
    END IF;

    IF (NEW.dataFine < NEW.dataInizio) THEN
      SET NEW.dataFine = NEW.dataInizio;
    END IF;

    SET IDD = NEW.dipendente;

    IF ((SELECT a.dataInizio FROM Abilitazione AS a WHERE a.dipendente = IDD AND
a.mansione = IDM) IS NULL) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il dipendente selezionato
non ha l\'abilitzione a svolgere la mansione';
    END IF;

  END//

CREATE TRIGGER before_lavorazione_update BEFORE UPDATE ON Lavorazione
  FOR EACH ROW BEGIN
    DECLARE IDM int(11);
    DECLARE TM ENUM('S', 'H');
    DECLARE IDF int(11);
    DECLARE POM decimal(12,2);
    DECLARE PHM decimal(12,2);
    DECLARE IDD int(11);

    SET IDM = (SELECT l.mansione FROM Lavorazione AS l WHERE l.ID = NEW.ID);
    SET TM = (SELECT m.tipo FROM Mansione as m WHERE m.ID = IDM LIMIT 1);

    IF (TM = 'S' AND NEW.quantitaHW IS NOT NULL) THEN
      SET NEW.quantitaHW = NULL;
    ELSEIF (TM = 'H' AND NEW.quantitaHW IS NULL) THEN
      SET NEW.quantitaHW = OLD.quantitaHW;
    END IF;
```

```
    SET POM = (SELECT m.prezzoOrario FROM Mansione as m WHERE m.ID = IDM LIMIT
1);
    SET PHM = (SELECT m.costoHW FROM Mansione as m WHERE m.ID = IDM LIMIT 1);
    SET NEW.costo = (POM * NEW.ore) + (IFNULL(PHM, 0) * IFNULL(NEW.quantitaHW,
0));

    SET IDF = (SELECT f.ID FROM Fattura as f WHERE f.incarico = NEW.incarico);

    IF (IDF IS NOT NULL) THEN
      BEGIN
        DECLARE TOT decimal(12,2);

        SET TOT = (SELECT SUM(l.costo) FROM Lavorazione as l WHERE l.incarico =
NEW.incarico);
        UPDATE fattura as f SET f.imponibile = TOT WHERE f.incarico =
NEW.incarico;
      END;
    END IF;

    IF (OLD.dataFine <> NEW.dataFine AND NEW.dataFine < NEW.dataInizio) THEN
            SET NEW.dataFine = OLD.dataFine;
    END IF;

    IF (OLD.dataInizio <> NEW.dataInizio) THEN
      IF (NEW.dataInizio > NEW.dataFine) THEN
        SET NEW.dataInizio = OLD.dataInizio;
      END IF;

      BEGIN
        DECLARE IDI int(11);
        DECLARE DII date;

        SET IDI = NEW.incarico;
        SET DII = (SELECT i.dataInizio FROM Incarico as i, Lavorazione as l
WHERE i.ID = IDI LIMIT 1);

        IF (NEW.dataInizio < DII) THEN
          SET NEW.dataInizio = DII;
        END IF;
      END;
    END IF;

    SET IDD = NEW.dipendente;

    IF (NEW.dipendente <> OLD.dipendente) THEN
        IF ((SELECT a.dataInizio FROM Abilitazione AS a WHERE a.dipendente = IDD
AND a.mansione = IDM) IS NULL) THEN
            SET NEW.dipendente = OLD.dipendente;
        END IF;
    END IF;

  END//

CREATE TRIGGER before_incarico_insert BEFORE INSERT ON Incarico
  FOR EACH ROW BEGIN
    IF (NEW.dataFine IS NOT NULL) THEN
      SET NEW.dataFine = NULL;
    END IF;
  END//
```

```
CREATE TRIGGER before_incarico_update BEFORE UPDATE ON Incarico
  FOR EACH ROW BEGIN
    IF (OLD.dataFine IS NULL AND NEW.dataFine IS NOT NULL) OR (OLD.dataFine <>
NEW.dataFine) THEN
      IF (NEW.dataFine < OLD.dataInizio) THEN
        SET NEW.dataFine = OLD.dataFine;
      END IF;

      BEGIN
        DECLARE IDI int(11);
        DECLARE DFL date;

        SET IDI = NEW.ID;
        SET DFL = (SELECT MAX(l.dataFine) FROM Lavorazione as l WHERE l.incarico
= IDI );

        IF (DFL IS NOT NULL) THEN
          IF (NEW.dataFine < DFL) THEN
            SET NEW.dataFine = DFL;
          END IF;
        END IF;
      END;
    END IF;

    IF (OLD.dataInizio <> NEW.dataInizio AND NEW.dataInizio > NEW.dataFine) THEN
      SET NEW.dataInizio = OLD.dataInizio;
    END IF;
  END//

CREATE TRIGGER before_abilitazione_insert BEFORE INSERT ON Abilitazione
  FOR EACH ROW BEGIN
    DECLARE DA int(11);
    DECLARE DAD date;

    SET DA = NEW.dipendente;
    SET DAD = (SELECT d.dataAssunzione FROM Dipendente as d WHERE d.ID = DA);

    IF (NEW.dataInizio < DAD) THEN
      SET NEW.dataInizio = DAD;
    END IF;
  END//

CREATE TRIGGER before_abilitazione_update BEFORE UPDATE ON Abilitazione
  FOR EACH ROW BEGIN
    DECLARE DA int(11);
    DECLARE DAD date;

    SET DA = NEW.dipendente;
    SET DAD = (SELECT d.dataAssunzione FROM Dipendente as d WHERE d.ID = DA);

    IF (NEW.dataInizio < DAD) THEN
      SET NEW.dataInizio = OLD.dataInizio;
    END IF;
  END//

CREATE TRIGGER before_cliente_insert BEFORE INSERT ON Cliente
  FOR EACH ROW BEGIN
    IF (NEW.codiceFiscale IS NOT NULL AND NEW.partitaIVA IS NOT NULL) THEN
      IF (NEW.tipo = 'P') THEN
        SET NEW.partitaIVA = NULL;
      ELSEIF (NEW.tipo = 'I') THEN
```

```
        SET NEW.codiceFiscale = NULL;
      END IF;
    END IF;

    IF (NEW.codiceFiscale IS NULL AND NEW.partitaIVA IS NULL) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '\'codiceFiscale\' e
\'partitaIVA\' non possono essere entrambi NULL';
    END IF;
  END//

CREATE TRIGGER before_cliente_update BEFORE UPDATE ON Cliente
  FOR EACH ROW BEGIN
    IF (NEW.codiceFiscale IS NOT NULL AND NEW.partitaIVA IS NOT NULL) THEN
      IF (NEW.tipo = 'P') THEN
        SET NEW.partitaIVA = NULL;
      ELSEIF (NEW.tipo = 'I') THEN
        SET NEW.codiceFiscale = NULL;
      END IF;
    END IF;
  END//

CREATE TRIGGER before_fattura_insert BEFORE INSERT ON Fattura
  FOR EACH ROW BEGIN
    DECLARE TCL decimal(12,2);

    SET TCL = (SELECT SUM(l.costo) FROM Lavorazione as l WHERE l.incarico =
NEW.incarico);
    SET NEW.imponibile = IFNULL(TCL,0);
  END//
```

7. Interfaccia Web

Questa parte del progetto è stata realizzata solamente da Jordan Gottardo (matricola 1070703) come integrazione per l'esame da 10 crediti.

L'interfaccia web alla base di dati presenta un semplice menù in cui sono elencate le varie tabelle e le possibili operazioni che vi si possono eseguire (inserimenti, modifiche, visualizzazioni). Tramite l'interfaccia di visualizzazione è anche possibile eseguire una cancellazione attraverso un'apposita opzione. E’ stata utilizzata l’estensione MySQLi per interagire con il database. L’interfaccia è dotata di controlli tramite Javascript per evitare che l’utente possa lasciare non valorizzati i campi che non possono essere NULL.