Relazione per il corso di

Basi di Dati

Agriturismo

Alessandro Rebosio
Filippo Ricciotti

18 settembre 2025

Anno Accademico 2024-2025

Indice

Capitolo 1

Analisi dei requisiti

1.1 Intervista

L'agriturismo intende dotarsi di una piattaforma digitale che razionalizzi le attività quotidiane e migliori l'esperienza dei clienti, integrando in un unico ambiente la gestione del personale, la vendita di prodotti e la promozione di eventi. Il titolare desidera uno strumento accessibile via web, utilizzabile da utenti registrati e dal personale autorizzato, in grado di offrire una visione chiara e centralizzata delle informazioni operative, riducendo errori e tempi di coordinamento.

Il cuore dell'applicativo è costituito da un catalogo di prodotti e da un calendario di eventi, visibili ai visitatori e consultabili dagli utenti registrati. I prodotti, identificati da un codice univoco, sono descritti da un nome e da un prezzo, con la garanzia che i valori economici rimangano sempre positivi. Gli eventi, invece, sono presentati con titolo, descrizione, data di svolgimento e un numero di posti disponibili; la loro pubblicazione è effettuata da dipendenti autorizzati, così da mantenere controllo e coerenza dell'offerta.

Gli utenti potranno creare un account fornendo un nome utente, un indirizzo email e una password; ogni profilo sarà associato a una persona identificata tramite codice fiscale, così da assicurare un'anagrafica pulita e non ridondante. Una volta autenticati, gli utenti potranno consultare il catalogo, comporre ordini di acquisto di prodotti e completarne la registrazione: ogni ordine sarà tracciato con data e ora, e conterrà le righe di dettaglio con quantità e prezzo unitario, in modo da consentire il calcolo del totale e la successiva rendicontazione. Gli acquisti rimarranno associati in modo permanente all'account dell'utente, così da poterli rivedere e analizzare nel tempo.

Per la dimensione esperienziale dell'agriturismo, la piattaforma offrirà una sezione dedicata agli eventi: gli utenti interessati potranno iscriversi indicando il numero di partecipanti; il sistema dovrà garantire che le prenotazioni non superino i posti disponibili e registrerà automaticamente data e ora di ciascuna iscrizione. In questo modo, il titolare potrà monitorare in tempo reale l'andamento delle adesioni e prevedere l'affluenza, ottimizzando l'organizzazione delle serate e delle attività tematiche.

La gestione del personale rappresenta un altro pilastro del sistema. Ciascun dipendente sarà un utente abilitato a funzioni specifiche e caratterizzato da un ruolo (ad esempio sala, cucina, reception), con la possibilità di tracciarne lo storico delle variazioni nel tempo. La pianificazione dei turni avverrà attraverso la definizione di modelli di turno (per giorno della settimana, con orari di inizio e fine) e la loro assegnazione a calendario per una data specifica. Ogni assegnazione prevede uno stato — programmato, completato o assente — così da fotografare l'effettiva presenza; inoltre, il sistema eviterà conflit-

ti, impedendo che uno stesso dipendente risulti assegnato a più turni nella medesima giornata.

Dal punto di vista direzionale, il titolare richiede una reportistica essenziale ma affidabile: l'andamento delle vendite per periodo, la partecipazione agli eventi e un quadro della presenza/assenza del personale sui turni. La piattaforma dovrà salvaguardare la sicurezza dei dati, conservando le password in forma sicura e applicando vincoli di integrità su prezzi e quantità; le operazioni frequenti — come consultare il catalogo, registrare un ordine o iscriversi a un evento — dovranno risultare rapide e semplici, privilegiando chiarezza e immediatezza d'uso.

1.2 Estrazione dei concetti principali

L'agriturismo intende realizzare una piattaforma digitale che unisca in un unico ecosistema la vendita di prodotti, la promozione e gestione degli eventi e l'organizzazione del personale. Il sistema sarà accessibile via web agli utenti registrati e al personale autorizzato, con l'obiettivo di offrire una vista centralizzata e coerente delle attività quotidiane, riducendo errori operativi e tempi di coordinamento. Il cuore dell'applicazione è rappresentato da un **catalogo di prodotti** e da un calendario eventi: i prodotti, identificati in modo univoco (**codice**), e descritti da **nome** e **prezzo**, saranno acquistabili dagli utenti autenticati; gli eventi, caratterizzati da **titolo**, **descrizione**, **data** e **posti disponibili**, saranno visibili e prenotabili secondo regole di capienza stabilite dall'azienda.

Gli utenti potranno creare un account fornendo **nome utente**, **email** e **password**; ogni account sarà associato a una persona identificata da **codice fiscale**, in modo da mantenere un'anagrafica solida e priva di duplicati. Una volta autenticati, gli utenti potranno consultare il catalogo e comporre ordini, che verranno registrati con **data** e **ora** e articolati in righe d'ordine di dettaglio con **quantità** e **prezzo unitario**, garantendo la correttezza dei totali e la tracciabilità nel tempo. Gli acquisti resteranno permanentemente associati al profilo dell'utente, consentendo storicizzazione e successive analisi gestionali.

La dimensione esperienziale sarà supportata da un modulo eventi: la creazione degli eventi è affidata a dipendenti autorizzati e prevede l'indicazione dei **posti disponibili**. Gli utenti potranno iscriversi (iscrizione evento) specificando il **numero di partecipanti**, mentre il sistema dovrà prevenire il superamento della capienza e registrare automaticamente **data** e **ora** di ogni iscrizione. In parallelo, la gestione interna del personale è fondata su ruoli e turni: ogni dipendente possiede un **ruolo** corrente, con storico delle variazioni per fini di audit, e partecipa a una pianificazione che combina modelli di turno (**giorno della settimana**, **nome**, **orari**) con assegnazioni di turno a calendario per **date** specifiche. Ogni assegnazione registra lo **stato** effettivo (**programmato**, **completato**, **assente**) e impedisce conflitti, evitando che un dipendente risulti pianificato su più turni nello stesso giorno.

A livello trasversale, la piattaforma tutela integrità e sicurezza dei dati: **prezzi** e **quantità** devono essere sempre positivi, le relazioni fra utenti, dipendenti, ordini ed eventi rispettano vincoli referenziali, e le informazioni sensibili come le **password** sono gestite in modo sicuro.

Il titolare dispone di una visione complessiva tramite report essenziali su vendite, adesioni agli eventi e presenza del personale, mentre l'interfaccia privilegia semplicità e rapidità nelle operazioni più frequenti.

Capitolo 2

Progettazione concettuale

In questo capitolo presenteremo lo schema ER, partendo da una versione iniziale e migliorandola passo dopo passo ad arrivare a quella definitiva, attraverso dei raffinamenti.

2.1 Schema iniziale

Dopo aver eseguito l'analisi del dominio iniziale, abbiamo creato uno schema di base con le entità e le relazioni principali, che sarà poi perfezionato nei passaggi successivi.

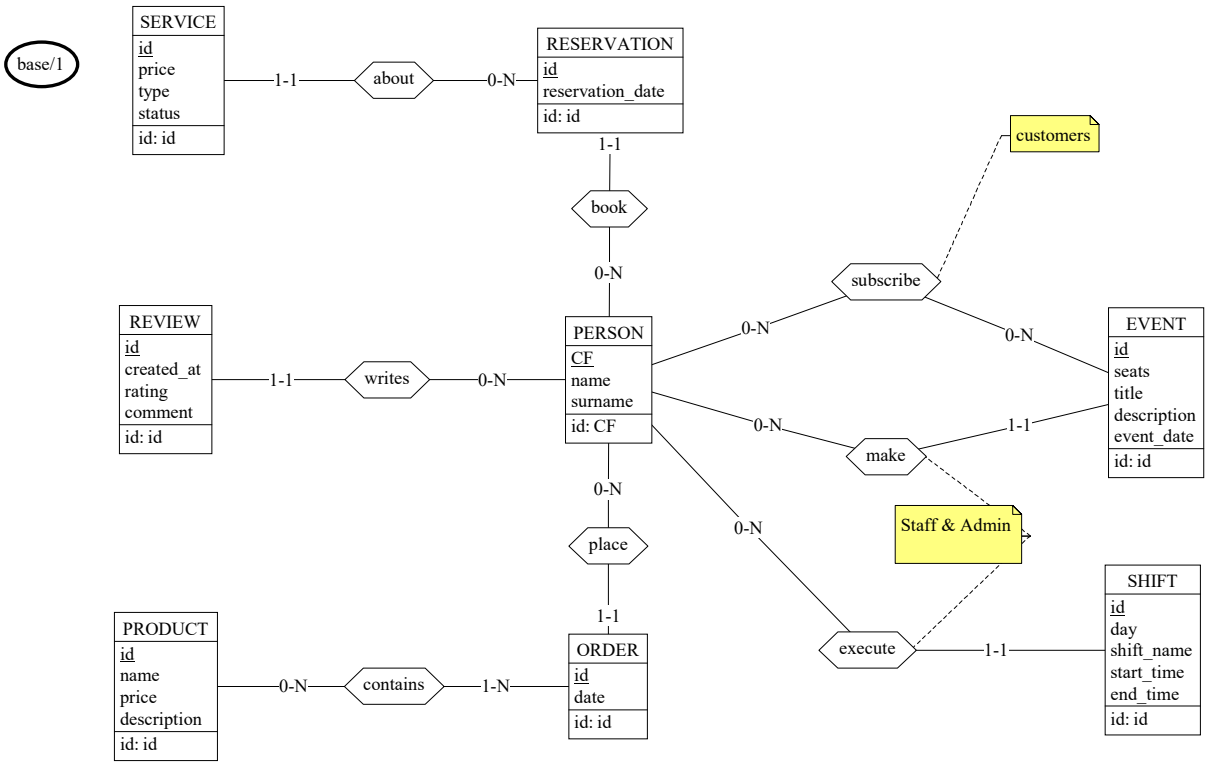

Figura 2.1: Schema ER iniziale

2.2 Raffinamenti proposti

2.2.1 Utente e Dipendente

Nel modello concettuale iniziale la **Persona** raggruppava tutte le possibili interazioni con il sistema: iscrizione, creazione di eventi, prenotazioni, ordini e recensioni. Questo approccio, sebbene corretto dal punto di vista logico, risultava poco chiaro perché attribuiva a un'unica entità responsabilità molto eterogenee.

Per migliorare la rappresentazione è stato introdotto un raffinamento mediante generalizzazione/specializzazione: la superclasse **Persona** è stata mantenuta per raccogliere gli attributi comuni (CF, nome, cognome), mentre le funzionalità specifiche sono state assegnate ai sottotipi **Cliente** e **Dipendente**.

In questo modo i clienti gestiscono attività come acquisti, recensioni, ordini e iscrizioni agli eventi, mentre i dipendenti si occupano della creazione degli eventi e della gestione dei servizi. Tale raffinamento migliora la chiarezza semantica del modello, riduce le ambiguità e riflette meglio la separazione dei ruoli reali all'interno del dominio applicativo.

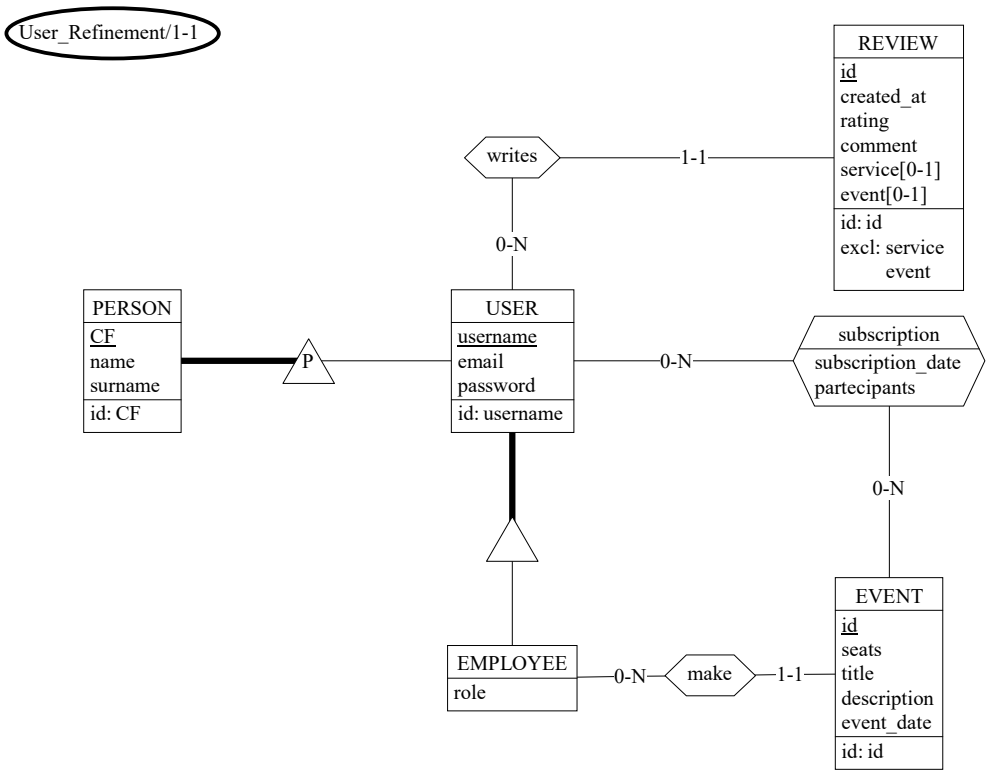

Figura 2.2: Raffinamento utente e dipendente

2.2.2 Prenotazione Servizi

Nel modello iniziale i diversi tipi di servizi potevano essere rappresentati come entità distinte, con il rischio però di ridondanza e frammentazione dei dati.

Con il raffinamento si è introdotta una **generalizzazione**: è stata creata la superclasse **Servizio**, che raccoglie gli attributi comuni (id, price, type), mentre ciascuna tipologia specifica di servizio (Camera e Ristorante) è modellata come sottoclasse.

Inoltre, è stato introdotto il legame con l'entità **Prenotazione**, che consente di registrare le informazioni su data di inizio e fine e di associare ogni prenotazione a uno o più servizi specifici tramite la relazione con **Dettagli Prenotazione**. Questo raffinamento permette di gestire correttamente scenari in cui un utente prenota più servizi differenti nello stesso arco temporale.

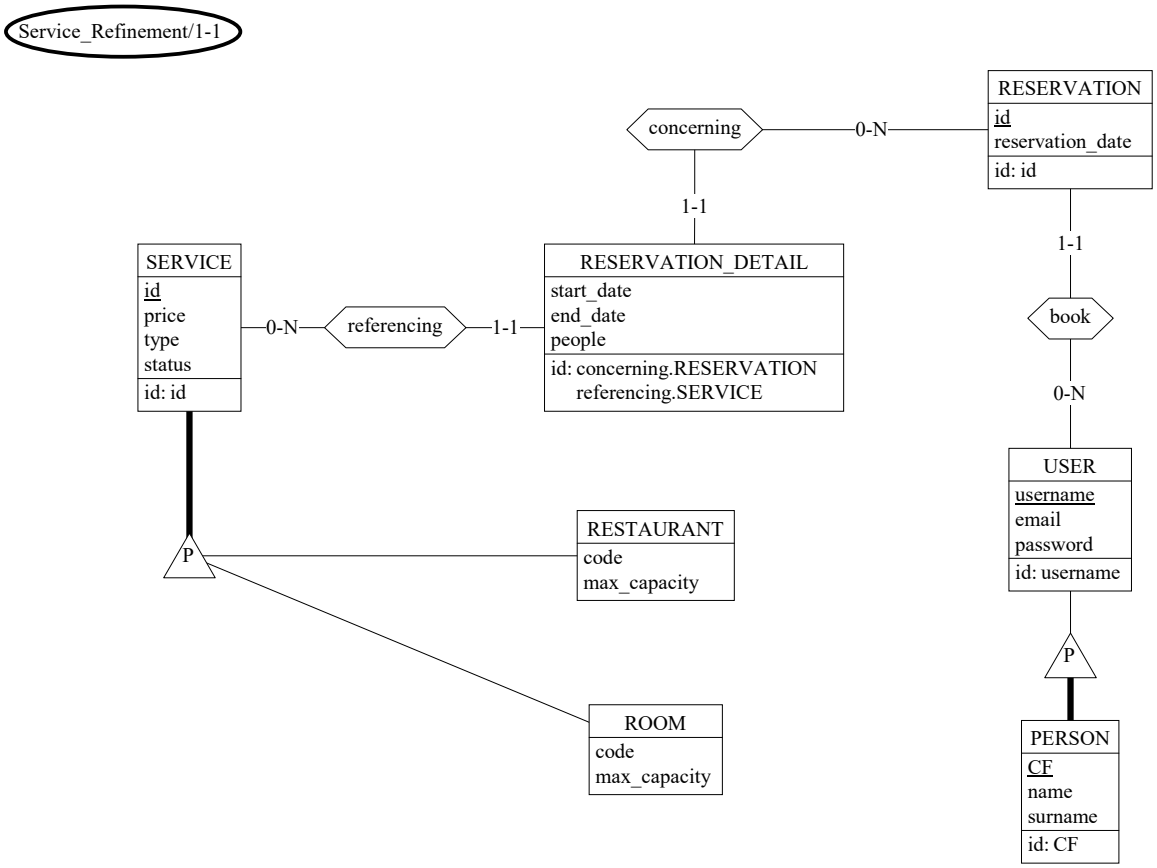

Figura 2.3: Raffinamento prenotazione e servizi

2.2.3 Prodotti e ordini

Nel modello concettuale iniziale, la gestione degli ordini e dei prodotti risultava poco dettagliata: un ordine era semplicemente collegato a uno o più prodotti, senza possibilità di specificare informazioni aggiuntive come quantità o prezzo unitario.

Con il raffinamento, è stata introdotta l'entità **Dettaglio Ordine**, che funge da associazione tra **Ordine** e **Prodotto**. Ogni dettaglio ordine consente di memorizzare, per ciascun prodotto incluso in un ordine, la quantità acquistata e il prezzo applicato. Questo permette di rappresentare in modo accurato scenari reali come ordini multiprodotto, applicazione di sconti o variazioni di prezzo nel tempo.

Inoltre, viene mantenuta la generalizzazione tra **Persona** e **Utente**, già introdotta nei raffinamenti precedenti, per distinguere i dati anagrafici comuni da quelli specifici per l'accesso al sistema e la gestione degli ordini. Questo approccio migliora la flessibilità e la chiarezza del modello, consentendo una gestione più efficace delle informazioni relative agli acquisti.

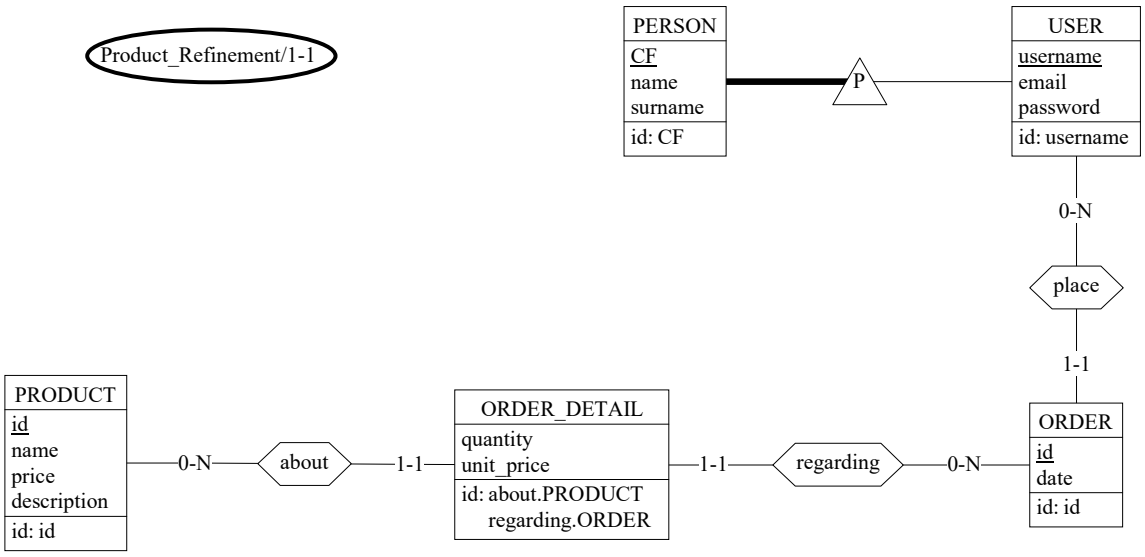

Figura 2.4: Raffinamento prodotti e ordini

2.3 Schema concettuale finale

Qui di seguito, è presente lo schema concettuale finale con tutti i raffinamenti.

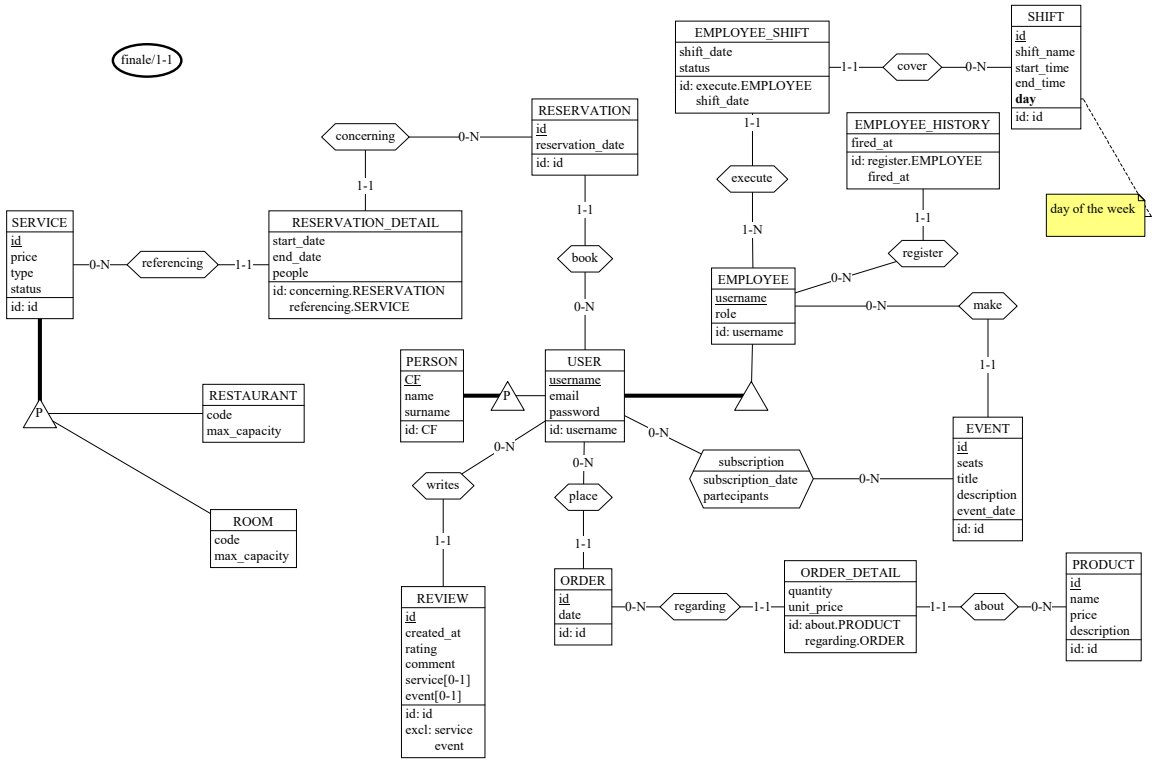

Figura 2.5: Schema ER, schema concettuale finale

Capitolo 3

Progettazione Logica

Capitolo 4

Progettazione della Base di Dati

Una volta creato il nostro database, riportiamo di seguito una parte del codice relazionale utilizzato per la sua implementazione.

4.1 Check

Sono stati utilizzati vincoli di tipo `CHECK` per definire alcuni domini e assicurare semplici proprietà degli attributi. Di seguito un esempio di vincolo `CHECK` utilizzato per assicurare che il prezzo di ogni prodotto sia maggiore zero:

```
CREATE TABLE PRODUCT (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    price DECIMAL(8,2) NOT NULL CHECK (price > 0)
);
```

4.2 Viste

La seguente vista `active_employees` restituisce l'elenco dei dipendenti attivi, mostrando per ciascuno username, email, nome, cognome e ruolo. Un dipendente è considerato attivo se il suo username non compare nella tabella `EMPLOYEE_HISTORY`, che traccia lo storico delle variazioni di stato.

```
CREATE VIEW active_employees AS
SELECT
    e.username,
    u.email,
    p.name,
    p.surname,
    e.role
FROM EMPLOYEE e
JOIN USER u ON e.username = u.username
JOIN PERSON p ON u.cf = p.cf
WHERE e.username NOT IN (
    SELECT username FROM EMPLOYEE_HISTORY
);
```

4.3 Trigger

Esempio di trigger per vincolare le recensioni: impedisce di recensire sia evento che servizio insieme, e consente la recensione solo se l'utente ha partecipato all'evento (già svolto) o ha usufruito del servizio.

```
DROP TRIGGER IF EXISTS trg_review_before_insert;
DELIMITER $$
CREATE TRIGGER trg_review_before_insert
BEFORE INSERT ON REVIEW
FOR EACH ROW
BEGIN
    DECLARE cnt INT DEFAULT 0;

    IF (NEW.event IS NOT NULL AND NEW.service IS NOT NULL) OR (NEW.event IS NULL AND NEW.service IS NULL) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Set either event or service (not both) for the review.';
    END IF;

    IF NEW.event IS NOT NULL THEN
        SELECT COUNT(*)
            INTO cnt
            FROM EVENT e
            JOIN EVENT_SUBSCRIPTION es
                ON es.event = e.id
             AND es.`user` = NEW.`user`
         WHERE e.id = NEW.event
             AND e.event_date < CURDATE();

        IF cnt = 0 THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'You can review the event only if you were subscribed and the event date is in the past.';
        END IF;
    END IF;

    IF NEW.service IS NOT NULL THEN
        SELECT COUNT(*)
            INTO cnt
            FROM RESERVATION r
            JOIN RESERVATION_DETAIL rd
                ON rd.reservation = r.id
             AND rd.service = NEW.service
         WHERE r.username = NEW.`user`
             AND rd.end_date < NOW();

        IF cnt = 0 THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'You can review the service only after you have used it (completed reservation).';
        END IF;
    END IF;
END$$
DELIMITER ;
```

4.4 Traduzione delle operazioni

Appendice A

Guida Utente

A.1 Clonazione del repository

Clonare il progetto da GitHub e accedere alla cartella:

```
> git clone https://github.com/alessandrorebosio/D25-farmhouse.git
> cd DB25-farmhouse
```

A.2 Installazione delle dipendenze

Si consiglia di utilizzare un ambiente virtuale Python per isolare le dipendenze del progetto.

```
> python3 -m venv venv
```

Attivazione dell'ambiente virtuale

```
# Su Linux/macOS:
> source venv/bin/activate
# Su Windows:
> venv\Scripts\activate
```

Installazione delle dipendenze dal file requirements.txt

```
> pip install -r requirements.txt
```

A.3 Creazione del database

Per creare il database MySQL a partire dagli script SQL forniti, assicurarsi di avere MySQL installato e in esecuzione.

```
> mysql -u root -p < app/sql/db.sql
> mysql -u root -p < app/sql/demo.sql
```

Verrà richiesta la password dell'utente `root`. Il comando eseguirà tutte le istruzioni SQL contenute nel file `db.sql`, creando tabelle, vincoli e dati di esempio necessari per l'applicazione.

A.4 Avvio dell'applicazione

Per avviare l'applicazione Django, assicurarsi che l'ambiente virtuale sia attivo e che il database sia stato creato correttamente.

```
> python manage.py migrate
> python manage.py runserver
```

L'applicazione sarà accessibile all'indirizzo `http://localhost:8000/` tramite browser. Effettuare il login o la registrazione per iniziare a utilizzare il sistema.