Progetto Basi di Dati
2016/2017

“Gestione della sicurezza interna delle aziende”

Andrea Chinello - Mat. 11436621
Irina Hornoiu - Mat. 1097656

1. Abstract

Il progetto in esame consiste nella realizzazione di una base di dati per la gestione della sicurezza interna dell'azienda.
La piattaforma tiene conto dello storico dei dati riguardanti il personale, in particolare le scadenze dei corsi e visite mediche di ogni lavoratore, e la gestione delle scadenze delle manutenzioni riguardanti l'oggettistica in possesso dell'azienda.

2. Analisi dei requisiti

Si vuole realizzare una base di dati che tenga traccia dello storico delle scadenze legate ai corsi e visite mediche dei lavoratori e delle manutenzioni dell'oggettistica dell'azienda.
Dell'**azienda** si vuole sapere

- il nome, la partita IVA (che la identifica univocamente) e la ragione sociale;

L'azienda può avere varie **Sedi**, delle cui si vuole sapere

- il codice identificativo della sede, l'indirizzo (composto da via, città, regione e stato), se è sede principale oppure secondaria e chi è il direttore;

Dei **lavoratori** si vuole sapere

- il codice identificativo del lavoratore, nome, cognome, data di nascita, codice fiscale, la residenza, l'azienda e la sede presso cui lavora, la mansione, il contratto stipulato con l'azienda, le visite mediche effettuate e i corsi a cui ha partecipato;

Dei **corsi** si vuole sapere

- il codice identificativo del corso, il nome, la data di inizio, la durata, una descrizione degli argomenti trattati, da chi è sostenuto, la data di scadenza della validità del corso e un grado di valutazione del rischio che va a trattare;

I vari **contratti** sono classificati per **tipologia**, per ogni tipologia si vuole sapere

- il nome, la durata in mesi, una descrizione delle clausole contrattuali;

Delle **visite mediche** si vuole sapere

- la data, il lavoratore per cui e' stata svolta la visita, una descrizione dello scopo della visita, il nome del medico (nome e cognome), il risultato della visita (positivo/negativo), eventuali note in caso di esito negativo;

Le sedi possono avere vari **Oggetti** (attrezzi, impianti, ecc.) di cui si vuole sapere

- il codice identificativo dell'oggetto, il nome, la data di acquisto, la sede in cui si trova, la zona dove è collocato e se necessita manutenzione;

Delle **manutenzioni** si vuole sapere

- l'oggetto per cui è stata svolta, la data in cui è stata svolta, il risultato, la data di rinnovo;

I dati dell'azienda vengono gestiti da un utente autorizzato.
Dell'**utente** interessa sapere

- lo username e la password, e le aziende che gestisce.

L'utente può accedere solo ai dati per cui dispone autorizzazione.

3. Progettazione concettuale

3.1 Lista delle classi

UTENTE
La classe Utenti mantiene le informazioni essenziali all'autenticazione di un utente nel sistema.
Attributi:

- *userName: string* l'username corrisponde all'email dell'utente
- *password: string* (con aggiunta di vincoli) una stringa di carattere scelta dall'utente

AZIENDA

La classe Azienda contiene le informazioni essenziali di un'azienda. Le informazioni riguardanti l'indirizzo vengono memorizzate nella classe Sede.

Attributi:

- *nome: string* indica il nome dell'azienda
- *partitaIVA: numeric* siccome la partita IVA è univoca sarà considerata chiave primaria per la classe Azienda
- *ragioneSociale: string* indica la ragione sociale dell'Azienda

SEDE

La classe Sede descrive le sedi di un'Azienda, ne specifica l'indirizzo e il tipo di sede.

Attributi:

- *idSede: int* sarò un codice che identifica ogni sede
- *indirizzo: string*
- *citta: string*
- *prov: var char(2)*
- *stato: var char(2)*
- *tipoSede: enum* indicherà se la sede è una sede 'Principale' oppure 'Secondaria'
- *idDirettore: int* indicherà il codice del direttore presente nella classe Lavoratore.
  l'attributo *idDirettore* è necessario per poter permettere ad un direttore di dirigere piu' di una sede.

OGGETTO

Nella classe Oggetto sono specificati, suddivisi per sede, gli oggetti in possesso di una Azienda. Gli oggetti possono essere di vario tipo (es. ascensore, estintore, attrezzi vari ma anche mobili se l'azienda vuole tenere traccia di uno storico degli acquisti fatti)

Attributi:

- *idOggetto*: *int* codice che identifica l'oggetto
- *dataAcq: date* data di acquisto dell'oggetto
- *nome:string* nome dell'oggetto
- *zonaCollocaz: int* sarà un codice che permette di identificare la collocazione dell'oggetto nella sede
- *controllo: enum* si potranno inserire i valori 'Sì' se l'oggetto avrà bisogno di manutenzione oppure 'No' in caso contrario

MANUTENZIONE

La classe Manutenzione specifica i dettagli delle manutenzioni relative ad un oggetto presente nella classe Oggetto.

Attributi:

- *oggetto: int* indica l'oggetto della classe Oggetto per il quale è stata svolta la manutenzione
- *data: date* indica la data in cui è stata svolta la manutenzione
- *dataRinnovo: date* data in cui dovrà essere svolta la prossima manutenzione
- *risultato: enum* in cui si potrà specificare il valore 'Sicuro' se la manutenzione ha reso sicuro, altrimenti sarà indicato il valore 'Non sicuro'

LAVORATORE

Un lavoratore sarà identificato da un codice e non potrà lavorare in più sedi contemporaneamente però può essere trasferito da una sede ad un'altra (i dati del trasferimento vengono gestiti dalla relazione tra Lavoratore e Sede). Inoltre, ogni lavoratore sarà classificato in 'Manager', 'Quadro', 'Impiegato', 'Operaio' e solo se è 'Manager' potrà diventare 'Direttore'.

Attributi:

- *idLav: int* codice che identifica univocamente ogni impiegato all'interno del db
- *nome: varchar*
- *cognome: varchar*
- *dataNascita: date*
- *codF: string*
- *indirizzo: string*
- *citta: string*
- *prov: varchar*
- *stato: varchar*
- *dataAssunzione: date* indicherà la data in cui è stato assunto dall'azienda
- *dataLicenziamento: date* indicherà la data in cui è stato licenziato dall'azienda
- *rischio: enum* può essere 'Alto', 'Medio', 'Basso' e indica il grado di rischio comportato dalla sua mansione

TIPOCONTRATTO

La classe TipoContratto definisce i tipi di contratto stipulabili.

Attributi:

- *idTipologia: int* codice che identifica il contratto
- *nome: string* indica l'oggetto del contratto
- *durataMesi: tiny int* indica la durata del contratto, il campo può essere lasciato nullo in caso di contratto indeterminato
- *descrizione: string* descrizione delle clausole del contratto

VISITAMEDICA

La classe VisitaMedica indicherà i dettagli delle visite mediche effettuate per i lavoratori della classe Lavoratore.

Attributi:

- *idLav: int* indica il lavoratore presente nella classe Lavoratore per il quale è stata svolta la visita
- *data: date* data in cui e' svolta la visita
- *causaVisita: string* indicherà i motivi per cui è stata svolta la visita
- *nomeMedico: string* indica nome e cognome del medico che ha eseguito la visita
- *risultato: enum* si potrà indicare il valore 'Visita Superata' oppure 'Visita non Superata'
- *note: string* dettagli da indicare in caso il *risultato* della visita sia negativo

CORSO

La classe Corso contiene i dati descrittivi dei corsi.

Attributi:

- *idCorso: int* codice che indentifica ogni corso
- *nome: string*
- *descrizione: string* descrizione degli argomenti che verranno trattati
- *durata: tiny int* durata in ore del corso
- *dataInizio: date* data di inizio del corso
- *dataFine: date* indica la data entro la quale sarà valido il corso
- *validitaRischio: enum* può essere 'Alto', 'Medio', 'Basso' e indica il grado di rischio per cui è consigliato il corso

3.2 Lista delle associazioni

UTENTE-AZIENDA: *Gestisce*

Cardinalità: (1,N) da Utente verso Azienda – (1,1) da Azienda verso Utente

Per ogni azienda, la relazione "Gestisce" identifica l'utente che gestisce i suoi dati. Un utente può gestire più di una azienda mentre un'azienda è gestita solo da un utente.

AZIENDA-SEDE: *Appartiene*

Cardinalità: (1,N) da Azienda verso Sede – (1,1) da Sede verso Azienda

La relazione "Appartiene" associa ogni sede alla propria azienda. Una sede appartiene solo ad una azienda mentre un'azienda può avere più di una sede di cui una principale e le altre secondarie.

SEDE-OGGETTO: *Possiede*

Cardinalità: (1,N) da Sede verso Oggetto – (1,1) da Oggetto verso Sede

La relazione "Possiede" associa gli oggetti alla sede a cui appartengono. Un oggetto appartiene solo ad una sede mentre una sede può avere più di un oggetto.

OGGETTO-MANUTENZIONE: *Necessita*

Cardinalità: (1,N) da Oggetto verso Manutenzione – (1,1) da Manutenzione verso Oggetto

La relazione "Necessita" associa ad ogni oggetto le sue manutenzioni. Un oggetto può avere più istanze di manutenzione mentre una manutenzione è svolta solo per un oggetto.

SEDE-LAVORATORE-TIPOCONTRATTO: *Contratto*

Cardinalità:

(1,N) da Sede verso TipoContratto e Lavoratore

(1,N) da TipoContratto verso Sede e Lavoratore

(1,1) da Lavoratore verso Sede e TipoContratto

1) Una sede può avere più Lavoratori e più Tipologie di Contratto. 2) una Tipologia di Contratto può essere usata per più di un lavoratore in più di una sede mentre 3) un Lavoratore deve avere un solo Contratto definito in TipoContratto e può lavorare in una sola sede.

Attributi dell'associazione "Contratto":

- *dataArrivoSede: date* indica la data in cui il Lavoratore è arrivato in tale sede
- *dataTermineSede: date* indica la data in cui il Lavoratore è stato trasferito dalla sede, se il campo è NULL indica che il lavoratore non è mai stato trasferito da tale sede.

LAVORATORE-CORSO: *Partecipa*

Cardinalità: (0,N) da Lavoratore verso Corso – (0,N) da Corso verso Lavoratore

Per ogni lavoratore identifica i corsi a cui ha partecipato. Un lavoratore può partecipare a zero o più Corsi mentre un corso può essere seguito da zero o più lavoratori.

LAVORATORE-VISITAMEDICA: *Effettua*

Cardinalità: (0,N) da Lavoratore verso VisitaMedica – (1,1) da VisitaMedica verso Lavoratore

Per ogni lavoratore identifica le visite mediche a cui ha partecipato. Un lavoratore può partecipare a zero o più visite mediche mentre una visita medica è svolta per un solo lavoratore.

3.3 Descrizione delle gerarchie

Sono presenti due gerarchie: la prima per la classe Sede e la seconda per la classe Lavoratore.

1) Su Sede è presente una generalizzazione totale che distingue la sede per tipologia. Una sede può essere generalizzata in *Sede Principale* oppure in *Sede Secondaria.*

Sede Principale U Sede Secondaria = Sede

Sede Principale ∩ Sede Secondaria = Ø

2) La classe Lavoratore presenta una generalizzazione totale che la classifica per mansione. Un lavoratore può essere *Manager, Quadro, Impiegato* oppure *Operaio.* Inoltre, è presente anche una

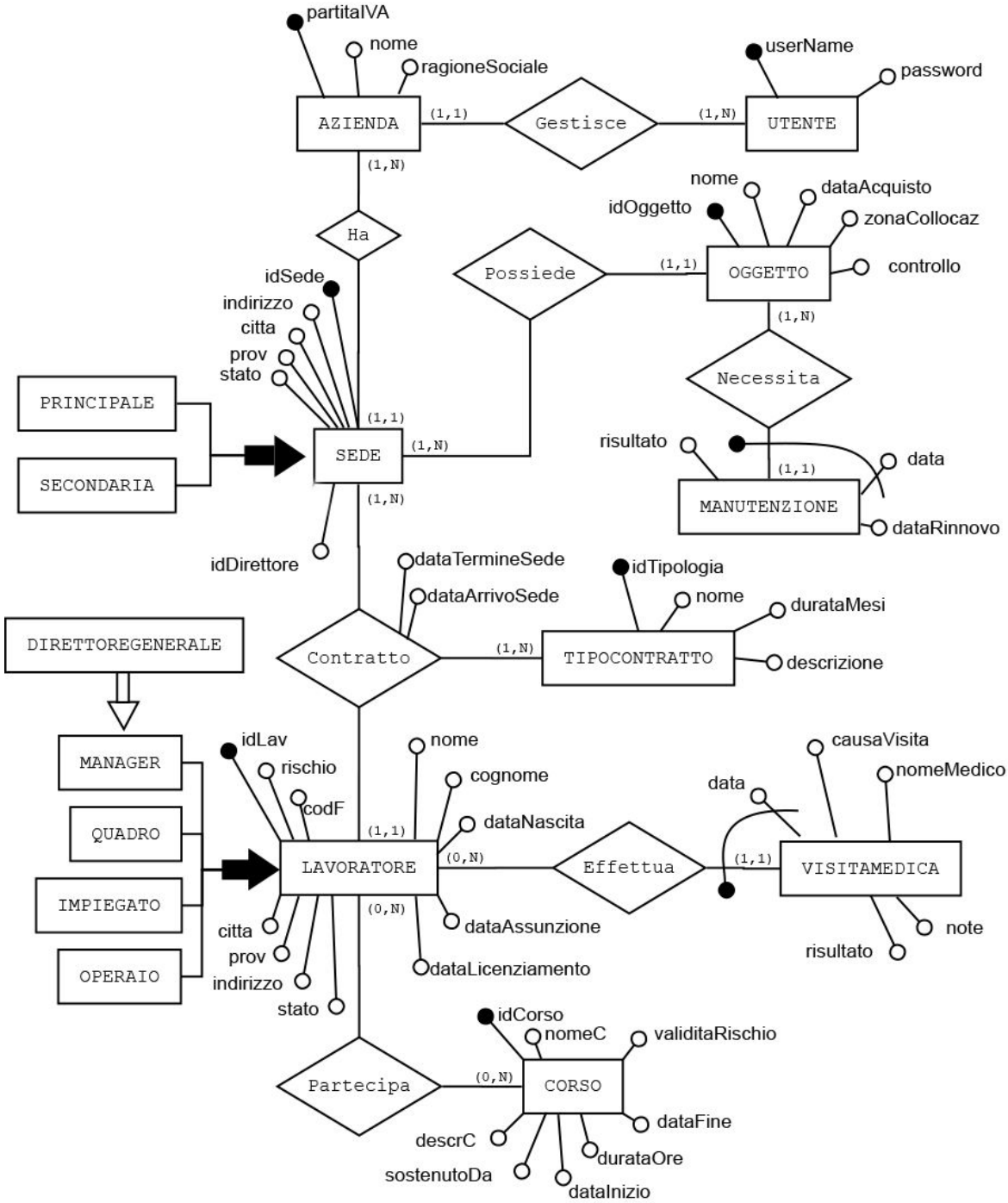
generalizzazione parziale sulla sottoclasse *Manager → Direttore:* un manager può diventare Direttore di una sede.

Direttore è un sottoinsieme di Manager
Manager U Quadro U Impiegato U Operaio = Lavoratore
Manager ∩ Quadro ∩ Impiegato ∩ Operaio = Ø

3.4 Descrizione di vincoli di integrità aggiuntivi

- Utente.username e Utente.password sono due stringhe speciali, username deve corrispondere ad una email mentre la password deve rispettare vincoli come la presenza di almeno un carattere maiuscolo e minuscolo, un carattere speciale e cifre;
- Nell'associazione "Contratto" tra Sede e Lavoratore, la dataTermineSede se inserita deve essere maggiore della dataArrivoSede (vincolo analogo per dataAssunzione e dataLicenziamento in Lavoratore), inoltre se un lavoratore viene licenziato o si licenzia, la dataLicenziamento presente nella classe Lavoratore deve essere copiata in dataTermineSede;
- In TipoContratto durataMesi deve essere un valore positivo oppure nullo in caso di contratto indeterminato;
- Un lavoratore può essere/diventare direttore solo se è manager

3.5 Schema concettuale in forma grafica



4. Progettazione logica

4.1 Ristrutturazione dello schema

Eliminazione delle gerarchie

Gerarchia Lavoratore: Siccome le entità figlie, tranne l'entità 'Manager' che ha una figlia a sua volta, non hanno attributi in più rispetto al Lavoratore classico si è scelto di accorpare tutte le entità figlie nell'entità padre inserendo i seguenti due attributi per distinguerle:

- mansione: enum('Manager', 'Quadro', 'Impiegato', 'Operaio')
- direttore: enum('Sì','No')

Con l'accorpamento delle figlie nel padre sarà necessario un minor numero di accessi e inoltre lo spreco di memoria a causa di valori null è stato limitato con l'uso del tipo enumerazione.
Gerarchia Sede: Anche per questa gerarchia è stato scelto di accorpare le entità figlie nell'entità padre aggiungendo l'attributo tipoSede: enum('Principale','Secondaria').

Traduzione delle associazioni con l'aggiunta delle chiavi esterne

Associazioni (1,1) - (1,N)
»Azienda-Utente ⇒ L'associazione 'Gestisce' viene eliminata; viene inserita la chiave esterna 'gestitaDa' in Azienda.
»Sede-Azienda ⇒ l'associazione 'Appartiene' viene eliminata; viene inserita la chiave esterna 'azienda' in Sede.
»Oggetto-Sede ⇒ l'associazione 'Possiede' viene eliminata; viene inserita la chiave esterna 'sede' in Oggetto.
»Manutenzione-Oggetto ⇒ l'associazione 'Necessita' viene eliminata; viene inserita la chiave esterna 'oggetto' in Manutenzione.

Associazione (1,1) - (0,N)
»VisitaMedica-Lavoratore ⇒ l'associazione 'Effettua' viene eliminata; viene inserita la chiave esterna idLav in VisitaMedica.

Associazione (0,N) - (0,N)
»Lavoratore-Corso ⇒ l'associazione 'Partecipa' diventa una relazione con tre attributi: idLavCorso, idLav e idCorso dove idLavCorso è la chiave primaria che permette di tenere lo storico dei corsi per ogni dimendente mentre idLav è chiave esterna di Lavoratore e idCorso chiave esterna di Corso.

Associazione ternaria
»Sede-Lavoratore-TipoContratto ⇒ la soluzione ottimale è di trasformare l'associazione 'Contratto' memorizzando idSede, idLav e idTipoC come chiavi esterne delle relazioni Sede, Lavoratore e TipoContratto rispettivamente con l'aggiunta degli attributi 'dataArrivoSede' e 'dataTermineSede' che appertengo alla'associazione 'Contratto'.

Scelta delle chiavi primarie

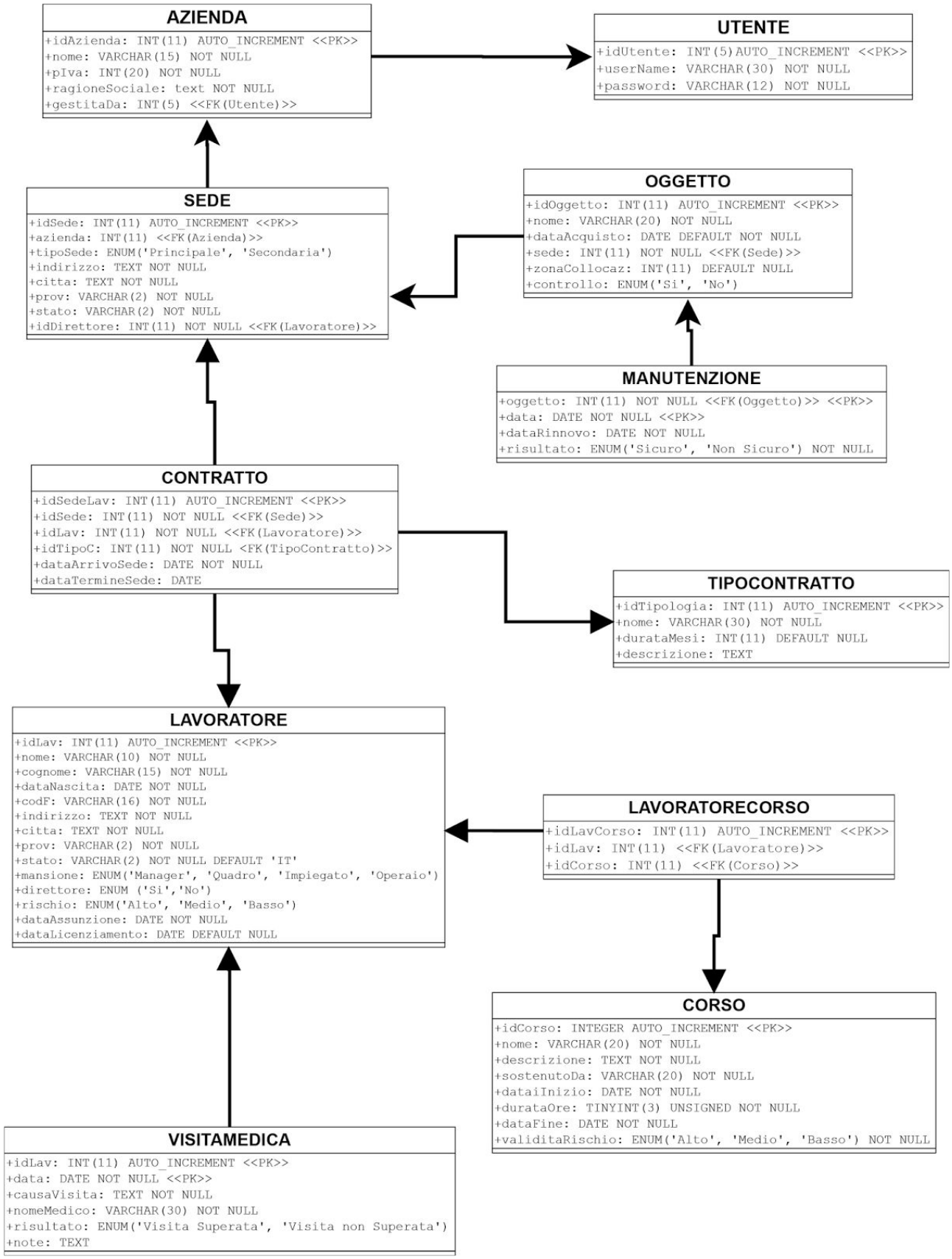
Per agevolare una certa flessibilità nella stesura del codice e semplificare la gestione del database sono state introdotte chiavi artificiali di tipo intero auto-incrementato per ogni relazione del DB, tranne per le relazioni Visita medica e Manutenzione che mantengono come chiave primaria gli attributi data e idLav/oggetto.
In particolare per la relazione LavoratoreCorso è stata inserita la chiave sintetica in quanto l'aggiunta di un attributo era necessaria per poter tenere traccia dello storico dei corsi dei lavoratori.
Il nome delle chiavi sintetiche segue il formalismo 'id'+'parola identificativa della tabella a cui appartiene'.

4.2 Schema relazionale

UTENTE(idUtente, username,password)
AZIENDA(idAzienda, nome, partitaIVA, ragioneSociale, gestitaDa)
SEDE(idSede,azienda,tipoSede, indirizzo, citta, prov, stato, idDirettore)
OGGETTO(idOggetto, nome, dataAcquisto, sede, zonaCollocaz, controllo)
MANUTENZIONE(oggetto, data, dataRinnovo, risultato)
CONTRATTO(idSedeLav, idSede, idLav, idTipoC, dataArrivoSede, dataTermineSede)
LAVORATORE(idLav,nome,cognome,dataNascita, codF,indirizzo, citta, prov, stato,
mansione, direttore, rischio, dataAssunzione, dataLicenziamento)
LAVORATORECORSO(idLavCorso, idLav, idCorso)
CORSO(idCorso, nome, descrizione, sostenutoDa, dataInizio, durataOre, dataFine,
validitaRischio)
VISITAMEDICA(idLav, data, causaVIsita, nomeMedico, risultato, note)

Azienda.gestitaDa è in integrità referenziale con *Utente.idUtente.*
Sede.azienda è in integrità referenziale con *Azienda.idAzienda.*
Oggetto.sede è in integrità referenziale con *Sede.idSede.*
Manutenzione.oggetto è in integrità referenziale con *Oggetto.idOggetto.*
Contratto.idSede è in integrità referenziale con *Sede.idSede.*
Contratto.idLav è in integrità referenziale con con *Lavoratore.idLav.*
Contratto.idTipoC è in integrità referenziale con *TipoContratto.idTipologia.*
LavoratoreCorso.idLav è in integrità referenziale con *Lavoratore.idLav.*
LavoratoreCorso.idCorso è in integrità referenziale con *Corso.idCorso.*
VisitaMedica.idLav è in integrità referenziale con *Lavoratore.idLav.*

4.3 Schema logico in forma grafica



5. Query, Trigger e Funzioni

5.1 Query

1. Per ogni azienda trovare i lavoratori attualmente assunti che hanno un corso in scadenza tra meno 2 mesi
    - SELECT s.azienda, s.idSede, l.idLav, l.nome, l.cognome, c.idCorso
      FROM Sede s JOIN Contratto co ON s.idSede=co.idSede
      JOIN Lavoratore l ON co.idLav=l.idLav
      JOIN LavoratoreCorso lc ON l.idLav=lc.idLav
      JOIN Corso c ON lc.idCorso=c.idCorso
      WHERE l.dataLicenziamento is null AND DATE(DATE_ADD(CURRENT_DATE(), INTERVAL 2 MONTH)) > c.dataFine AND c.dataFine>=CURRENT_DATE();
2. Per ogni azienda trovare i lavoratori attualmente assunti che devono rifare almeno una visita medica
    - SELECT s. azienda, s.idSede, l.idLav, l.nome, l.cognome
      FROM Sede s JOIN Contratto c ON s.idSede=c.idSede
      JOIN Lavoratore l ON c.idLav=l.idLav
      JOIN VisitaMedica v ON l.idLav=v.idLav
      WHERE v.risultato='Visita non Superata' AND l.dataLicenziamento is null;

3. Per ogni utente calcolare quanti lavoratori hanno partecipato ad un determinato corso (es corso 1)
    - SELECT u.username, count(l.idLav)
      FROM Utente u JOIN Azienda a ON u.idUtente=a.gestitaDa
      JOIN Sede s ON s.azienda=a.idAzienda
      JOIN Contratto cn ON s.idSede=cn.idSede
      JOIN Lavoratore l ON cn.idLav=l.idLav
      JOIN LavoratoreCorso lc ON l.idLav=lc.idLav
      WHERE lc.idCorso=1
      GROUP BY u.username;
4. Per ogni azienda e per ogni sua sede trovare la mansione piu' frequente
      SELECT Ta.a as idAzienda ,Ta.se as idSede , Ta.ma as Mansione , max(Ta.cm) as NumeroComparse
      FROM
      (SELECT s.idSede as se ,COUNT(l.mansione) as cm , l.mansione as ma , s.azienda as a
      FROM Lavoratore l
      JOIN Contratto c ON l.idLav = c.idLav
      JOIN Sede s ON s.idSede = c.idSede
      GROUP BY s.idSede , l.mansione
) as Ta
      GROUP BY Ta.se;
5. Trovare i lavoratori che hanno partecipato ad almeno un corso con difficolta' rischio maggiore o uguale al rischio determinato dalla sua mansione

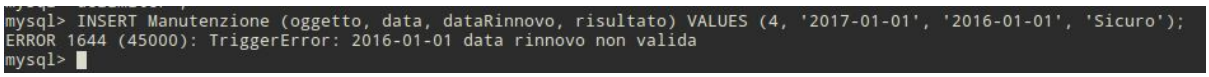
```
SELECT l.idLav , l.nome , l.cognome
FROM Corso c JOIN LavoratoreCorso lc ON c.idCorso = lc.idCorso
                    JOIN Lavoratore l ON lc.idLav = l.idLav
WHERE l.mansione = c.validitaRischio  OR (l.Rischio = 'Basso'
        AND c.validitaRischio = 'Medio') OR (l.Rischio = 'Basso'
        AND c.validitaRischio = 'Alto') OR (l.Rischio = 'Medio'
        AND c.validitaRischio = 'Alto')
GROUP BY l.idLav;
```

6. Per una determinata azienda trovare i lavoratori che non fanno parte di una determinata mansione (esempio azienda=2 AND mansione='Operaio')
    - SELECT c.idSede, l.idLav, l.nome, l.cognome, l.mansione
      FROM Sede s JOIN Contratto c ON s.idSede=c.idSede
            JOIN Lavoratore l ON c.idLav=l.idLav
      WHERE s.azienda=2 AND l.mansione<> 'Operaio';

5.2 Trigger

1. **Controllo che in manutenzione la dataRinnovo sia maggiore della data in cui è stata effettuata la manutenzione**

```
DROP TRIGGER IF EXISTS CheckManutenzione;
DELIMITER /
CREATE TRIGGER  CheckManutenzione
BEFORE INSERT ON Manutenzione FOR EACH ROW
BEGIN
        DECLARE msg varchar(128);
        if ( new.data>new.dataRinnovo) then
                set msg = concat('TriggerError: ', cast(new.dataRinnovo as date),
                                'data rinnovo non valida');
                signal sqlstate '45000' set message_text = msg;                 end if;
END /
DELIMITER ;
```



2. **Se aggiorno dataTermineSede controllo che sia maggiore di dataArrivoSede**

```
DROP TRIGGER IF EXISTS CheckTrasferimento;
DELIMITER /
CREATE TRIGGER CheckTrasferimento
BEFORE UPDATE ON Contratto FOR EACH ROW
BEGIN
        DECLARE msg varchar(128);
        if ( new.dataTermineSede<new.dataArrivoSede) then
                set msg = 'TriggerError: Inserisci una data trasferimento valida';
                signal sqlstate '45002' set message_text = msg;         end if;
END /
DELIMITER ;
```

```
+------------+---------+-------+-------------+-----------------+------------------+
| idSedeLav | idSede | idLav | idContratto | dataArrivoSede | dataTermineSede |
+------------+---------+-------+-------------+-----------------+------------------+
|          4 |       2 |     4 |           6 | 1990-01-01      | NULL             |
+------------+---------+-------+-------------+-----------------+------------------+
1 row in set (0,00 sec)

mysql> UPDATE Contratto SET datatermineSede='1989-12-01' WHERE idSedeLav=4;
ERROR 1644 (45002): TriggerError: Inserisci una data trasferimento valida
mysql> SELECT * FROM Contratto WHERE idSedeLav=4;
+------------+---------+-------+-------------+-----------------+------------------+
| idSedeLav | idSede | idLav | idContratto | dataArrivoSede | dataTermineSede |
+------------+---------+-------+-------------+-----------------+------------------+
|          4 |       2 |     4 |           6 | 1990-01-01      | NULL             |
+------------+---------+-------+-------------+-----------------+------------------+
1 row in set (0,00 sec)

mysql> UPDATE Contratto SET datatermineSede='2000-01-01' WHERE idSedeLav=4;
Query OK, 1 row affected (0,06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Contratto WHERE idSedeLav=4;
+------------+---------+-------+-------------+-----------------+------------------+
| idSedeLav | idSede | idLav | idContratto | dataArrivoSede | dataTermineSede |
+------------+---------+-------+-------------+-----------------+------------------+
|          4 |       2 |     4 |           6 | 1990-01-01      | 2000-01-01       |
+------------+---------+-------+-------------+-----------------+------------------+
```

3. **Quando aggiorno il campo dataLicenziamento in Lavoratore, copio la data anche in Contratto.dataTermineSede**

```
DROP TRIGGER IF EXISTS integraLicenziamento;
delimiter /
CREATE TRIGGER  integraLicenziamento
AFTER UPDATE ON Lavoratore FOR EACH ROW
BEGIN
        UPDATE Contratto
        SET dataTermineSede=new.dataLicenziamento
        WHERE Contratto.idLav=new.idLav AND Contratto.dataTermineSede IS NULL;
END /
delimiter ;
```

```
+-------+-----------+---------+----------------+-------------------+
| idLav | nome      | cognome | dataAssunzione | dataLicenziamento |
+-------+-----------+---------+----------------+-------------------+
|     6 | asdrubale | carli   | 2016-01-03     | NULL              |
+-------+-----------+---------+----------------+-------------------+
1 row in set (0,00 sec)

mysql> SELECT * FROM Contratto WHERE idLav=6;
+-----------+---------+-------+-------------+-----------------+------------------+
| idSedeLav | idSede | idLav | idContratto | dataArrivoSede | dataTermineSede |
+-----------+---------+-------+-------------+-----------------+------------------+
|         6 |       2 |     6 |           1 | 1990-05-01      | NULL             |
+-----------+---------+-------+-------------+-----------------+------------------+
1 row in set (0,00 sec)

mysql> UPDATE Lavoratore SET dataLicenziamento='2017-01-01' WHERE idLav=6;
Query OK, 1 row affected (0,05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT idLav, nome,cognome, dataAssunzione, dataLicenziamento  FROM Lavoratore WHERE idLav=6;
+-------+-----------+---------+----------------+-------------------+
| idLav | nome      | cognome | dataAssunzione | dataLicenziamento |
+-------+-----------+---------+----------------+-------------------+
|     6 | asdrubale | carli   | 2016-01-03     | 2017-01-01        |
+-------+-----------+---------+----------------+-------------------+
1 row in set (0,00 sec)

mysql> SELECT * FROM Contratto WHERE idLav=6;
+-----------+---------+-------+-------------+-----------------+------------------+
| idSedeLav | idSede | idLav | idContratto | dataArrivoSede | dataTermineSede |
+-----------+---------+-------+-------------+-----------------+------------------+
|         6 |       2 |     6 |           1 | 1990-05-01      | 2017-01-01       |
+-----------+---------+-------+-------------+-----------------+------------------+
```

4. **Se inserisco una nuova sede e la imposto come sede principale, controllo se c'è un'altra sede principale per tale azienda**

```
DROP TRIGGER IF EXISTS CheckSediPrincipali;
DELIMITER /
CREATE TRIGGER CheckSediPrincipali
BEFORE INSERT ON Sede FOR EACH ROW
BEGIN
        DECLARE n,idS INT;
        DECLARE msg VARCHAR(128);
        if (new.tipoSede='Principale') then
                SELECT count(Sede.idSede) INTO n
                FROM Sede
                WHERE Sede.azienda=new.azienda AND
                        Sede.tipoSede='Principale';
                if (n>=1) then
                        signal sqlstate '45005' set message_text = 'Trigger Error: E' già presente una sede impostata come Principale';
                end if;
        end if;
END /
DELIMITER ;
```

```
+--------+---------+------------+---------------------------------+---------+------+-------+-------------+
| idSede | azienda | principale | indirizzo                       | citta   | prov | stato | idDirettore |
+--------+---------+------------+---------------------------------+---------+------+-------+-------------+
|      1 |       1 | Principale | via Tiepolo 955                 | Brugine | PD   | IT    |           1 |
|      2 |       1 | Secondaria | via Frassignoni 23              | Legnaro | PD   | IT    |           1 |
|      3 |       2 | Principale | via delle industrie 14/F        | Treviso | TV   | IT    |           7 |
|      4 |       2 | Secondaria | via san benedetto di assisi 666 | Aosta   | AO   | IT    |           1 |
+--------+---------+------------+---------------------------------+---------+------+-------+-------------+
4 rows in set (0,01 sec)

mysql> INSERT INTO Sede (azienda, principale, indirizzo, citta, prov, stato, idDirettore) VALUES (1,'Principale','x','x','x','IT',12);
ERROR 1644 (45005): Trigger Error: E' già presente una sede impostata come Principale
mysql> INSERT INTO Sede (azienda, principale, indirizzo, citta, prov, stato, idDirettore) VALUES (1,'Secondaria','x','x','x','IT',12);
Query OK, 1 row affected (0,05 sec)
```

5.3 Funzioni

1. **Data una azienda trovare l'id del direttore della sua sede principale**

```
DROP FUNCTION IF EXISTS direttore;
DELIMITER /
CREATE FUNCTION direttore(Aid INT)
RETURNS INT
BEGIN
        DECLARE direttore INT;
        SELECT l.idLav INTO direttore
        FROM Lavoratore l JOIN Contratto c ON l.idLav = c.idLav
                JOIN Sede s ON c.idSede = s.idSede
        WHERE s.azienda=Aid AND s.tipoSede='Principale' AND l.idLav=s.idDirettore;
        RETURN direttore ;
END/
DELIMITER ;
//Utilizzo funzione
```

```
SELECT * FROM Lavoratore WHERE idLav=direttore(1);
```

2. Dato un rischio trovare l'azienda che ha il maggior numero di lavoratori che hanno partecipato a corsi di tale rischio

```
DROP FUNCTION IF EXISTS aziende;
DELIMITER /
CREATE FUNCTION aziende(rischio CHAR(5) ) RETURNS INT
BEGIN
DECLARE  countAzienda, ris, minimo , v  INT;
DECLARE curID INT DEFAULT 1;
SET countAzienda = (SELECT COUNT(idAzienda ) from Azienda );
SET minimo = 0 ;
REPEAT
        SELECT COUNT(l.idLav) INTO ris
        FROM  Sede s
        JOIN Contratto c ON s.idSede = c.idSede
        JOIN Lavoratore l ON l.idLav = c.idLav
        JOIN LavoratoreCorso lc ON lc.idLav=l.idLav
        JOIN Corso cs ON cs.idCorso= lc.idCorso
        WHERE s.azienda = curID AND cs.validitaRischio=rischio ;
        if (ris > minimo) then
                set minimo = ris;
                set v= curID;
        end if;
        set curID = curID +1;
        UNTIL curID <= countAzienda
END REPEAT;
RETURN v;
END /
DELIMITER ;
//Utilizzo funzione
SELECT nome FROM Azienda WHERE idAzienda= aziende('Alto');
```