GRUPPO 8

di Davide Sapienza,

Marco Torreggiani e

Alberto Vezzani

Progetto originale

Si vuole realizzare un'applicazione database per gestire le informazioni relative all'utilizzo dei locali e delle risorse di un'Università da parte degli studenti dei Corsi di Laurea e del personale, docente e non docente, dell'Università stessa.

Gli accessi ad alcuni locali dell'Università sono riservati a persone autorizzate in base alle seguenti modalità. Un utente può avere per un locale un solo permesso. Ogni permesso appartiene ad una certa tipologia che specifica tra l'altro, per ciascuno dei sei giorni lavorativi, gli intervalli temporali di accesso. Gli accessi sono controllati e rilevati tramite la lettura di una tessera magnetica assegnata a ciascun utente.

Vi sono alcuni locali per cui bisogna tenere traccia di tutti gli accessi, rilevando per ogni utente: data, ora e identificazione dell'utente. Per altri locali bisogna tenere traccia solo del numero di accessi giornalieri. Tutte le volte che il sistema rifiuta l'accesso ad un locale, si deve tenere traccia dell'evento, memorizzando tutte le informazioni relative. In particolare, interessa evidenziare i casi in cui per uno stesso utente e per lo stesso locale ci siano stati più di tre rifiuti giornalieri.

Tra i locali dell'Università vi sono dei laboratori didattici che contengono un insieme di posti di lavoro ed un insieme di risorse. Ad ogni posto di lavoro sono assegnate alcune risorse (quali ad esempio, unità di calcolo, stampanti, software applicativo). Alcuni posti di lavoro sono resi disponibili a tutti gli studenti senza controlli, altri vengono mensilmente assegnati a particolari studenti, quali ad esempio laureandi, previa autorizzazione di un docente. Infine, in un laboratorio vi possono essere un certo numero di posti di lavoro prenotabili giornalmente da parte di un singolo studente oppure da parte di un docente titolare di un insegnamento; per questi posti di lavoro si deve tenere traccia di tutte le prenotazioni e di tutti le utilizzazioni da parte degli studenti. Per le prenotazioni devono essere garantite le seguenti regole di non sovrapposizione:

1. in una certa ora, un posto può essere prenotato da un solo studente;

2. in una certa ora, uno studente non può avere la prenotazione per due posti differenti.

Ogni laboratorio ha come responsabile organizzativo un docente e come responsabile operativo un tecnico dell'Università.

Alcune unità di calcolo richiedono un account per accedervi; gli account vengono rilasciati automaticamente, su un insieme di server, agli studenti sulla base della loro iscrizione ad un corso di laurea e all'anno di iscrizione. A ciascuno dei gruppi così individuato il System Administrator assegna e gestisce determinati privilegi e risorse. Per ogni account viene gestita la configurazione del profilo utente, memorizzando informazioni quali lo "shell" e gli applicativi utilizzati.

Interpretazione

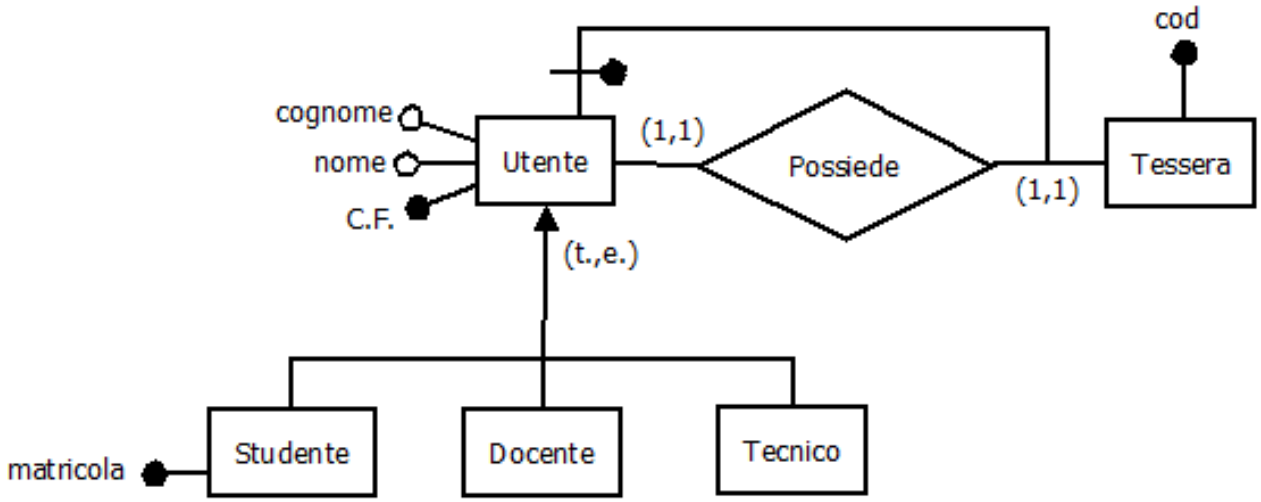
Glossario del progetto:

TERMINE	DESCRIZIONE	SINONIMI	LEGAME
Locali	Nome, descrizione, codice,		Riservato Tracciato Non tracciato Laboratori
Riservato	Cod-locali		Locali Permesso Accesso negato
Tracciato	Cod-locali		Locali Accesso
Non tracciato	Cod-locali		Locali N-accessi
Laboratori	Cod-locali		Locali Posti di lavoro Docente Tecnico
Utente	Nome, cognome, CF, cod-tessera		Studente Docente Tecnico Tessera Permesso Accesso negato Prenotazione
Studente	Matricola, CF		Utente Assegnati Account Libero
Docente	CF		Utente Assegnati Laboratori
Tecnico	CF		Utente Laboratori
Tessera	Cod		Utente Accesso
Permesso	CF-utente, cod-Riservato		Riservato Utente Tipo
Tipo	Tipo		Permesso Fasce

Fasce	Giorno,ora-i,ora-f	Intervalli temporale	Tipo
Accesso negato	tessera-utente, cod-Riservato, ora, data		Utenti Riservato
Tre Accessi Negati	Cod-Riservato, data, tessera-utente		Riservato, utenti
Accesso	Cod-tessera, ora, data		Tracciato Tessera
N-Accessi	Cod-Nontracciato, data, num-accessi		Non tracciato
Posti di lavoro	Cod-Laboratori, numero		Laboratori Prenotabili Previo autorizzazione Libero Risorse
Prenotabili	Cod-Laboratori, n-posto		Posti di lavoro Prenotazione
Previo autorizzazione	Cod-Laboratori, n-posto		Posti di lavoro Assegnati
Libero	Cod-Laboratori, n-posto		Posti di lavoro Studente
Risorse	Cod, descrizione		Posti di lavoro Unità calcolo con account Privilegi
Unità calcolo con account	Cod-risorse		Risorse Account
Privilegi	Cod, tipo, cod-risorse		Risorse Gruppo
Account	Utente, password		Unità calcolo con account Gruppo Studente
Gruppo	Anno, codl		Account Privilegi Corso laurea
Corso laurea	Codl		Gruppo
Prenotazione	CF, data, orai, oraf, n_posto, cod_locale		Utente Prenotabili
Assegnati	Mese, nposto, cod_locale		Docente Studente Previo autorizzazione

Progetto concettuale

Abbiamo deciso di rappresentare tutto il personale dell'università, identificato dai consueti dati anagrafici, come entità figlie di un unico padre.

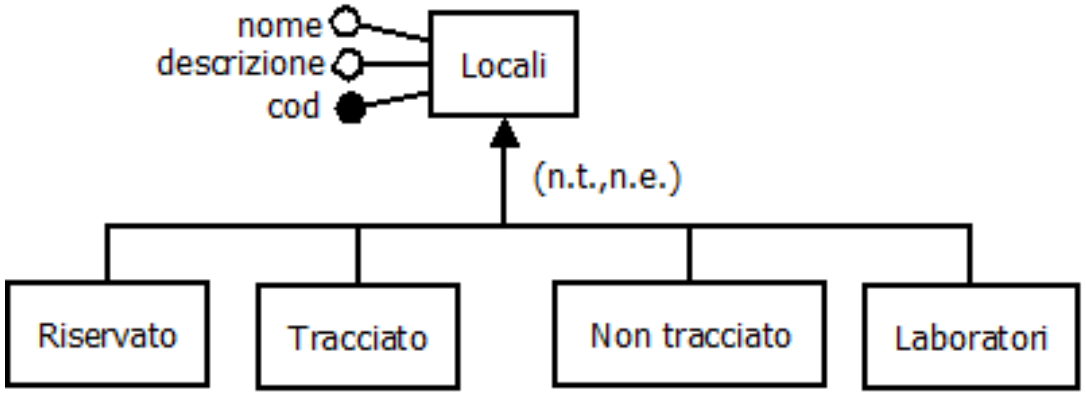


La gerarchia è di tipo totale in quanto gli utenti devono essere necessariamente classificati come studenti, docenti o tecnici. Inoltre è esclusiva in quanto un utente non può rivestire ruoli diversi contemporaneamente.

Seguendo le specifiche ogni utente possiede una tessera necessaria per accedere ad alcuni locali.

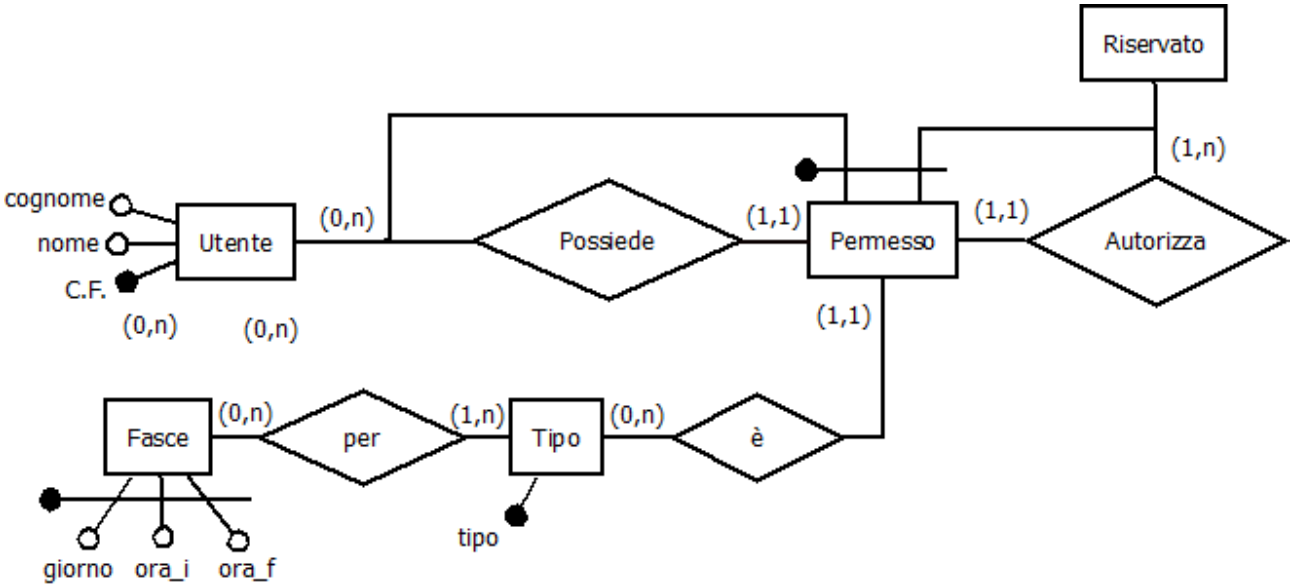
La cardinalità di questa relazione è uno a uno in quanto ogni utente possiede una sola tessera e questa tessera è personale per questo il codice della tessera poteva essere direttamente incluso nell'entità utente ma in questa fase del progetto ci è sembrato più chiaro mantenere le due entità separate.

Per quanto riguarda i locali abbiamo deciso di utilizzare una gerarchia in modo da distinguere i vari sottoinsiemi.



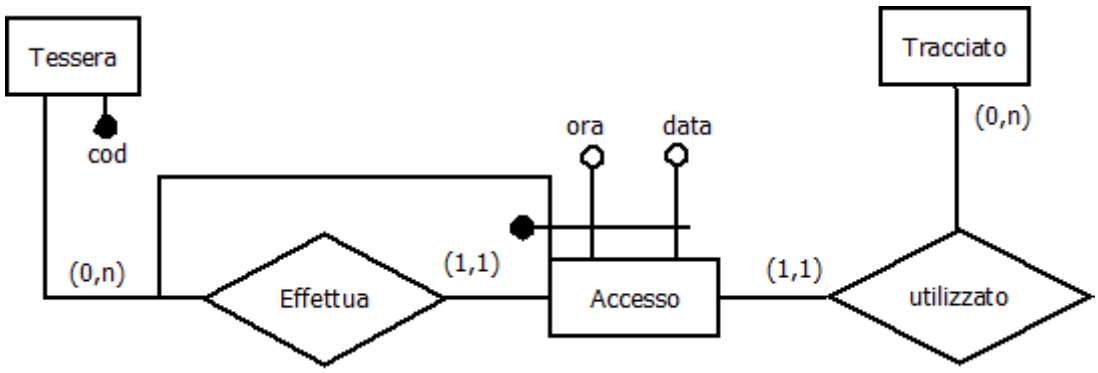
In questo caso la gerarchia è non totale, in quanto un locale potrebbe non rientrare in nessuna delle categorie specificate, e non esclusiva, in quanto un locale potrebbe richiedere sia una tracciatura degli accessi, che un laboratorio.

Procediamo all'analisi dei tipi di locali. Per esempio, per quelli riservati è necessario avere un permesso d'accesso.



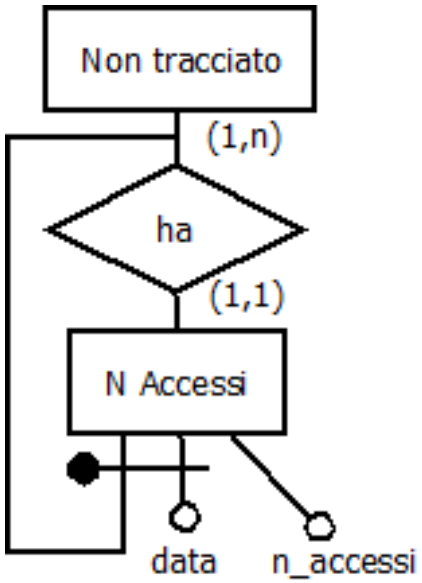
In questa parte, viene implementato il concetto di "permesso" che permette all'utente di accedere a determinati locali. Supponiamo che un utente possa avere più permessi (ovviamente per locali diversi) e per questo motivo la cardinalità di Utente è (0,n). Inoltre il permesso è di un solo tipo, ma abilitato per fasce diverse. Una fascia, infine, non è altro che un giorno (es lunedì), un orario di inizio validità del permesso (es 09.00) e un orario di fine validità (es 11.00). In questo modo l'utente X sarà abilitato ad accedere al locale Y il lunedì dalle 9.00 alle 11.00.

Per quanto riguarda i locali tracciati abbiamo deciso di registrare ogni accesso e, per esso, sia la data che l'ora in cui è stato effettuato.



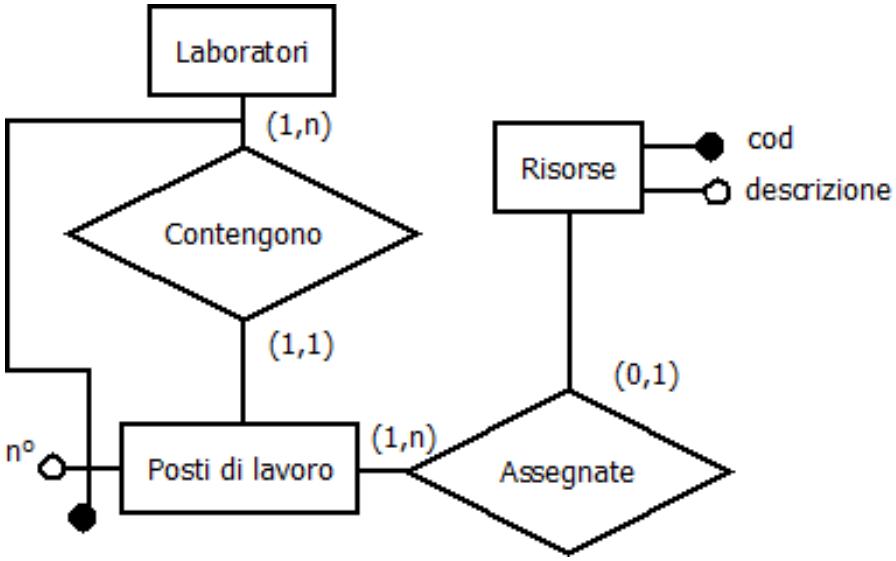
Come vincolo aggiuntivo abbiamo introdotto che un utente non possa trovarsi in due locali diversi nello stesso istante, per questo abbiamo scelto come chiave codtessera ora e data. In questo modo si possono registrare tutti gli accessi ad un locale. Teoricamente parlando due utenti possono accedere a un locale anche contemporaneamente (se supponiamo di avere due lettori di tessere o due ingressi al locale).

Per quanto riguarda i locali di cui si vuole tener traccia solo del numero di accessi effettuati abbiamo risolto in questo modo:



L'entità N Accessi registra il numero di accessi ad ogni locale. Per questo abbiamo deciso di mettere in chiave data e codlocale; in questo modo per ogni giorno è possibile memorizzare il numero degli accessi effettuati.

I laboratori mettono a disposizione dei posti di lavoro numerati, ai quali sono assegnate delle risorse (stampanti, computer, lampade da lettura, …)

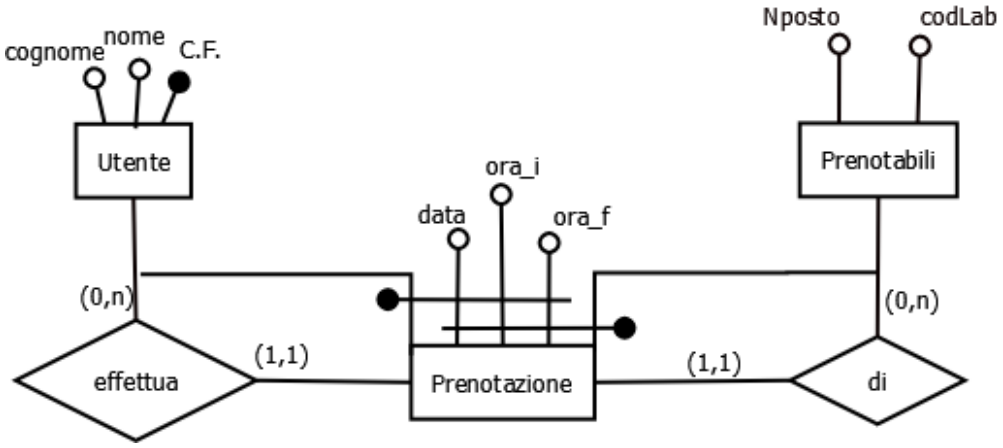


La cardinalità di Laboratorio è (1,n), in quanto un laboratorio ha sicuramente uno o più posti di lavoro, mentre un posto di lavoro appartiene ad un solo laboratorio (1,1); la chiave di posto di lavoro è: codlocale, n.

Sui posti di lavoro possono essere eseguite operazioni differenti, per questo abbiamo deciso di classificare i posti di lavoro.

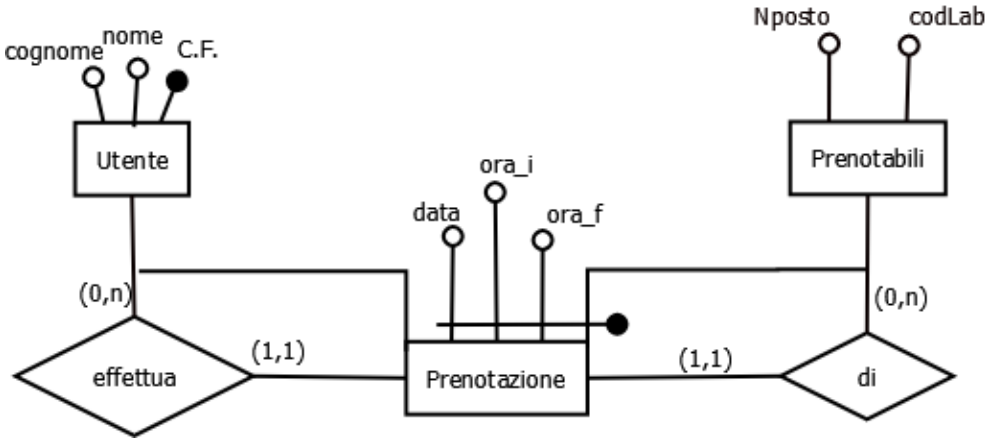
La gerarchia è totale ed esclusiva: tutti i posti di lavoro devono rientrare in una delle 3 categorie e una categoria esclude l'altra. (Un posto di lavoro che richiede un'autorizzazione non può essere ad accesso libero).

Per quanto riguarda i posti di lavoro prenotabili scegliamo di dare il diritto di prenotazione a qualunque utente del database:

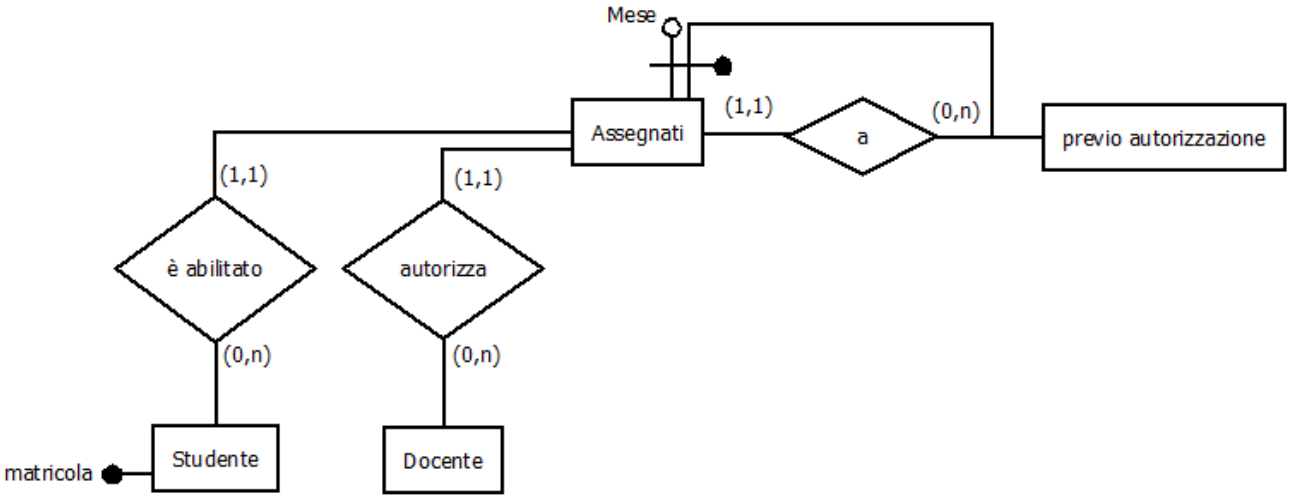


Nell'entità Prenotazione vengono registrate tutte le prenotazioni evitando sovrapposizioni (grazie alla chiave n, data, ora_i e ora_f) e evitando anche che lo stesso utente prenoti posti diversi nello stesso arco temporale (grazie alla chiave C.F., data, ora_i e ora_f).

E' richiesto però, di poter effettuare prenotazioni multiple da parte di un docente, allora decidiamo di eliminare la chiave alternativa (C.F., data, ora_i e ora_f) e di gestire il controllo attraverso un trigger.



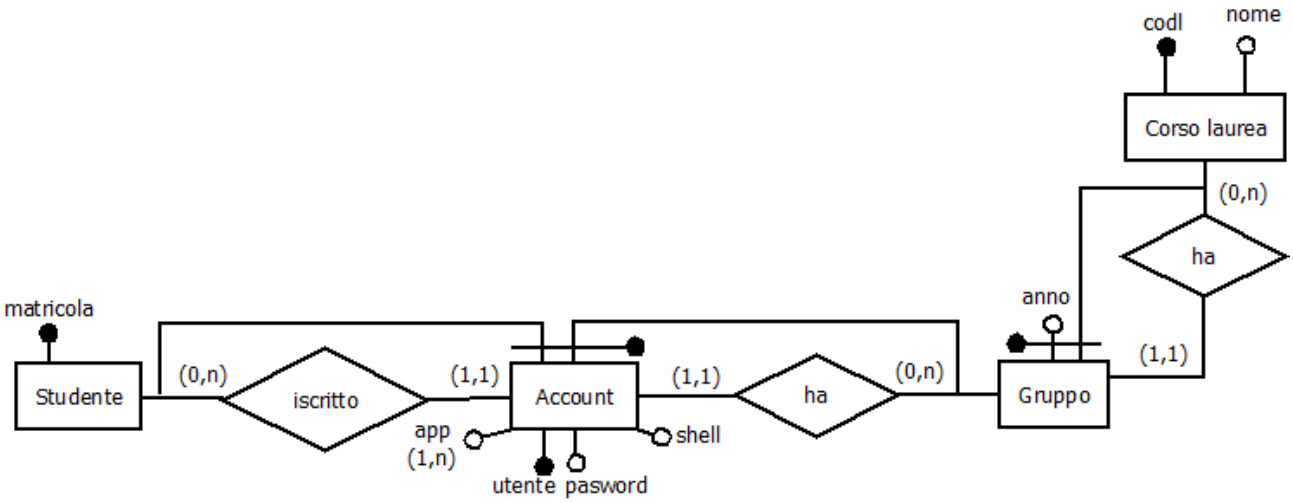
Per i posti di lavoro che vengono assegnati a studenti previo l'autorizzazione del docente, per un dato mese, abbiamo pensato di risolvere come segue:



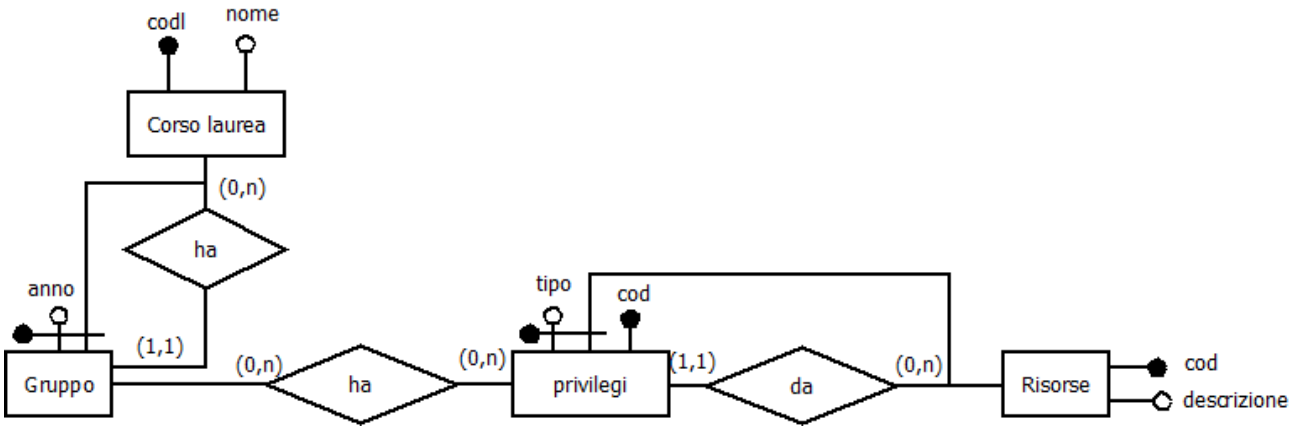
Lo studente viene autorizzato da un docente in un dato mese per un certo posto. Abbiamo deciso di mettere in chiave mese e n° posto-locale per evitare che lo stesso posto nello stesso mese possa essere assegnato a studenti differenti.

La gerarchia esclusiva dei posti di lavoro impedisce conflitti tra posti assegnati e prenotati.

Abbiamo considerato che uno studente possa avere più account: uno studente che cambia facoltà mantiene la matricola, ma cambia account; uno studente iscritto a più corsi di laurea ha più account.

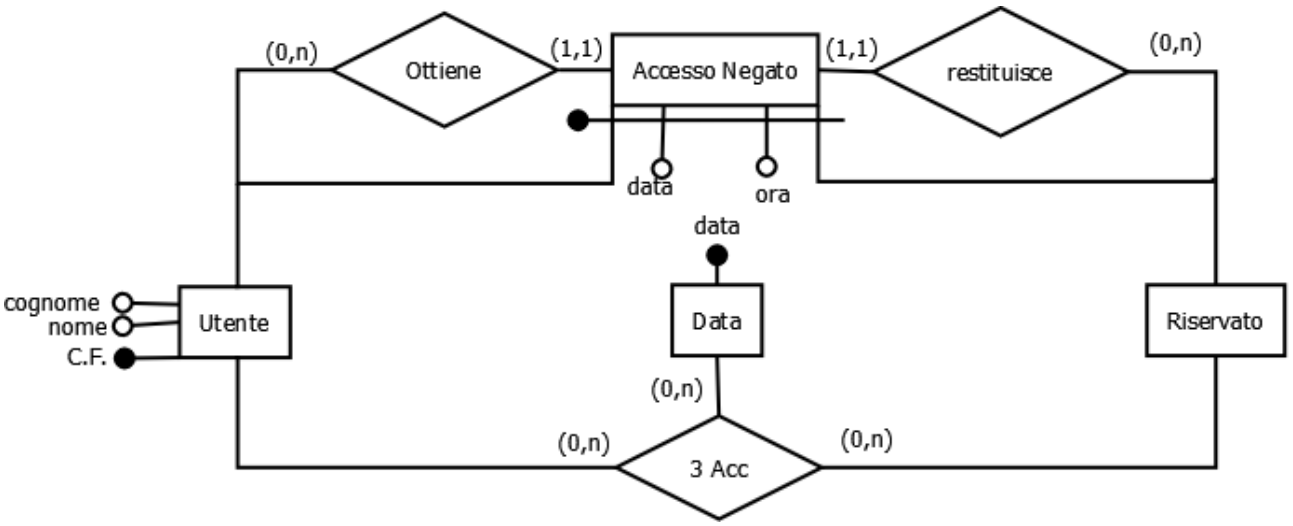


Un account è quindi identificato dal nome utente e come chiave alternativa ha matricola, anno, e codl, per riuscire a rispettare il vincolo: “Gli account vengono rilasciati […] agli studenti sulla base della loro iscrizione ad un corso di laurea e all’anno di iscrizione”. Inoltre un account fa parte di un solo gruppo (identificato come corso di laurea in un dato anno).

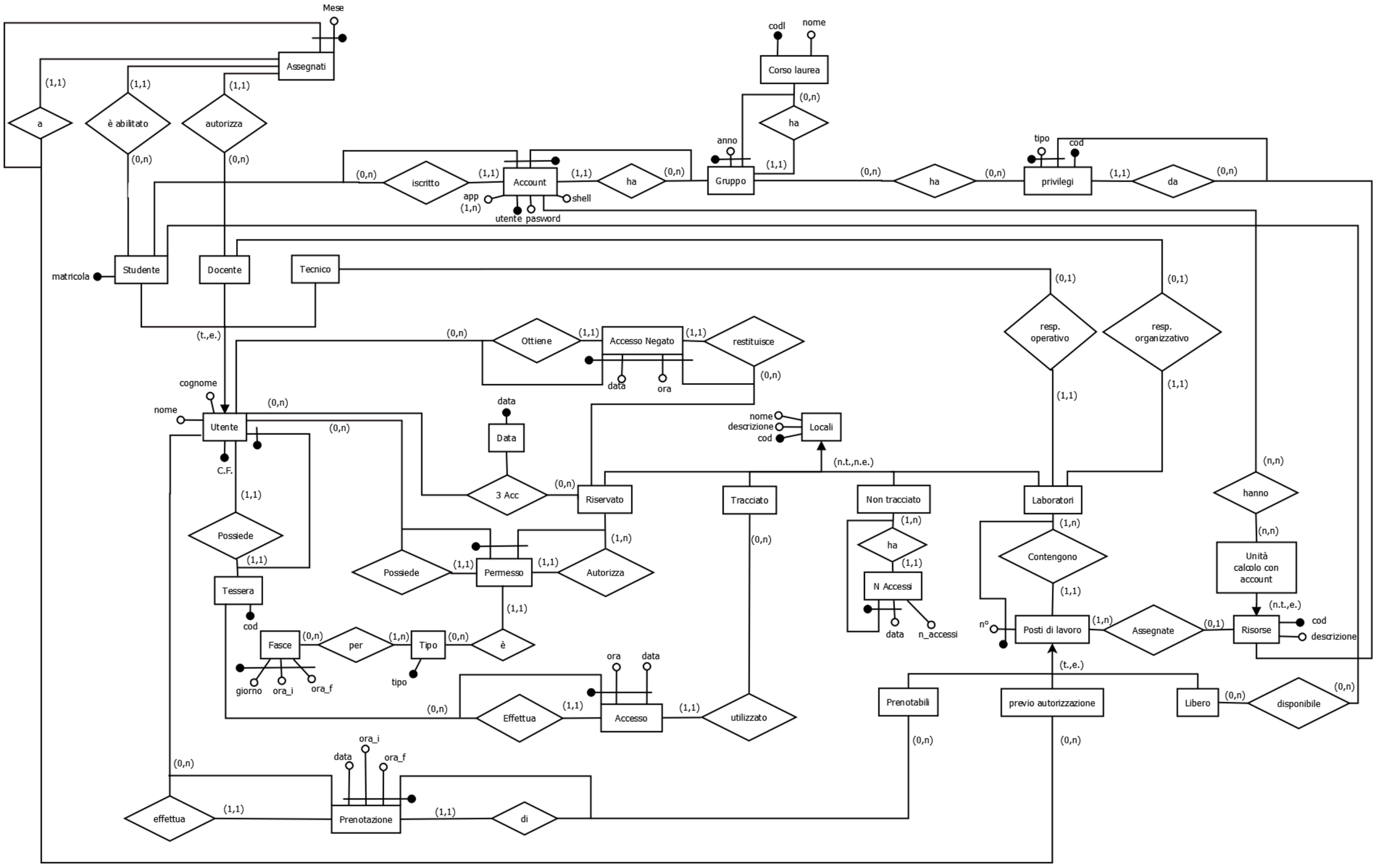


I privilegi sono identificati da un codice; la chiave alternativa è composta dal tipo di privilegio e dal codice risorsa a cui è associato. Un privilegio può dare accesso ad una singola risorsa, ma la stessa risorsa può essere legata a privilegi diversi. Un gruppo, identificato da anno e corso di laurea, può avere più privilegi.

Gli accessi negati sono gestiti attraverso due entità: la prima tiene traccia di tutti gli accessi rifiutati dal sistema, la seconda evidenzia solo gli utenti che giornalmente tentano, senza successo, 3 o più accessi allo stesso locale.



Modello E/R

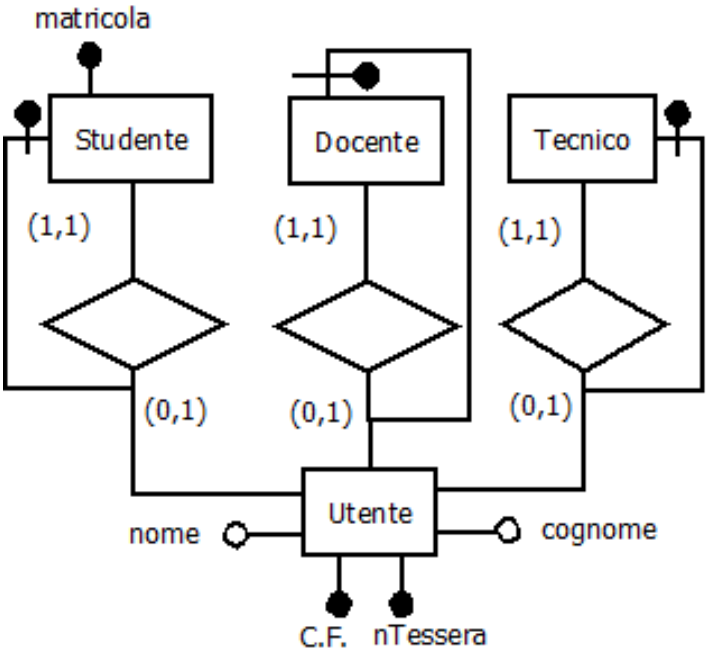

Mese
Assegnati
(1,1)
a
(1,1)
è abilitato
(1,1)
autorizza
(0,n)
(0,n)
codl
nome
Corso laurea
(0,n)
ha
(1,1)
anno
Gruppo
(0,n)
iscritto
(1,1)
Account
(1,1)
ha
(0,n)
app
(1,n)
shell
utente
pasword
(0,n)
ha
(0,n)
tipo
cod
privilegi
(1,1)
da
(0,n)
matricola
Studente
Docente
Tecnico
(t.,e.)
(0,1)
resp. operativo
(0,1)
resp. organizzativo
(0,n)
Ottiene
(1,1)
Accesso Negato
(1,1)
restituisce
(0,n)
data
ora
cognome
nome
Utente
C.F.
(0,n)
(0,n)
data
Data
3 Acc
(0,n)
Riservato
nome
descrizione
cod
Locali
(n.t.,n.e.)
Tracciato
(0,n)
Non tracciato
(1,n)
ha
(1,1)
N Accessi
data
n_accessi
(1,1)
(1,1)
Laboratori
(n,n)
hanno
(n,n)
Unità calcolo con account
(n.t.,e.)
cod
descrizione
Risorse
(0,1)
(1,n)
Assegnate
n°
Posti di lavoro
(1,n)
Contengono
(1,1)
(1,1)
Possiede
(1,1)
Tessera
cod
(1,1)
Possiede
(1,1)
Permesso
(1,1)
Autorizza
(1,n)
(1,1)
Fasce
(0,n)
per
(1,n)
Tipo
(0,n)
è
giorno
ora_i
ora_f
tipo
ora
data
(0,n)
Effettua
(1,1)
Accesso
(1,1)
utilizzato
(t.,e.)
Prenotabili
previo autorizzazione
Libero
(0,n)
disponibile
(0,n)
(0,n)
(0,n)
ora_i
data
ora_f
(0,n)
effettua
(1,1)
Prenotazione
(1,1)
di

Progetto logico

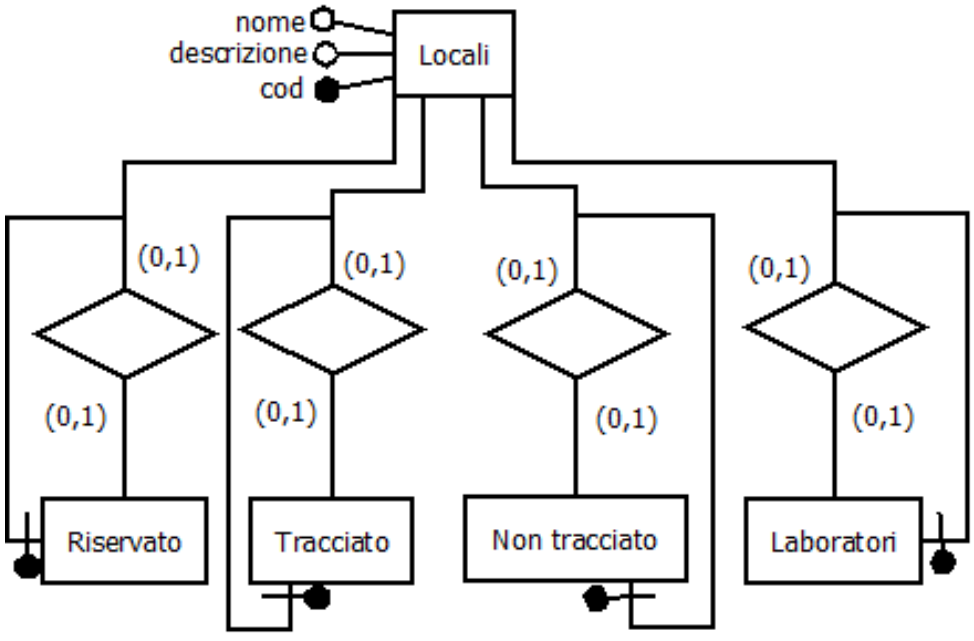
ELIMINAZIONE DELLE GERARCHIE ISA

Per tutte le gerarchie abbiamo deciso di optare per il mantenimento delle entità con associazioni.

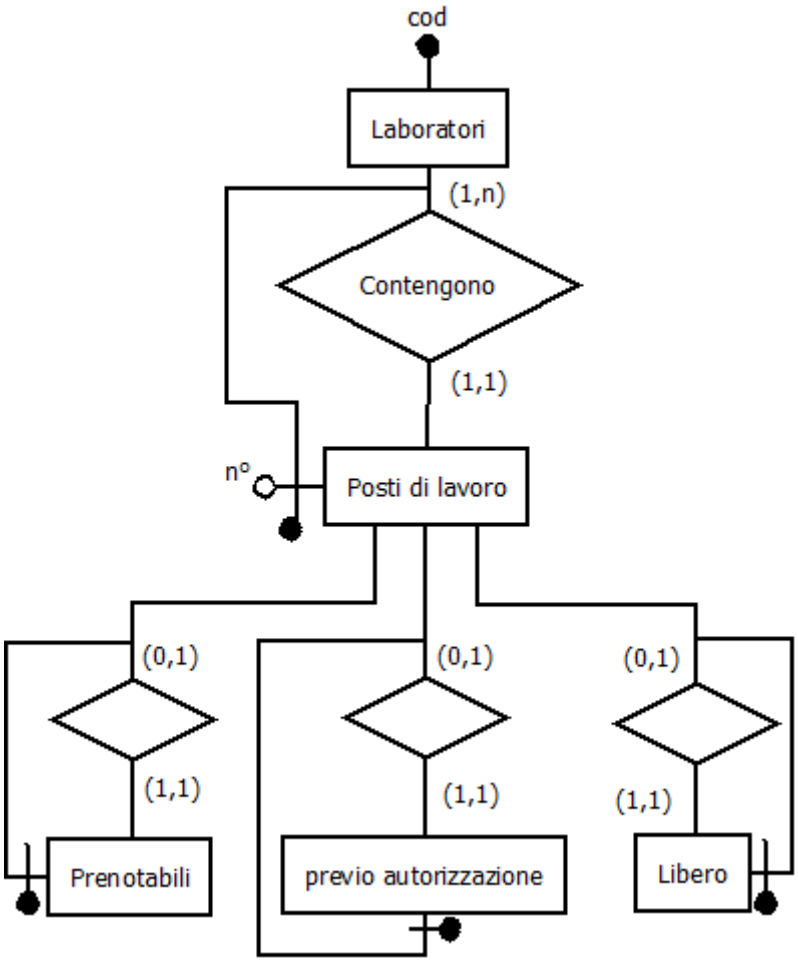
La gerarchia di utente è di tipo totale ed esclusiva. Abbiamo deciso di mantenere tutte le entità per sfruttare i vincoli espressi nello schema ER senza doverli gestire attraverso trigger o software.



La gerarchia di locali è di tipo non totale e non esclusiva. Il collasso verso il basso quindi non è possibile, e abbiamo deciso di mantenere anche qui tutte le entità per lo stesso motivo.



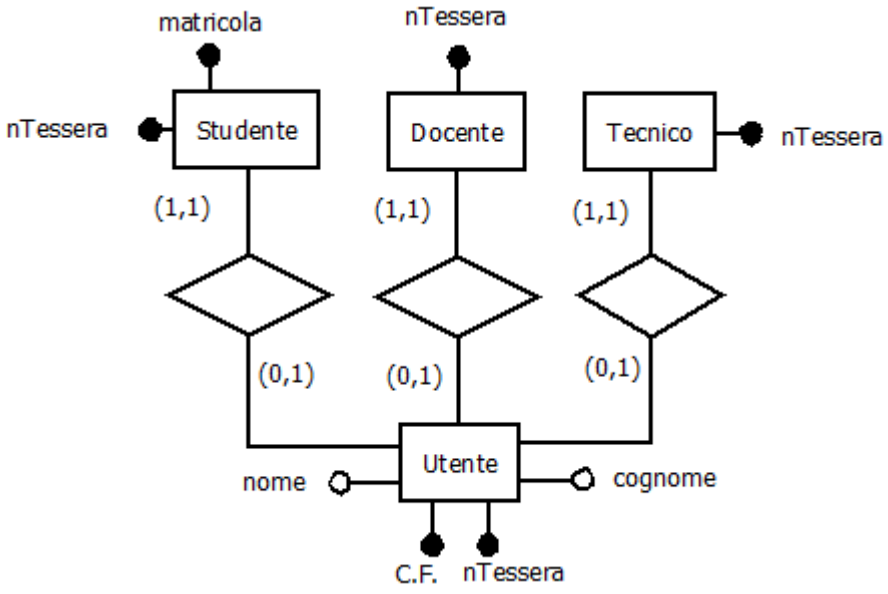
La gerarchia di posti di lavoro è totale ed esclusiva.



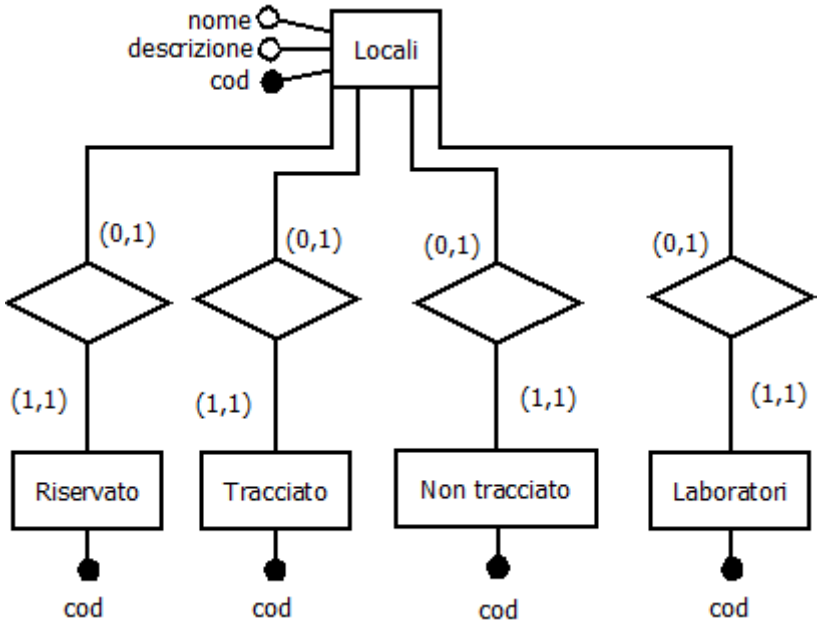
SELEZIONE DELLE CHIAVI PRIMARIE, ELIMINAZIONE DELLE IDENTIFICAZIONI ESTERNE

Partiamo dalle vecchie gerarchie ISA che diventano:

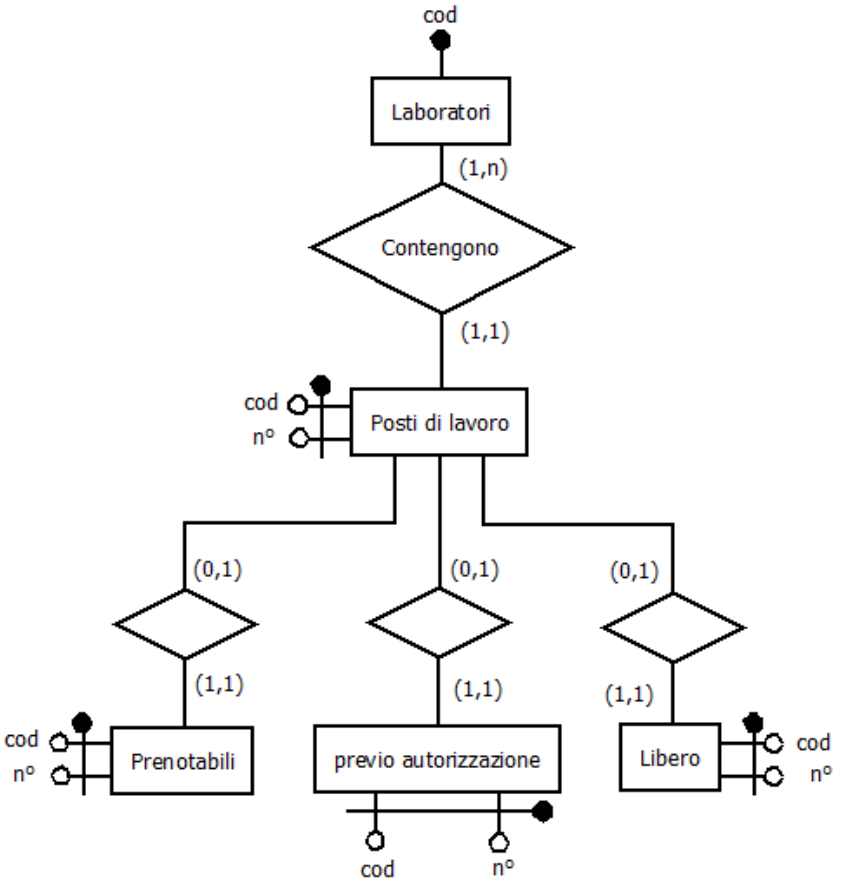
La vecchia gerarchia utente diventa:

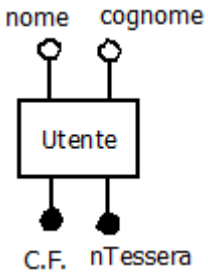


La vecchia gerarchia locali diventa:

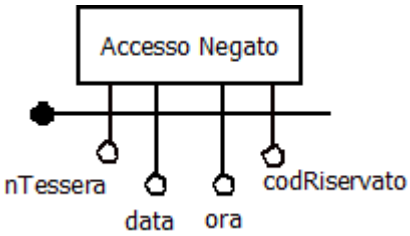


La vecchia gerarchia di posti di lavoro diventa:

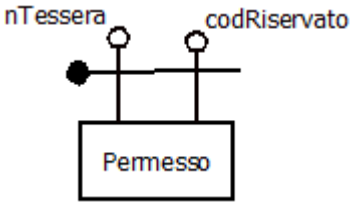




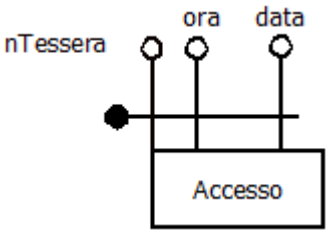
L'entità Utente è identificata dal numero di tessera. Come chiave alternativa abbiamo deciso di usare CF. Abbiamo adottato questa scelta in quanto nel contesto universitario è utilizzata l'identificazione mediante la tessera.



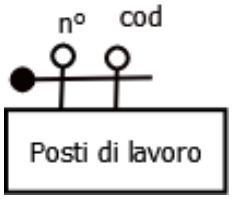
Accesso negato ha due identificatori esterni, da Utente e da Riservato.



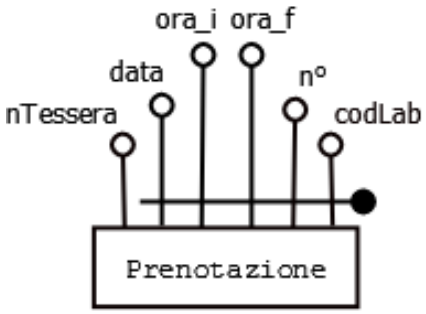
La chiave di Permesso è composta dalla chiave di Utente e la chiave di Riservato, tramite identificatori esterni.



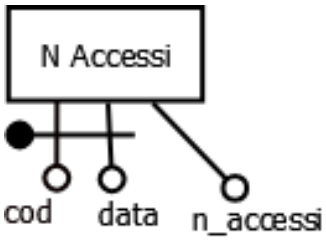
Accesso ha un identificatore esterno da Tessera.



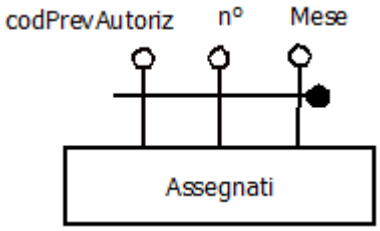
Posti di lavoro ha un identificatore esterno da Laboratori, la chiave quindi è composta dal numero del posto e dal codice del laboratorio.



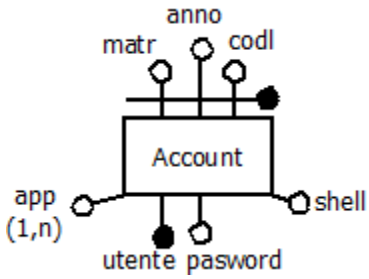
Prenotazione ha anch'esso due identificatori esterni. Fa riferimento a Utente, con chiave numero tessera, e a Previo autorizzazione, che a sua volta si riferisce a Posti di lavoro e Laboratori.



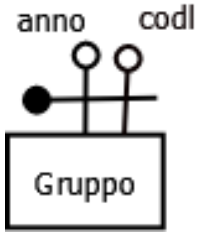
N Accessi si riferisce a Non tracciato, quindi la chiave è codice del locale unito alla data.



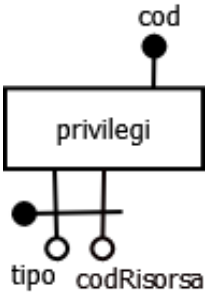
Assegnati ha un identificatore esterno. Prende numero e codice locale da quei posti che hanno bisogno di un'autorizzazione.



In account abbiamo due identificatori esterni:
-matricola da studente,
-anno e codice corso di laurea da gruppo



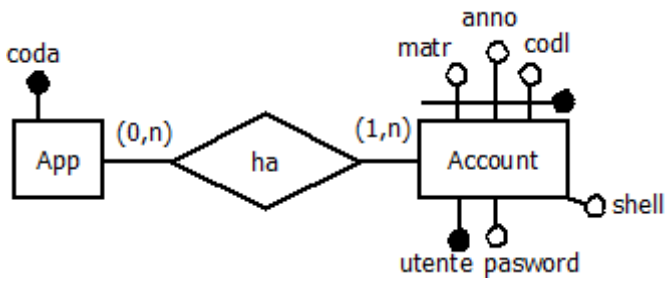
Gruppo fa riferimento a Corso di laurea, e come chiave ha la coppia di codice corso di laurea e anno.



Privilegi ha un identificatore esterno da Risorse. La sua chiave è la coppia codice risorsa e tipo di privilegio.

TRASFORMAZIONE DI ATTRIBUTI COMPOSTI O MULTIPLI

Solo nell'entità Account abbiamo un attributo multiplo, semplificando diventa:



TRADUZIONE DI ENTITA' E ASSOCIAZIONI IN SCHEMI DI RELAZIONI

Utente (nTessera, CF, nome, cognome)
AK: ***CF***

Studente (matr, nTessera)
FK: nTessera references Utente
AK: ***nTessera***

Docente (nTessera)
FK: nTessera references Utente

Tecnico (nTessera)
FK: nTessera references Utente

Locali (cod, descrizione, nome)

Riservato (cod)
FK: cod references Locali

Tracciato (cod)
FK: cod references Locali

NonTracciato (cod)
FK: cod references Locali

Laboratori (cod, nTessera as operativo, nTessera as organizzativo)
FK: operativo references Tecnico
FK: organizzativo references Docente
FK: cod references Locali
AK: ***operativo***
AK: ***organizzativo***

Tipo (tipo)

Fasce (giorno, orai, oraf)

Per (tipo, giorno, orai, oraf)
FK: tipo references Tipo
FK: giorno, orai, oraf references Fasce

Permesso (cod, nTessera, tipo)
FK: tipo references Tipo
FK: cod references Riservato
FK: nTessera references Utente

AccessoNegato (nTessera, data, ora, cod)
FK: nTessera references Utente
FK: cod references Riservato

3AccNegati (nTessera,data,cod)
FK: nTessera references Utente
FK: cod references Riservato

nAccessi (cod, data, n)
FK: cod references NonTracciato

Accesso (ntessera, data, ora, cod)
FK: cod references Tracciato
FK: ntessera reference Utente

PostiDiLavoro (n, cod)
FK: cod references Laboratori

Prenotabili (n, cod)
FK: n, cod references PostiDiLavoro

conAutorizzazione (n, cod)
FK: n, cod references PostiDiLavoro

Libero (n, cod)
FK: n, cod references PostiDiLavoro

Disponibile (n, cod, matr)
FK: n, cod references Libero
FK: matr references Studente

Prenotazione (n, cod, data, orai, oraf, nTessera)
FK: n, cod references Prenotabili
FK: nTessera references Utente

Assegnati (mese, n, cod, matr, nTessera)
FK: n, cod references conAutorizzazione
FK: matr references Stedente
FK: nTessera references Docente

Risorse (cod, descr)

UnitaCalcolo (cod)
FK: cod references Risorse

Privilegi (codp, cod, tipo)
FK: cod references Risorse
AK: ***tipo, cod***

CorsoLaurea (codl, nome)

Gruppo (codl, anno)
FK: codl references CorsoLaurea

Ha (codl, anno, codp)
FK: codl, anno references Gruppo
FK: codp references Privilegi

Account (matr, codl, anno, utente, password, shell)
FK: matr references Studente
FK: codl, anno references Gruppo
AK: ***matr, codl, anno***

App (coda, descrizione)

Utilizzate (utente, coda)
FK: utente references Account
FK: coda references App

Hanno (cod, utente)
FK: cod references UnitaCalcolo
FK: utente references Account

SQL

```
CREATE TABLE Utente (
        nTessera SERIAL,
        cf CHAR(16) UNIQUE NOT NULL,
        nome CHAR(25),
        cognome CHAR (25),
PRIMARY KEY (nTessera)
);

CREATE TABLE Studente (
        matr INTEGER,
        nTessera INTEGER UNIQUE NOT NULL,
PRIMARY KEY (matr),
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE
);

CREATE TABLE Docente (
        nTessera INTEGER,
PRIMARY KEY (nTessera),
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE
);

CREATE TABLE Tecnico (
nTessera INTEGER,
PRIMARY KEY (nTessera),
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE
);

CREATE TABLE Locali (
        cod CHAR(5),
        nome VARCHAR(30),
        descrizione VARCHAR(300),
PRIMARY KEY(cod)
);

CREATE TABLE Riservato (
        cod CHAR(5),
PRIMARY KEY(cod),
FOREIGN KEY(cod) REFERENCES Locali (cod) ON DELETE CASCADE
);
```

```
CREATE TABLE Tracciato (
        cod CHAR(5),
PRIMARY KEY(cod),
FOREIGN KEY(cod) REFERENCES Locali (cod) ON DELETE CASCADE
);


CREATE TABLE NonTracciato (
        cod CHAR(5),
PRIMARY KEY(cod),
FOREIGN KEY(cod) REFERENCES Locali (cod) ON DELETE CASCADE
);

CREATE TABLE Laboratori (
        cod CHAR(5),
        operativo INTEGER UNIQUE NOT NULL,
        organizzativo INTEGER UNIQUE NOT NULL,
PRIMARY KEY (cod),
FOREIGN KEY (cod) REFERENCES Locali (cod) ON DELETE CASCADE,
FOREIGN KEY (operativo) REFERENCES Tecnico(nTessera) ON DELETE NO ACTION,
FOREIGN KEY (organizzativo) REFERENCES Docente (nTessera) ON DELETE NO ACTION
);

CREATE TABLE Tipo (
        tipo CHAR(4),
PRIMARY KEY (tipo)
);

CREATE TABLE Fasce (
        giorno SMALLINT CHECK (giorno>=1 and giorno<=7),
        orai TIME(0),
        oraf TIME(0) CHECK (orai<oraf),
PRIMARY KEY (giorno, orai, oraf)
);

CREATE TABLE Per (
        tipo CHAR(4),
giorno SMALLINT,
        orai TIME(0),
        oraf TIME(0),
PRIMARY KEY (tipo, giorno, orai, oraf),
FOREIGN KEY(tipo) REFERENCES Tipo (tipo) ON DELETE CASCADE,
FOREIGN KEY (giorno, orai, oraf) REFERENCES Fasce (giorno, orai, oraf) ON DELETE CASCADE
);
```

```
CREATE TABLE Permesso (
        cod CHAR(5),
        nTessera INTEGER,
        tipo CHAR(4),
PRIMARY KEY (cod, nTessera),
FOREIGN KEY (cod) REFERENCES Riservato (cod) ON DELETE CASCADE,
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE,
FOREIGN KEY (tipo) REFERENCES Tipo (tipo) ON DELETE CASCADE
);

CREATE TABLE AccessoNegato (
        nTessera INTEGER,
        data DATE,
        ora TIME(4),
        cod CHAR(5),
PRIMARY KEY (nTessera, data, ora, cod),
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE,
FOREIGN KEY (cod) REFERENCES Riservato (cod) ON DELETE CASCADE
);

CREATE TABLE TreAccNegati (
        nTessera INTEGER,
        data DATE,
        cod CHAR(5),
PRIMARY KEY (nTessera, data, cod),
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE,
FOREIGN KEY (cod) REFERENCES Riservato (cod) ON DELETE CASCADE
);

CREATE TABLE nAccessi (
        cod CHAR(4),
        data DATE,
        n INTEGER,
PRIMARY KEY (cod, data),
FOREIGN KEY (cod) REFERENCES NonTracciato (cod) ON DELETE CASCADE
);

CREATE TABLE Accesso (
        ntessera INTEGER,
        data DATE,
        ora TIME(4),
        cod CHAR(5),
PRIMARY KEY (ntessera, ora, data),
FOREIGN KEY (ntessera) REFERENCES Utente (ntessera) ON DELETE NO ACTION,
FOREIGN KEY (cod) REFERENCES Tracciato (cod) ON DELETE CASCADE
);
```

```
CREATE TABLE PostiDiLavoro (
        n SMALLINT,
        cod CHAR(4),
PRIMARY KEY (n, cod),
FOREIGN KEY (cod) REFERENCES Laboratori (cod) ON DELETE CASCADE
);

CREATE TABLE Prenotabili (
        n SMALLINT,
        cod CHAR(4),
PRIMARY KEY (n, cod),
FOREIGN KEY (cod, n) REFERENCES PostiDiLavoro (cod, n) ON DELETE CASCADE
);

CREATE TABLE conAutorizzazione (
        n SMALLINT,
        cod CHAR(4),
PRIMARY KEY (n, cod),
FOREIGN KEY (cod, n) REFERENCES PostiDiLavoro (cod, n) ON DELETE CASCADE
);

CREATE TABLE Libero (
        n SMALLINT,
        cod CHAR(4),
PRIMARY KEY (n, cod),
FOREIGN KEY (cod, n) REFERENCES PostiDiLavoro (cod, n) ON DELETE CASCADE
);

CREATE TABLE Disponibile (
        n SMALLINT,
        cod CHAR(4),
        matr INTEGER,
PRIMARY KEY (n, cod, matr),
FOREIGN KEY (cod, n) REFERENCES Libero (cod, n) ON DELETE CASCADE,
FOREIGN KEY (matr) REFERENCES Studente (matr) ON DELETE CASCADE
);
```

```
CREATE TABLE Prenotazione (
        n SMALLINT,
        cod CHAR(4),
        data DATE,
        orai TIME(0),
        oraf TIME(0) CHECK(orai<oraf),
        nTessera INTEGER NOT NULL,
PRIMARY KEY (n, cod, data, orai, oraf),
FOREIGN KEY (n, cod) REFERENCES Prenotabili (n, cod) ON DELETE CASCADE,
FOREIGN KEY (nTessera) REFERENCES Utente (nTessera) ON DELETE CASCADE
);

CREATE TABLE Assegnati (
        mese SMALLINT CHECK(mese>=1 and mese<=12),
        n SMALLINT,
        cod CHAR(4),
        matr INT,
        nTessera INT,
PRIMARY KEY (mese, n, cod),
FOREIGN KEY (n, cod) REFERENCES conAutorizzazione (n, cod) ON DELETE CASCADE,
FOREIGN KEY (matr) REFERENCES Studente (matr) ON DELETE CASCADE,
FOREIGN KEY (nTessera) REFERENCES Docente (nTessera) ON DELETE CASCADE
);

CREATE TABLE Risorse (
        cod CHAR(6),
        descrizione VARCHAR(30),
PRIMARY KEY (cod)
);

CREATE TABLE UnitaCalcolo (
        cod CHAR(6),
PRIMARY KEY (cod),
FOREIGN KEY (cod) REFERENCES Risorse (cod) ON DELETE CASCADE
);

CREATE TABLE Privilegi (
        codp CHAR(6),
        cod CHAR(6) NOT NULL,
        tipo CHAR (30) NOT NULL,
PRIMARY KEY (codp),
FOREIGN KEY (cod) REFERENCES Risorse (cod) ON DELETE CASCADE,
UNIQUE (cod, tipo)
);
```

```
CREATE TABLE CorsoLaurea (
        codl CHAR(4),
        nome VARCHAR(30),
PRIMARY KEY (codl)
);

CREATE TABLE Gruppo (
        codl CHAR(4),
        anno SMALLINT CHECK (anno>1700),
PRIMARY KEY (codl, anno),
FOREIGN KEY (codl) REFERENCES CorsoLaurea (codl) ON DELETE CASCADE
);

CREATE TABLE Ha (
        Codl CHAR(4),
        anno SMALLINT,
        codp CHAR(6),
PRIMARY KEY (codl, anno, codp),
FOREIGN KEY (codl, anno) REFERENCES Gruppo (codl, anno) ON DELETE CASCADE,
FOREIGN KEY (codp) REFERENCES Privilegi (codp) ON DELETE CASCADE
);

CREATE TABLE Account (
        matr INTEGER NOT NULL,
        codl CHAR(4) NOT NULL,
        anno SMALLINT NOT NULL,
        utente VARCHAR(20),
        password CHAR(128),
shell CHAR(3),
PRIMARY KEY (utente),
FOREIGN KEY (matr) REFERENCES Studente (matr) ON DELETE CASCADE,
FOREIGN KEY (codl, anno) REFERENCES Gruppo (codl, anno) ON DELETE CASCADE,
UNIQUE (matr, codl, anno)
);
CREATE TABLE App (
        coda CHAR(5),
        descrizione VARCHAR(30),
PRIMARY KEY (coda)
);

CREATE TABLE Utilizzate (
        utente VARCHAR(20),
        coda CHAR(5),
PRIMARY KEY (utente, coda),
FOREIGN KEY (utente) REFERENCES Account (utente) ON DELETE CASCADE,
FOREIGN KEY (coda) REFERENCES App (coda) ON DELETE CASCADE
);
```

```
CREATE TABLE Hanno (
        utente VARCHAR(20),
        cod CHAR(6),
PRIMARY KEY (utente, cod),
FOREIGN KEY (utente) REFERENCES Account (utente) ON DELETE CASCADE,
FOREIGN KEY (cod) REFERENCES UnitaCalcolo (cod) ON DELETE CASCADE
);
```

Trigger

Abbiamo pensato di evidenziare tre accessi negati per un locale inserendo una nuova tupla all'interno della tabella TreAccessiNegati in modo automatico. Così facendo riusciamo a tenere traccia di tutti gli accessi rifiutati.

```
CREATE FUNCTION check_negati() RETURNS trigger AS $check_negati$
 BEGIN
IF (SELECT COUNT(*) FROM accessonegato WHERE cod=new.cod and ntessera=new.ntessera and
data=new.data) =3 THEN
INSERT INTO "treaccnegati" (ntessera,cod,data) VALUES (new.ntessera, new.cod, new.data);
END IF;
RETURN NEW;
END;
$check_negati$ LANGUAGE plpgsql;
CREATE TRIGGER GESTIONE_NEGATI
AFTER INSERT OR UPDATE ON accessonegato
FOR EACH ROW EXECUTE PROCEDURE check_negati();
```

Per quanto riguarda le gerarchie abbiamo pensato di impedire la possibilità di categorizzare un utente in modi diversi rendendo effettivo il vincolo di esclusività della gerarchia.

```
CREATE FUNCTION check_categoria() RETURNS trigger AS $check_categoria$
 BEGIN
IF (SELECT COUNT(*) FROM ((SELECT ntessera FROM studente) UNION (SELECT ntessera FROM docente)
UNION (SELECT ntessera FROM tecnico)) AS tot WHERE tot.ntessera=new.ntessera ) >0 THEN
RAISE EXCEPTION 'Utente già categorizzato';
END IF;
RETURN NEW;
END;
$check_categoria$ LANGUAGE plpgsql;

CREATE TRIGGER GESTIONE_studente
BEFORE INSERT OR UPDATE ON studente
FOR EACH ROW EXECUTE PROCEDURE check_categoria();

CREATE TRIGGER GESTIONE_tecnico
BEFORE INSERT OR UPDATE ON tecnico
FOR EACH ROW EXECUTE PROCEDURE check_categoria();

CREATE TRIGGER GESTIONE_docente
BEFORE INSERT OR UPDATE ON docente
FOR EACH ROW EXECUTE PROCEDURE check_categoria();
```

Allo stesso modo abbiamo fatto per I posti dei laboratori.

```
CREATE FUNCTION check_tipo_posto() RETURNS trigger AS $check_tipo_posto$
 BEGIN
IF (SELECT COUNT(*) FROM ((SELECT n, cod FROM prenotabili) UNION (SELECT n, cod FROM
conautorizzazione) UNION (SELECT n, cod FROM libero)) AS tot WHERE tot.n=new.n and tot.cod=new.cod )
>0 THEN
RAISE EXCEPTION 'Posto già categorizzato';
END IF;
RETURN NEW;
END;
$check_tipo_posto$ LANGUAGE plpgsql;

CREATE TRIGGER GESTIONE_prenotabili
BEFORE INSERT OR UPDATE ON prenotabili
FOR EACH ROW EXECUTE PROCEDURE check_tipo_posto();

CREATE TRIGGER GESTIONE_conautorizzazione
BEFORE INSERT OR UPDATE ON conautorizzazione
FOR EACH ROW EXECUTE PROCEDURE check_tipo_posto();

CREATE TRIGGER GESTIONE_libero
BEFORE INSERT OR UPDATE ON libero
FOR EACH ROW EXECUTE PROCEDURE check_tipo_posto();
```

Il prossimo trigger impedisce ad uno studente di prenotare più posti nella stessa fascia oraria:

```
CREATE FUNCTION check_prenotazione() RETURNS trigger AS $check_prenotazione$
 BEGIN
IF ((SELECT COUNT(*) FROM Studente WHERE ntessera=new.ntessera)=1 AND (SELECT COUNT(*) FROM
Prenotazione WHERE ntessera=new.ntessera AND data=new.data AND oraf=new.oraf AND
orai=new.orai)>=1) THEN
RAISE EXCEPTION 'Uno studente non può prenotare più posti nella stessa fascia.';
END IF;
RETURN NEW;
END;
$check_prenotazione$ LANGUAGE plpgsql;

CREATE TRIGGER GESTIONE_prenotazione
BEFORE INSERT OR UPDATE ON prenotazione
FOR EACH ROW EXECUTE PROCEDURE check_prenotazione ();
```

Popolamento database

```
INSERT INTO "utente" (CF, nome, cognome)
VALUES
('TRRMRC92C14H223Q','Marco','Torreggiani'),
('SPNDVD93T11F240N','Davide','Sapienza'),
('VZZLRT91B08B819X','Alberto','Vezzani'),
('PSDFKB2132SDF5GX','Pippo','Piri'),
('RCCMRT90K256Z85W','Riccardo','Martoglia'),
('LNCMRA80B12C218R','Mauro','Leoncini'),
('RSSNTN80F21H223Q','Antonia','Rossi'),
('NDRRSL80B12C218R','Rosalba','Andreotti');

INSERT INTO "studente" (nTessera,matr)
VALUES
(1,70297),
(2,70649),
(3,73272),
(4,55555);

INSERT INTO "docente"
VALUES
(5),
(6),
(7);

INSERT INTO "tecnico"
VALUES
(8);

INSERT INTO "locali"(cod, nome, descrizione)
VALUES
('L1','Lab Base','Laboratorio di informatica'),
('L2','Biblioteca','BSI'),
('L3','Aula Studio','Riservata a una categoria di studenti'),
('L4','BOX','Box studio'),
('L5','Sgabuzzino','contiene detersivi');

INSERT INTO "riservato"
VALUES ('L3');

INSERT INTO "tracciato"
VALUES ('L4');

INSERT INTO "nontracciato"
VALUES ('L2');
```

```
INSERT INTO "laboratori"
VALUES ('L1',8,5);

INSERT INTO "tipo"
VALUES ('T001'),('T002'),('T003'),('T004');

INSERT INTO "fasce" (giorno,orai,oraf)
VALUES
(1,'09:00','11:00'),
(1,'12:00','13:00'),
(2,'11:00','13:00'),
(3,'11:00','13:00'),
(4,'11:00','13:00'),
(5,'14:00','15:00'),
(1,'18:00','19:00'),
(2,'9:00','13:00'),
(3,'16:00','18:00'),
(4,'9:00','13:00'),
(5,'14:00','15:30');

INSERT INTO "per" (tipo,giorno,orai,oraf)
VALUES
('T001',1,'09:00','11:00'),
('T001',1,'12:00','13:00'),
('T003',2,'11:00','13:00'),
('T002',3,'11:00','13:00'),
('T004',4,'11:00','13:00');

INSERT INTO "permesso" (cod,ntessera,tipo)
VALUES
('L3',1,'T001'),
('L3',2,'T003'),
('L3',3,'T002');

INSERT INTO "postidilavoro" (n,cod)
VALUES
(1,'L1'),(2,'L1'),(3,'L1'),(4,'L1'),(5,'L1'),(6,'L1'),(7,'L1'),(8,'L1'),(9,'L1'),(10,'L1'),(11,'L1'),(12,'L1'),(13,'L1'),
(14,'L1'),(15,'L1'),(16,'L1'),(17,'L1'),(18,'L1'),(19,'L1'),(20,'L1');

INSERT INTO "prenotabili" (n,cod)
VALUES
(1,'L1'),(2,'L1'),(3,'L1'),(4,'L1'),(5,'L1');
```

```
INSERT INTO "conautorizzazione" (n,cod)
VALUES
(15,'L1'),
(16,'L1'),
(17,'L1'),
(18,'L1'),
(19,'L1'),
(20,'L1');

INSERT INTO "libero" (n,cod)
VALUES
(6,'L1'),(7,'L1'),(8,'L1'),(9,'L1'),(10,'L1'),(11,'L1'),(12,'L1'),(13,'L1'),(14,'L1');

INSERT INTO "risorse" (cod, descrizione)
VALUES
('R00001','stampante'),
('R00002','stampante'),
('R00003','stampante'),
('R00004','proiettore'),
('R00005','computer'),
('R00006','computer'),
('R00007','computer'),
('R00008','computer'),
('R00009','computer'),
('R00010','computer'),
('R00011','condizionatore');

INSERT INTO "unitacalcolo"
VALUES
('R00005'),('R00006'),('R00007'),('R00008'),('R00009'),('R00010');

INSERT INTO "privilegi" (codp,cod,tipo)
VALUES
('PRIV01','R00001','stampaFree'),
('PRIV02','R00005','admin'),
('PRIV03','R00004','telecomando');

INSERT INTO "corsolaurea" (codl,nome)
VALUES
('INF1','Informatica'),
('MAT1','Matematica'),
('MED1','Medicina');
```

```
INSERT INTO "gruppo" (codl,anno)
VALUES
('INF1',2010),
('MAT1',2010),
('MED1',2010),
('INF1',2011),
('MAT1',2011),
('MED1',2011),
('INF1',2012),
('MAT1',2012),
('MED1',2012);

INSERT INTO "account" (matr, codl, anno, utente, password)
VALUES
(70297, 'INF1', 2010, 'torre', 'pass1'),
(70649, 'INF1', 2011, 'sappi', 'pass2'),
(73272, 'MAT1', 2011, 'albi', 'pass3');

INSERT INTO "app" (coda,descrizione)
VALUES
('APP01','Matlab'),
('APP02','Postgres'),
('APP03','Netlogo'),
('APP04','g++');
```

Query

- *Visualizzare, per un certo laboratorio, il piano di occupazione relativo ad un certo periodo di tempo (quali, ad esempio, un giorno, una settimana, un mese) riportandone i posti disponibili, assegnati e prenotati.*

Nell'applicazione abbiamo pensato di gestire la visualizzazione per giorno e di mettere a disposizione dell'utente la possibilità di avanzare nella ricerca giorno per giorno, in modo da poter accedere alle informazioni sul periodo desiderato.

Per fare ciò dobbiamo fare le interrogazioni su due tabelle (Assegnati e Prenotati), tralasciando i posti disponibili che solo liberamente fruibili.

Supponendo di ricevere in input dall'utente una data, in formato gg/mm/aaaa, e il codice del laboratorio che vuole monitorare, eseguiamo:

```
SELECT n
FROM assegnati
WHERE cod='codice' AND mese= mm
ORDER BY n
```

```
SELECT n
FROM prenotazione
WHERE cod='codice' AND data = 'gg/mm/aaaa'
ORDER BY n
```

Nell'applicazione la seconda query viene eseguita per fasce orarie di un'ora in modo da poter rappresentare l'occupazione dell'intera giornata in un layout tabellare.

```
for (i=8; i<19; i++){
        SELECT n
        FROM prenotazione
        WHERE cod='codice' AND
                data = 'gg/mm/aaaa' AND
                orai <= 'i:00' AND
                oraf >= '(i+1):00'
        ORDER BY n
}
```

- *Prenotare i posti di lavoro in un laboratorio didattico. Generalmente una prenotazione, soprattutto se effettuata da un docente, avviene specificando le sequenti informazioni: le ore del giorno, i giorni della settimana e un periodo.*

Per effettuare questa operazione supponiamo di ricevere in input: il numero di posti in un laboratorio (n, cod), una data di inizio delle prenotazioni (gg/mm/aaaa), una fascia oraria (orai, oraf) e il numero delle settimane per le quali si intende estendere la prenotazione (set). A questo punto si tratta soltanto di effettuare un inserimento nella tabella Prenotazione per ogni posto e per ogni giorno selezionato.

```
for (i=0; i<set; i++){
        INSERT INTO "prenotazione" (n, cod, data, orai, oraf, ntessera)
        VALUES (n, 'cod', '(gg+i*7)/mm/aaaa', 'orai', 'oraf', ntessera);
}
```

- *Dato un locale elencare tutti gli utenti che hanno il permesso di accedere al locale con i relativi giorni e intervalli temporali di accesso.*

```
SELECT p.ntessera, pr.giorno, pr.orai, pr.oraf
FROM permesso p, tipo t, per pr WHERE
p.cod= 'cod' AND p.tipo=t.tipo AND pr.tipo=t.tipo
ORDER BY p.ntessera
```

- *Riportare, per un dato laboratorio, statistiche riguardanti la sua occupazione in un certo periodo di tempo (un giorno, una settimana, un mese).*

Per quanto riguarda le statistiche abbiamo deciso di far scegliere all'utente due giorni su cui basare la ricerca (datai, dataf). Dopodiché si conteggiano quante prenotazioni sono presenti e quanti posti assegnati ci sono nel periodo specificato.

```
SELECT COUNT(*)
FROM prenotazione
WHERE cod='cod' and data >= 'datai' AND
        data <= 'dataf'
```

```
SELECT COUNT(*)
FROM assegnati
WHERE cod='cod' AND
        mese >= mesei AND
        mese <= mesef
```

Funzioni Aggiuntive

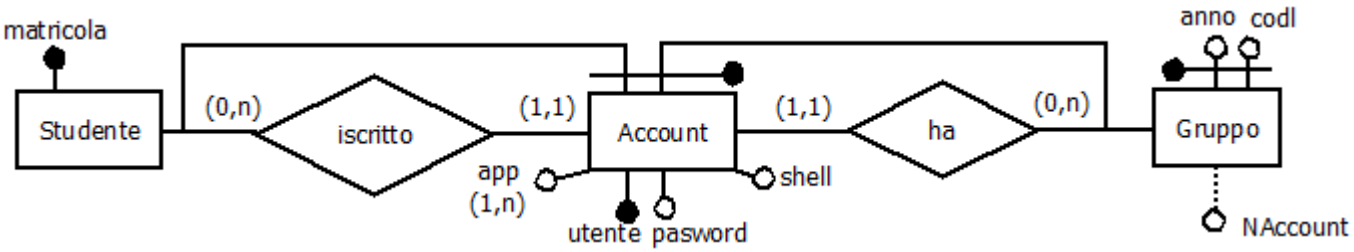
- Nell'applicazione abbiamo deciso di implementare l'inserimento di un nuovo utente, specificando la categoria di appartenenza.
  Spezziamo quindi l'operazione in due query di inserimento: una in Utente e una nella categoria di appartenenza (Studente, Tecnico, Docente)

- Nel caso in cui si scelga di inserire uno studente l'applicazione chiede all'utente di creare anche un account e di associarlo ad un gruppo (per es. INFORMATICA 2012), oltre che ad assegnare una matricola.

- Rendiamo disponibile dall'applicazione anche la creazione di un corso di laurea, di un gruppo e la visualizzazione di tutti gli utenti archiviati.

Studio dato derivato

Abbiamo effettuato uno studio di dato derivato sul numero di account per gruppo. Abbiamo utilizzato valori esemplificativi di volume dei dati e operazioni.
È utile visualizzare tutte le informazioni di un gruppo, compreso il numero di account per avere una stima sul numero di iscrizioni ad un certo cdl nel dato anno.



OPERAZIONE 1
Inserimento di un nuovo account.

OPERAZIONE 2
Visualizzazione di tutti i campi di gruppo e il numero di account.

OPER.	**TIPO**	**FREQ.**
Oper. 1	L	80/anno
Oper. 2	L	4/anno

CONCETTO	**TIPO**	**VOL.**
Gruppo	E	5
Account	E	80

Studio con dato derivato:

OP1	Account	S	1
	Ha	L	1
	Gruppo	L	1
	Gruppo	S	1
OP2	Gruppo	L	1

Risultato:

OP1 2*L + S = 160 + 2*80 = 320/anno
OP2 L = 4/anno

Tot: 324/anno

Studio senza dato derivato:

OP1	Account	S	1
OP2	Gruppo	L	1
	Ha	L	16
	Account	L	16

Risultato:

OP1 2 = 2*80 = 160/anno
OP2 33*L = 33*4 = 132/anno

Tot: 292/anno

Conclusione: Non conviene tenere il dato derivato

Applicazione JAVA

Abbiamo pensato di fare un applicazione in JAVA dotata di una semplice interfaccia grafica che aiuti l'utente a comunicare con il database.
Siamo consapevoli che l'applicazione presenta solo poche funzionalità rispetto alle tante possibili, ma abbiamo optato per una scelta di questo tipo, in quanto in linea con le richieste del progetto. Siamo comunque riusciti a implementare le operazione di base anche con un interfaccia grafica più che discreta.

Per effettuare la connessione tra applicazione e DBMS Postgres le credenziali sono da settare nella classe DataBase del progetto.