Università degli Studi di Napoli Federico II
Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di laurea in Informatica
Insegnamento di Basi di Dati I
Anno Accademico 2019-2020

Progettazione e sviluppo di una base di dati relazionale di un software gestionale per recensioni turistiche

Autori:
Davide Pio Faicchia
MATRICOLA : N86003018
da.faicchia@studenti.unina.it

Mauro Guida
MATRICOLA : N86002889
maur.guida@studenti.unina.it

Docenti:
Prof. Francesco Cutugno
Dott. Marco Grazioso

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

Capitolo 1
Analisi del problema 5

Capitolo 2
Progettazione concettuale 6
2.1 Dizionario delle classi 7
2.2 Dizionario delle associazioni 9
2.3 Dizionario dei vincoli 10

Capitolo 3
Progettazione logica 12
3.1 Schema logico 12
3.1.1 Traduzione delle associazioni 13

Capitolo 4
Progettazione fisica 14
4.1 Note sull'implementazione 14
4.2 Definizione delle tabelle 15
4.2.1 Definizione della tabella UTENTE 15
4.2.2 Definizione della tabella MAPPA 15
4.2.3 Definizione della tabella BUSINESS 16
4.2.4 Definizione della tabella RECENSIONE 16
4.2.5 Definizione della tabella ASSOCIAZIONERAFFINAZIONE 17
4.2.6 Definizione della tabella IMMAGINEPROPRIETA 17
4.2.7 Definizione della tabella IMMAGINERECENSIONE 17
4.3 Funzioni, procedure e Trigger 18
4.3.1 Integrità delle raffinazioni per tipologia 'Ristorante' 18
4.3.2 Integrità delle raffinazioni per tipologia 'Alloggio' 18
4.3.3 Integrità delle raffinazioni per tipologia 'Attrazione' 19
4.3.4 Generazione codice verifica per Utente 20
4.3.5 Effettua login 20
4.3.6 Effettua la registrazione 20
4.3.7 Imposta nuova passoword 21
4.3.8 Controlla documenti utente 21
4.3.9 Controlla codice verifica 21
4.3.10 Inserisci documenti utente 22
4.3.11 Inserisci business 22
4.3.12 Inserisci immagini a business 22
4.3.13 Recupera codice business 23
4.3.14 Inserisci raffinazioni 23
4.3.15 Funzione INSTR 24
4.3.17 Inserisci immagine recensione 24
4.3.16 Inserisci recensione 25
4.3.18 Ricerca locale 25
4.3.19 Aggiorna media stelle 26

4.3.20 Utente con recensione 26
4.4 Chiamate SQL integrate nell'Applicativo 27
4.4.1 Recupera locali da ricerca 27
4.4.2 Recupera locali da tipo 27
4.4.3 Recupera business da codUtente 27

Capitolo 1

Analisi del problema

In questo primo capitolo verrà presentata una panoramica generale analizzato il problema al livello di astrazione più alto.

E' stata richiesta la progettazione di una base di dati relazionale per un software di recensioni turistiche, il focus del problema è stato, quindi, gestire un numero indefinito di utenti, permettendo loro l'inserimento in piattaforma di recensioni e nuove attività a scopo pubblicitario e valutativo delle stesse, su tutto il territorio nazionale (Italia).

Da come è possibile analizzare dal *class diagram* al Capitolo 2, il concetto cardine del software presentato è il concetto di **Business** , al quale è permesso a uno o più utenti di associarvi una recensione, accompagnata, o meno, da immagini e descrizione.

Tale implementazione ha richiesto l'uso di 7 Tabelle e 2 Enumerazioni, come riportato graficamente al Capitolo 2 e descritto ampiamente al paragrafo 1 del medesimo capitolo. Lo sviluppo, in oltre, non ha richiesto la ristrutturazione del *class diagram* in quanto non presenti specializzazioni o associazioni molti a molti. La completa descrizione delle tabelle e le relative associazioni può essere osservata al Capitolo 3.

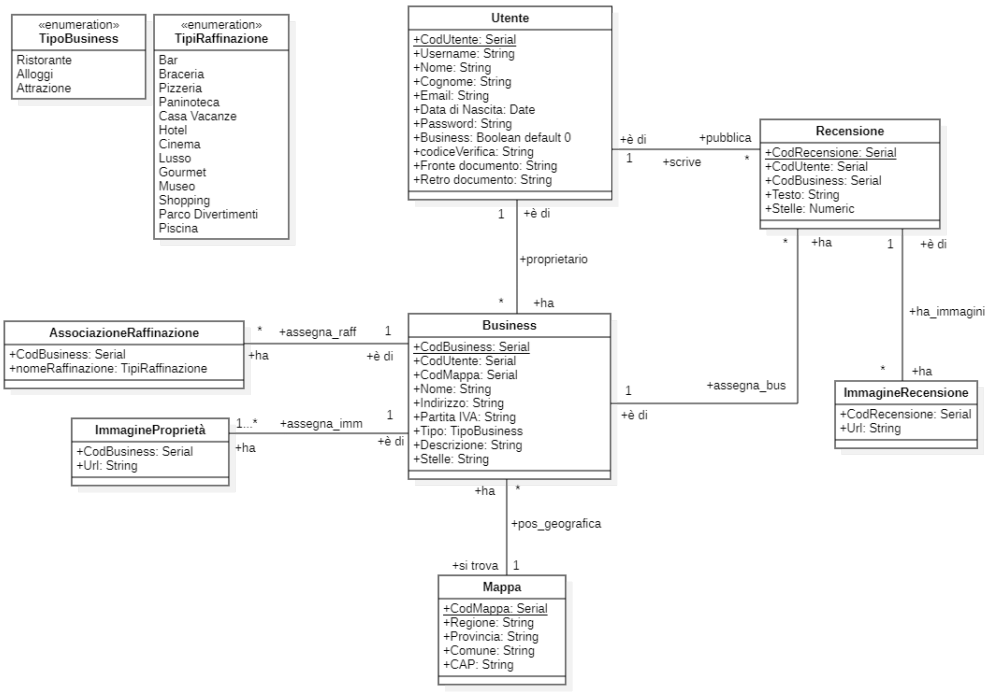
Ove possibile si è preferito evitare un'interazione diretta tra l'applicativo sviluppato ed il database. Per soddisfare tale scelta implementativa si è preferito dotare il database di funzioni specifiche (4.3), fornendo all'applicativo solo le informazioni strettamente necessarie alla corretta visualizzazione delle informazioni.

Onde evitare alterazioni *non* volontarie da parte del personale addetto alla manutenzione, con accesso diretto alla struttura dati, ogni vincolo è implementato direttamente nel database, così da impedire l'errato inserimento o modifica delle tabelle, limitando la possibilità di portare il sistema in uno stato instabile. Il *dizionario dei vincoli* può essere consultato al Capitolo 2 (2.3), dove è riportata anche una descrizione degli stessi.

Da qui in avanti saranno usati in modo intercambiabile i termini *Attività* e *Business*.

Capitolo 2

Progettazione concettuale



** Durante lo sviluppo del progetto non è stato necessario ristrutturare il class diagram.*

Le associazioni tra tabelle avvengono tramite l'uso delle chiavi primarie: CodUtente, CodBusiness, CodRecensione e CodMappa. Come si evince dal nome, tali chiavi indicano univocamente: Utenti, Attività, Recensioni di un determinato utente ad un determinato Business e uno specifico luogo sulla mappa.

Affinché fosse possibile categorizzare un Business, si sono dovuti creare delle tabelle enumerazione per definire dei tipi specifici per le 3 Attività da gestire (Ristorante, Alloggio, Attrazione) e le relative specializzazioni (4.3.1).

Un'ampia descrizione delle Classi utilizzate e i relativi vincoli, può essere consultata nei paragrafi immediatamente successivi.

2.1 Dizionario delle classi

Classe	Descrizione	Attributi
Business	Descrittore di ciascuna attività presente nel portale	**CodBusiness**(*serial*): chiave primaria. Identifica univocamente ciascuna attività. **Nome**(*varchar*): nome dell'attività. **Indirizzo**(*varchar*): indirizzo dell'attività. **PartitaIVA**(*varchar*): partitaIVA associata all'attività. **tipo**(*tipoBusiness*): tipologia dell'attività. **Descrizione**(*varchar*): descrizione attività. **Stelle**(*numeric*): valutazione corrente. **Telefono**(*varchar*): numero telefonico. **codUtente**(*integer*): chiave esterna. proprietario dell'attività. chiave esterna. **codMappa**(*integer*): chiave esterna. posizione dell'attività.
Utente	Descrittore degli utenti	**CodUtente**(*serial*): chiave primaria. Identifica univocamente ciascun utente. **Username**(*varchar*): username utilizzato dall'utente durante la registrazione. **Nome**(*varchar*): nome dell'utente. **Cognome**(*varchar*): cognome dell'utente. **Email**(*varchar*): email dell'utente. **DataDiNascita**(*date*): data nascita dell'utente. **Password**(*varchar*): password dell'utente. **CodiceVerifica**(*varchar*): codice di verifica utilizzato dall'utente durante la registrazione. **FronteDocumento**(*varchar*): immagine fronte documento inserita dall'utente per pubblicare una propria attività. **RetroDocumento**(*varchar*): immagine retro documento inserita dall'utente per pubblicare una propria attività.
Recensione	Descrittore delle recensioni inserite dagli utenti	**CodRecensione**(*serial*): chiave primaria. Identifica univocamente ciascuna recensione. **Nome**(*varchar*): testo della recensione. **Stelle**(*numeric*): valutazione locale inserita nella recensione. **codUtente**(*integer*): chiave esterna. identifica l'utente che ha scritto la recensione. **codBusiness**(*integer*): chiave esterna. identifica l'attività a cui è associata la recensione.

*Tabella 1 :*continua a pagina successiva*

Tabella 1: *continua dalla pagina precedente.*

Classe	Descrizione	Attributi
Mappa	Descrittore della posizione geografica di ciascuna attività	**CodMappa**(*serial*): chiave primaria. Identifica univocamente ciascuna posizione geografica. **Stato**(*varchar*): Attributo di locazione. **CAP**(*varchar*): Attributo di locazione. **Comune**(*varchar*): Attributo di locazione. **Provincia**(*varchar*): Attributo di locazione. **Regione**(*varchar*): Attributo di locazione. **SiglaProvincia**(*varchar*): Attributo di locazione.
Immagine Proprietà	Descrittore delle immagini inserite dal proprietario associate ad una propria attività.	**URL**(*varchar*): URL dell'immagine. **codBusiness**(*integer*): chiave esterna. Identifica l'attività a cui è associata l'immagine.
Associazione Raffinazione	Descrittore delle raffinazioni di tipologia associate a ciascuna attività.	**raffinazione**(*tipoRaffinazione*): Raffinazione di una tipologia di attività. **codBusiness**(*integer*): chiave esterna. Identifica l'attività a cui è associata la raffinazione specificata.
Immagine Recensione	Descrittore delle immagini associate a ciascuna recensione.	**URL**(*varchar*): URL dell'immagine. **codRecensione**(*integer*): chiave esterna. Identifica la recensione a cui è stata associata l'immagine.

Tabella 1 - Dizionario delle classi

2.2 Dizionario delle associazioni

Nome	Descrizione	Classi coinvolte
assegna_raff	esprime l'assegnazione di una raffinazione di tipologia a un business.	**Business**[1] ruolo **è di**: indica l'attività a cui viene assegnata la raffinazine. **AssociazioneRaffinazione**[0..*] ruolo **ha**: esprime la possibilità di associare a un'attivita delle raffinazioni
pos_geografica	esprime l'assegnazione di un'attività alla propria posizione geografica (in termini di CAP, Regione, Comune,...)	**Business**[0..*] ruolo **ha**: esprime la condizione per la quale una stessa posizione geografica può essere assegnata a più attività. **Mappa**[1] ruolo **si trova**: indica la posizione geografica associata al business.
assegna_imm	esprime l'assegnazione di una certa immagine a una attività.	**Business**[1] ruolo **è di**: esprime il business al quale è associata la coppia (*url,codBusiness*). **ImmagineProprietà**[1..*] ruolo **ha**: esprime l'immagine associata al business in termini di coppia (*url,codBusiness*).
proprietario	esprime l'assegnazione di una certa attività al suo proprietario	**Business**[0..*] ruolo **ha**: esprime la condizione per la quale un certo utente può o non avere un'attività. **Utente**[1] ruolo **è di**: definisce il proprietario dell'attività.
scrive	esprime l'assegnazione di una recensione all'utente che l'ha scritta.	**Recensione**[0..*] ruolo **pubblica**: esprime la possibilità per un utente di scrivere o meno delle recensioni. **Utente**[1] ruolo **è di**: definisce l'utente che ha scritto la recensione.
assegna_bus	esprime l'assegnazione di una certa recensione ad un'attività.	**Business**[1] ruolo **è di**: definisce il business a cui è associata la recensione **Recensione**[0..*] ruolo **ha**: definisce la possibilità di associare o meno una recensione ad un'attività.
ha_immagini	esprime l'assegnazione di una certa immagine ad una recensione.	**ImmagineRecensione**[0..*] ruolo **ha**: indica la possibilità per una recensione di avere o meno delle immagini associate. **Recensione**[1] ruolo **è di**: definisce la recensione ha cui viene associata una certa immagine.

Tabella 2 - Dizionario delle associazioni

2.3 Dizionario dei vincoli

Nome Vincolo	Descrizione
legit emails	Le emails inserite dagli utente devono avere un formato leggittimo, ergo devono contenere almeno un carattere prima della @ , almeno un carattere tra essa e il punto e almeno due caratteri nella parte finale. vedi: *4.2.1 Definizione della tabella UTENTE.*
LunghezzaPassword	Le password inserite dagli utenti devono avere lunghezza maggiore o uguale di 6. vedi: *4.2.1 Definizione della tabella UTENTE.*
RecensioneUnicaUtenteLuogo	Un utente può pubblicare al più una recensione per attività. vedi: *4.2.4 Definizione della tabella RECENSIONE.*
RaffinazioneUnica	Non è possibile associare più di una volta una stessa raffinazione a una attività. vedi: *4.2.5 Definizione della tabella ASSOCIAZIONERAFFINAZIONE.*
NumeroDiTelefonoNonValido	I numeri di telefono dei business devono avere un formato leggittimo. Devono contentere 10 cifre. vedi: *4.2.3 Definizione della tabella BUSINESS.*
NumeroDiTelefono Troppo-Corto	I numeri di telefono dei business devono avere un formato leggittimo. Devono contentere solo cifre numeriche. vedi: *4.2.3 Definizione della tabella BUSINESS.*
lunghezzaPartitaIVA	La partitaIVA deve avere esattamente 10 caratteri. vedi: *4.2.3 Definizione della tabella BUSINESS.*
DataDiNascita NonValida	Non è possibile che l'utente sia nato nell'istante presente o nel futuro. vedi: *4.2.1 Definizione della tabella UTENTE.*
noImmagineProprieta	Non è possibile associare più volte una stessa immagine a un business. vedi: *4.2.6 Definizione della tabella IMMAGINEPROPRIETA.*
checkRaffinazioneRistoranti	vedi: *4.3.1 Integrità delle raffinazioni per tipologia 'Ristorante'.*
checkRaffinazioneAlloggio	vedi: *4.3.2 Integrità delle raffinazioni per tipologia 'Alloggio'.*
checkRaffinazioneAttrazioni	vedi: *4.3.3 Integrità delle raffinazioni per tipologia 'Attrazione'.*
legit Stelle Business	Un business può avere (Stelle IS NULL) o meno una valutazione((Stelle BETWEEN 1 AND 5). vedi: *4.2.3 Definizione della tabella BUSINESS.*

*Tabella 3 - *continua a pagina successiva*

Tabella 3: *continua dalla pagina precedente.*

Nome Vincolo	**Descrizione**
legit Stelle Recensione	Una recensione *deve* avere una valutazione((Stelle BETWEEN 1 AND 5). vedi: *4.2.4 Definizione della tabella RECENSIONE.*
unique Username	Non è possibile che più utenti abbiano lo stesso Username. vedi: *4.2.1 Definizione della tabella UTENTE.*
unique PartitaIVA	Non è possibile che più business abbiano lo stesso Username. vedi: *4.2.3 Definizione della tabella BUSINESS.*
domain tipoRaffinazione	Vincolo di dominio attributo `tipo` nella tabella `BUSINESS`. vedi: *4.2.3 Definizione della tabella BUSINESS.*
domain tipoRaffinazione	Vincolo di dominio attributo `raffinazione` nella tabella `ASSOCIAZIONERAFFINAZIONE`. vedi: *4.2.3 Definizione della tabella ASSOCIAZIONERAFFINAZIONE.*
domain tipoBusiness	Vincolo di dominio attributo `tipo` nella tabella `BUSINESS`. vedi: *4.2.3 Definizione della tabella BUSINESS.*

Tabella 3 - Dizionario dei vincoli

Capitolo 3

Progettazione logica

In questo capitolo saranno presentati gli schemi logico e relazionali.

3.1 Schema logico

UTENTE	(**codUtente**, Username, Nome, Cognome, Email, Password, DataDiNascita, codiceVerifica, FronteDocumento, Retro Documento)
BUSINESS	(**codBusiness**, Nome, Indirizzo, PartitaIVA, tipo, Descrizione, Stelle, Telefono, codUtente, codMappa)
MAPPA	(**codMappa**, Stato, CAP, Regione, Comune, Provincia, SiglaProvincia)
RECENSIONE	(**codRecensione**, Testo, Stelle, codUtente, codBusiness)
IMMAGINE PROPRIETA	(URL, codBusiness)
ASSOCIAZIONE RAFFINAZIONE	(raffinazione, codBusiness)
IMMAGINE RECENSIONE	(URL, codRecensione)

Tabella 4 - Schema logico

3.1.1 Traduzione delle associazioni

Associazione	Implementazione
assegna_raff	Chiave esterna in ASSOCIAZIONERAFFINAZIONE → BUSINESS
pos_geografica	Chiave esterna in BUSINESS → MAPPA
assegna_imm	Chiave esterna in IMMAGINEPROPRIETA → BUSINESS
propretario	Chiave esterna in BUSINESS → UTENTE
scrive	Chiave esterna in RECENSIONE → UTENTE
assegna_bus	Chiave esterna in RECENSIONE → BUSINESS
ha_immagini	Chiave esterna in IMMAGINERECENSIONE → RECENSIONE

Tabella 5 - Traduzione delle associazioni

Capitolo 4

Progettazione fisica

La base di dati verrà implementata sul DBMS `PostgreSQL 11.7`

4.1 Note sull'implementazione

Al fine di agevolare lo sviluppo delle funzioni necessarie, ove possibile, saranno utilizzate funzioni e procedure caratteristiche dello standard del DBMS in questione. Tuttavia, alcune funzionalità come `INSTR` appartenente ad `ORACLE`, non disponibile su `PostgreSQL`, saranno implementate manualmente come funzioni del seguente Database, discorso equivalente per l'implementazione di `ASSERTION`, anch'essa rimpiazzata da una specifica procedure poiché non disponibili.

Nel seguente capitolo saranno mostrate le funzionalità implementate lato Database, quali: Vincoli su Tabelle, Procedure, Funzioni e Trigger, oltre ad evidenziare come ogni Tabella sia stata definita.

Nel paragrafo 4 saranno approfondite le funzioni implementate direttamente dell'applicativo e pertanto non presenti nel Database, tuttavia questa pratica, ove possibile, sia stata scongiurata per semplificare future manutenzioni ed implementazioni, rendendo possibile effettuare aggiornamenti esclusivamente lato Database senza rendere obsolete vecchie release degli applicativi.

L'obbiettivo principale è rendere ogni richiesta dell'utente una chiamata ad una funzione presente nella Base di Dati, limitando possibili manomissioni e garantendo una maggiore stabilità del sistema, oltre ad un supporto a lungo termine degli applicativi.

4.2 Definizione delle tabelle

Seguono le definizioni delle tabelle utilizzate nella base di dati, con le relative descrizioni dei vincoli adottati, per gli attributi, di ognuna di esse.

** Alcune parole risulteranno essere grammaticalmente errate data l'incompatibilità dei caratteri speciali col DBMS.*

4.2.1 Definizione della tabella UTENTE

```
/**
*   TABELLA: UTENTE
*   Crea la tabella e implementa i vincoli piu semplici
*/
CREATE TABLE Utente(
  codUtente SERIAL PRIMARY KEY,
  Username VARCHAR(50) NOT NULL UNIQUE,
  Nome VARCHAR(50) NOT NULL,
  Cognome VARCHAR(50) NOT NULL,
  Email VARCHAR(100) NOT NULL UNIQUE CHECK(Email LIKE '_%@%.__%'),
  DataDiNascita date NOT NULL,
  Password VARCHAR(100) NOT NULL,
  codiceVerifica VARCHAR(10) DEFAULT NULL,
  FronteDocumento VARCHAR(1000) DEFAULT NULL,
  RetroDocumento VARCHAR(1000) DEFAULT NULL
);

-- Vincolo di lunghezza della password
ALTER TABLE Utente
  ADD CONSTRAINT LunghezzaPassword CHECK(length(Password)>=6);

-- Vincolo di integrita della DataDiNascita inserita dall'utente
ALTER TABLE Utente
  ADD CONSTRAINT DataDiNascitaNonValida
    CHECK(NOT(DataDiNascita >= date('now')));
```

4.2.2 Definizione della tabella MAPPA

```
/**
*   TABELLA: MAPPA
*   Crea la tabella e implementa i vincoli piu semplici
*/
CREATE TABLE Mappa(
    codMappa SERIAL PRIMARY KEY
    Stato VARCHAR(5),
    CAP VARCHAR(7),
    Comune VARCHAR(100),
    Regione VARCHAR(100),
    Provincia VARCHAR(100),
    SiglaProvincia VARCHAR(5),
);
```

4.2.3 Definizione della tabella BUSINESS

```
/**
*   TABELLA: BUSINESS
*   Crea la tabella e implementa i vincoli piu semplici
*/

-- Vincolo di dominio per l'attributo 'tipo'
CREATE TYPE tipoBusiness AS ENUM ('Attrazione', 'Alloggio', 'Ristorante');

CREATE TABLE Business(
  codBusiness SERIAL PRIMARY KEY,
  Nome VARCHAR(50) NOT NULL,
  Indirizzo VARCHAR(100) NOT NULL,
  PartitaIVA VARCHAR(100) NOT NULL UNIQUE,
  tipo tipoBusiness NOT NULL,
  Descrizione VARCHAR(2000) NOT NULL,
  Stelle NUMERIC DEFAULT NULL CHECK((Stelle BETWEEN 1 AND 5)
                                    OR Stelle IS NULL),
  Telefono VARCHAR(10) NOT NULL,
  codUtente INTEGER REFERENCES Utente(codUtente) ON DELETE CASCADE,
  codMappa INTEGER REFERENCES Mappa(codMappa)
);

-- Vincolo di correttezza del numero telefonico (deve contenere solo cifre)
ALTER TABLE Business
  ADD CONSTRAINT NumeroDiTelefonoNonValido CHECK(Telefono ~ '^[0-9 ]*$');

-- Vincolo di correttezza del numero telefonico (deve contenere esattamente
    10 cifre)
ALTER TABLE Business
  ADD CONSTRAINT NumeroDiTelefonoTroppoCorto CHECK(LENGTH(Telefono) = 10);

-- Vincolo di correttezza sulla lunghezza della partitaIVA inserita
ALTER TABLE Business
  ADD CONSTRAINT lunghezzaPartitaIVA CHECK(LENGTH(PartitaIVA) = 11);
```

4.2.4 Definizione della tabella RECENSIONE

```
/**
*   TABELLA: RECENSIONE
*   Crea la tabella e implementa i vincoli piu semplici
*/
CREATE TABLE Recensione(
  CodRecensione SERIAL PRIMARY KEY,
  Testo VARCHAR(2000) NOT NULL,
  Stelle NUMERIC NOT NULL CHECK(Stelle BETWEEN 1 AND 5),
  CodBusiness INTEGER REFERENCES Business(codBusiness) ON DELETE CASCADE,
  CodUtente INTEGER REFERENCES Utente(codUtente) ON DELETE CASCADE
);

-- Non sara possibile per un utente recensire piu volte lo stesso business
ALTER TABLE Recensione
  ADD CONSTRAINT RecensioneUnicaUtenteLuogo UNIQUE(codBusiness, codUtente);
```

4.2.5 Definizione della tabella ASSOCIAZIONERAFFINAZIONE

```
/**
*   TABELLA: ASSOCIAZIONERAFFINAZIONE
*   Crea la tabella e implementa i vincoli piu semplici
*/

-- Vincolo di dominio per l'attributo 'raffinazione'
CREATE TYPE tipoRaffinazione AS
ENUM ('Pizzeria', 'Braceria', 'FastFood',
'Paninoteca', 'Osteria', 'TavolaCalda',
'Taverna', 'Trattoria', 'Pesce',
'Cinema', 'Shopping','Monumento',
'Museo', 'ParcoGiochi', 'Piscina',
'Lounge', 'Hotel', 'Bed&Breakfast',
'Ostello', 'CasaVacanze' ,'Residence');

CREATE TABLE AssociazioneRaffinazione(
  codBusiness INTEGER REFERENCES Business(codBusiness) ON DELETE CASCADE,
  raffinazione tipoRaffinazione
);

-- Non sara possibile associare piu di una volta una stessa raffinazione a
   un business
ALTER TABLE AssociazioneRaffinazione
  ADD CONSTRAINT RaffinazioneUnica UNIQUE(codBusiness,raffinazione);
);
```

4.2.6 Definizione della tabella IMMAGINEPROPRIETA

```
/**
*   TABELLA: IMMAGINEPROPRIETA
*   Crea la tabella e implementa i vincoli piu semplici
*/
CREATE TABLE ImmagineProprieta(
  Url VARCHAR(1000),
  codBusiness INTEGER REFERENCES Business(codBusiness) ON DELETE CASCADE
);

-- Non sara possibile associare piu di una volta una stessa immagine a un
   business
ALTER TABLE ImmagineProprieta
  ADD CONSTRAINT noImmagineProprieta UNIQUE(Url,CodBusiness);
```

4.2.7 Definizione della tabella IMMAGINERECENSIONE

```
/**
*   TABELLA: IMMAGINERECENSIONE
*   Crea la tabella e implementa i vincoli piu semplici
*/
CREATE TABLE ImmagineRecensione(
  Url VARCHAR(1000) NOT NULL,
  codRecensione INTEGER REFERENCES Recensione(CodRecensione) ON DELETE
   CASCADE
);
```

4.3 Funzioni, procedure e Trigger

In questo paragrafo verrà fornita una descrizione accurata dei metodi di automazione adottati, quali trigger e delle funzioni e procedure richiamate dall'applicativo con lo scopo di ricavare le informazioni necessarie al suo funzionamento.

4.3.1 Integrità delle raffinazioni per tipologia 'Ristorante'

Se l'utente seleziona come tipologia `'Ristorante'`, allora le raffinazioni devono essere necessariamente pescate tra le seguenti :
`('Pizzeria', 'Braceria', 'FastFood', 'Paninoteca', 'Osteria', 'Tavola', 'Taverna','Trattoria', 'Pesce')`

```
CREATE FUNCTION checkRaffinazioneRistoranti() RETURNS TRIGGER AS
$BODY$
DECLARE
  raff AssociazioneRaffinazione.raffinazione%TYPE;
BEGIN
  FOR raff IN SELECT raffinazione FROM AssociazioneRaffinazione WHERE
   codBusiness = NEW.codBusiness
  LOOP
    IF raff.raffinazione NOT IN
      ('Pizzeria', 'Braceria', 'FastFood',
       'Paninoteca', 'Osteria', 'Tavola Calda',
       'Taverna','Trattoria', 'Pesce') THEN
      RAISE EXCEPTION 'Errore: Raffinazione non consentita';
    END IF;
  END LOOP;
END;
$BODY$
LANGUAGE PLPGSQL;


CREATE TRIGGER checkRaffinazioneRistoranti
BEFORE INSERT ON Business
FOR EACH ROW
WHEN (NEW.tipo = 'Ristorante')
EXECUTE PROCEDURE checkRaffinazioneRistoranti();
```

4.3.2 Integrità delle raffinazioni per tipologia 'Alloggio'

Se l'utente seleziona come tipologia `'Alloggio'`, allora le raffinazioni devono essere necessariamente pescate tra le seguenti :
`('Hotel', 'Bed&Breakfast', 'Ostello', 'CasaVacanze' ,'Residence')`

```
CREATE FUNCTION checkRaffinazioneAlloggio() RETURNS TRIGGER AS
$BODY$
DECLARE
  raff AssociazioneRaffinazione.raffinazione%TYPE;
BEGIN
  FOR raff IN SELECT raffinazione FROM AssociazioneRaffinazione WHERE
   codBusiness = NEW.codBusiness
  LOOP
    IF raff.raffinazione NOT IN
      ('Hotel', 'Bed&Breakfast', 'Ostello', 'CasaVacanze' ,'Residence') THEN
      RAISE EXCEPTION 'Errore: Raffinazione non consentita';
    END IF;
  END LOOP;
END;
```

```
$BODY$
LANGUAGE PLPGSQL;


CREATE TRIGGER checkRaffinazioneAlloggio
BEFORE INSERT ON Business
FOR EACH ROW
WHEN (NEW.tipo = 'Alloggio')
EXECUTE PROCEDURE checkRaffinazioneAlloggio();
```

4.3.3 Integrità delle raffinazioni per tipologia 'Attrazione'

Se l'utente seleziona come tipologia `'Attrazione'`, allora le raffinazioni devono essere necessariamente pescate tra le seguenti :
`('Cinema', 'Shopping','Monumento', 'Museo', 'Parco Giochi', 'Piscina', 'Bar/Lounge')`

```
CREATE FUNCTION checkRaffinazioneAttrazioni() RETURNS TRIGGER AS
$BODY$
DECLARE
  raff AssociazioneRaffinazione.raffinazione%TYPE;
BEGIN
  FOR raff IN SELECT raffinazione
           FROM AssociazioneRaffinazione
           WHERE codBusiness = NEW.codBusiness LOOP
    IF raff.raffinazione NOT IN
       ('Cinema', 'Shopping','Monumento', 'Museo',
        'Parco Giochi', 'Piscina', 'Bar/Lounge') THEN
      RAISE EXCEPTION 'Errore: Raffinazione non consentita';
    END IF;
  END LOOP;
END;
$BODY$
LANGUAGE PLPGSQL;


CREATE TRIGGER checkRaffinazioneAttrazioni
BEFORE INSERT ON Business
FOR EACH ROW
WHEN (NEW.tipo = 'Attrazione')
EXECUTE PROCEDURE checkRaffinazioneAttrazioni();
```

4.3.4 Generazione codice verifica per Utente

Tale procedura genera un codice di 10 cifre e lo inserisce nel record della tabella `UTENTE` grazie al `codUtente` passato in input.

```
CREATE OR REPLACE PROCEDURE generaCodiceVerifica(INT)
LANGUAGE plpgsql
AS $$
BEGIN
  Update utente
  SET codiceVerifica=(SELECT substr(md5(random()::text), 0, 10))
  WHERE codUtente=$1;

  COMMIT;
END;
$$;
```

4.3.5 Effettua login

Tale funzione effettua l'accesso al portare tramite le credenziali d'accesso fornite dall'utente. (Attraverso il suo `codUtente`)

```
CREATE OR REPLACE FUNCTION login(INusername VARCHAR(50), INpassword VARCHAR
    (100))
RETURNS INTEGER AS $$
DECLARE codUtenteDaRestituire INTEGER;
BEGIN
        SELECT  codUtente INTO codUtenteDaRestituire
        FROM    utente
        WHERE   username = $1 AND password = $2;

        RETURN codUtenteDaRestituire;
END;
$$  LANGUAGE plpgsql;
```

4.3.6 Effettua la registrazione

Tale procedura aggiorna il database inserendo i parametri dati in input dall'utente in modo tale che egli possa effettuare il login in un secondo momento.

```
CREATE OR REPLACE PROCEDURE registrati(INusername VARCHAR(50), INnome
VARCHAR(50), INcognome VARCHAR(50), INemail VARCHAR(100), INdata DATE,
INpassword VARCHAR(100))
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO Utente(Username, Nome, Cognome, Email,
                     DataDiNascita, Password)
  Values($1,$2,$3,$4,$5,$6);

  COMMIT;
END;
$$;
```

4.3.7 Imposta nuova password

Tale procedura permette all'utente di aggiornare la password.

```
CREATE OR REPLACE PROCEDURE impostaNuovaPassword(INT, VARCHAR(50))
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE utente
  SET password = $2
  WHERE codutente=$1;

  COMMIT;
END;
$$;
```

4.3.8 Controlla documenti utente

Questa funzione controlla che l'utente abbia verificato la propria identità avendo fornito un documento. (`FronteDocumento` & `RetroDocumento`)

```
CREATE OR REPLACE FUNCTION controllaDocumentiUtente(INT)
RETURNS BOOLEAN AS $$
DECLARE flag BOOLEAN = '1';
BEGIN
  IF EXISTS (SELECT 1
       FROM Utente U
       WHERE U.codUtente = $1 AND U.FronteDocumento IS NULL
       AND U.RetroDocumento IS NULL) THEN
    flag = '0';
  END IF;
  RETURN flag;
END;
$$  LANGUAGE plpgsql;
$$;
```

4.3.9 Controlla codice verifica

Questa funzione controlla che l'utente sia verificato, ovvero abbia inserito il `'codiceVerifica'` inviatogli per email.

```
CREATE OR REPLACE FUNCTION controllaCodiceVerifica(INT, VARCHAR(10))
RETURNS BOOLEAN AS $$
DECLARE flag BOOLEAN = '0';
BEGIN
  IF EXISTS (SELECT 1
              FROM Utente U
              WHERE codUtente = $1 AND U.codiceVerifica = $2)  THEN
    flag = '1';
    UPDATE Utente
    SET codiceVerifica = NULL
    WHERE codUtente = $1;
  END IF;
  RETURN flag;
END;
$$  LANGUAGE plpgsql;
```

4.3.10 Inserisci documenti utente

Tale procedura aggiorna il database inserendo le foto del documento date in input dall'utente.

```
CREATE OR REPLACE PROCEDURE inserisciDocumentiUtente(INT, VARCHAR(1000),
    VARCHAR(1000))
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE Utente
  SET FronteDocumento = $2, RetroDocumento = $3
  WHERE codUtente = $1;

  COMMIT;
END;
$$;
```

4.3.11 Inserisci business

Tale procedura inserisce o aggiorna un business. Se la `PartitaIVA` risulta già presente nel database e l'utente che tenta di reinserirla, è il proprietario del business, allora questo verrà aggiornato, altrimenti verrà inserito ex novo.

```
CREATE OR REPLACE PROCEDURE inserisciBusiness( VARCHAR(50), VARCHAR(100),
    VARCHAR(10), VARCHAR(100), VARCHAR(100), VARCHAR(2000), INTEGER, INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
  IF EXISTS ( SELECT 1 FROM Business WHERE PartitaIVA = $4 ) THEN
  UPDATE BUSINESS
  SET Nome = $1, Indirizzo = $2, Telefono = $3,
      tipo = ($5)::tipoBusiness, Descrizione = $6
  WHERE PartitaIVA = $4;
  ELSE
  INSERT INTO Business (Nome, Indirizzo, Telefono, PartitaIVA, tipo,
                        Descrizione, codUtente, codMappa)
  VALUES ( $1, $2, $3, $4, ($5)::tipoBusiness , $6, $7, $8);
  END IF;
  COMMIT;
END;
$$;
```

4.3.12 Inserisci immagini a business

Tale procedura associata una immagine fornite dal proprietario al suo business.

```
CREATE OR REPLACE PROCEDURE inserisciImmaginiABusiness(INTEGER, VARCHAR
    (1000))
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO ImmagineProprieta(codBusiness, Url)
  VALUES($1, $2);
END;
$$;
```

4.3.13 Recupera codBusiness

Tale funzione recupera il codice del business attraverso la `PartitaIVA`.

```
CREATE OR REPLACE FUNCTION recuperaCodBusiness(VARCHAR(100))
RETURNS INTEGER
AS $$
DECLARE codiceBusiness INTEGER;
BEGIN
  SELECT codBusiness INTO codiceBusiness
  FROM Business
  WHERE PartitaIVA = $1;
  RETURN codiceBusiness;
END;
$$  LANGUAGE plpgsql;
```

4.3.14 Inserisci raffinazioni

Questa procedura, dati in input un codBusiness ed una stringa di raffinazioni separate da virgole, scompone quest'ultima estraendo le singole raffinazioni così da poterle inserire singolarmente nella tabella `ASSOCIAZIONERAFFINAZIONE`.

```
CREATE OR REPLACE PROCEDURE inserisciRaffinazioni(INTEGER, VARCHAR(3000))
LANGUAGE plpgsql
AS $$
DECLARE lunghezza INTEGER;
DECLARE stringa VARCHAR(3000);
DECLARE pos1 INTEGER;
DECLARE pos2 INTEGER;
DECLARE raffinazione VARCHAR(200);
DECLARE occorrenza INTEGER = 1;
BEGIN
  DELETE
  FROM AssociazioneRaffinazione
  WHERE codBusiness = $1;
  lunghezza = LENGTH($2);
  stringa = $2;
  pos1 = 1;
  LOOP
    pos2 = INSTR(stringa, ',', 1, occorrenza);
  EXIT WHEN pos2 = 0;
  raffinazione = SUBSTRING(stringa, pos1, pos2-pos1);
        INSERT INTO AssociazioneRaffinazione
  VALUES ($1, raffinazione::tipoRaffinazione);
  EXIT WHEN pos2 = lunghezza;
  occorrenza = occorrenza+1;
  pos1 = pos2+1;
  END LOOP;
END;
$$;
```

4.3.15 Funzione INSTR

Implementazione funzione INSTR per PostgreSQL.

```
create or replace function instr(str text, sub text, startpos int,
    occurrence int)
returns int language plpgsql
as $$
declare
    tail text;
    shift int;
    pos int;
    i int;
begin
    shift:= 0;
    if startpos = 0 or occurrence <= 0 then
        return 0;
    end if;
    if startpos < 0 then
        str:= reverse(str);
        sub:= reverse(sub);
        pos:= -startpos;
    else
        pos:= startpos;
    end if;
    for i in 1..occurrence loop
        shift:= shift+ pos;
        tail:= substr(str, shift);
        pos:= strpos(tail, sub);
        if pos = 0 then
            return 0;
        end if;
    end loop;
    if startpos > 0 then
        return pos+ shift- 1;
    else
        return length(str)- length(sub)- pos- shift+ 3;
    end if;
end $$;
```

4.3.16 Inserisci immagine recensione

Tale procedura associa un' immagine fornita dall' utente a una sua recensione.

```
CREATE OR REPLACE PROCEDURE inserisciImmagineRecensione(VARCHAR(1000),
    INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO ImmagineRecensione
  VALUES ( $1, $2);
  COMMIT;
END;
$$;
```

4.3.17 Inserisci recensione

Tale funzione inserisce una recensione nel database e restituisce il codice della stessa così da poterle associare delle immagini successivamente.

```
CREATE OR REPLACE FUNCTION inserisciRecensione(VARCHAR(2000), NUMERIC,
   INTEGER, INTEGER)
RETURNS INTEGER
AS $$
DECLARE codRecen INTEGER;
BEGIN
  INSERT INTO Recensione (Testo, Stelle, CodBusiness, CodUtente)
  VALUES ( $1, $2, $3, $4) RETURNING codRecensione INTO codRecen;
  RETURN codRecen;
END;
$$  LANGUAGE plpgsql;
```

4.3.18 Ricerca locale

Tale funzione restituisce tutte le attività trovate in base alla tipologia e/o luogo passati in input dall'utente.

```
CREATE OR REPLACE FUNCTION ricercaLocale(VARCHAR(20),VARCHAR(20))
RETURNS SETOF record
AS $$
BEGIN
  IF ( $1 = '' AND $2 = '' ) THEN
    RETURN QUERY
    SELECT B.codBusiness, B.Nome,
           B.Indirizzo, B.Stelle, I.Url
    FROM Business B, ImmagineProprieta I
    WHERE B.codBusiness = I.codBusiness AND
        I.URL = (SELECT IP.URL
                 FROM ImmagineProprieta IP
                 WHERE IP.codBusiness = B.codBusiness LIMIT 1);
  ELSE
    RETURN QUERY
    SELECT B.codBusiness, B.Nome,
           B.Indirizzo, B.Stelle, I.Url
    FROM ((Business B JOIN ImmagineProprieta I
           ON (B.codBusiness = I.codBusiness))
           JOIN Mappa M ON (B.codMappa = M.codMappa))
    WHERE I.URL = (SELECT IP.URL
                   FROM ImmagineProprieta IP
                   WHERE IP.codBusiness = B.codBusiness LIMIT 1)
                   AND ((B.Nome ILIKE '%' || $1 || '%')
                   OR (SELECT 1
                       FROM AssociazioneRaffinazione A
                       WHERE A.codBusiness = B.codBusiness AND
                       CAST(A.raffinazione AS VARCHAR(100)) ILIKE
                       '%' || $1 || '%') IS NOT NULL) AND
                             ((M.Provincia ILIKE '%' || $2 || '%') OR
                             (M.Comune ILIKE '%' || $2 || '%'));
  END IF;
END;
$$  LANGUAGE plpgsql;
```

4.3.19 Aggiorna media stelle

Tale funzione aggiorna la valutazione del locale in seguito all'aggiunta di una nuova recensione, per mezzo del trigger `'calcolaNuovaMediaDopoInserimentoRecensione'`.

```
CREATE OR REPLACE FUNCTION aggiornaMediaStelle()
RETURNS TRIGGER
AS $$
DECLARE nuovaMedia NUMERIC;
BEGIN
    SELECT AVG(Stelle) INTO nuovaMedia
    FROM Recensione
    WHERE codBusiness = NEW.codBusiness;

    UPDATE Business
    SET Stelle = nuovaMedia
    WHERE codBusiness = NEW.codBusiness;

    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER calcolaNuovaMediaDopoInserimentoRecensione
AFTER INSERT ON Recensione
FOR EACH ROW
EXECUTE PROCEDURE aggiornaMediaStelle();
```

4.3.20 Utente con recensione

Tale funzione controlla se un utente ha già scritto una recensione per un certo business.

```
CREATE OR REPLACE FUNCTION utenteConRecensione(INT, INT)
RETURNS BOOLEAN AS $$
DECLARE flag BOOLEAN = '0';
BEGIN
  IF EXISTS (SELECT 1
          FROM Recensione
          WHERE codUtente = $1 AND codBusiness = $2) THEN
    flag = '1';
  END IF;
  RETURN flag;
END;
$$  LANGUAGE plpgsql;
```

4.4 Chiamate SQL integrate nell'Applicativo

In questo paragrafo sono riportate e descritte alcune interrogazioni al Database eseguite dall'applicativo sviluppato in `Java`.

** il ? è un placehold per l'input dell'utente che verrà sostituito da una specifica funzione Java.*

4.4.1 Recupera locali da ricerca

L'utente, grazie a questa `SELECT` recupera tutti i locali con un opportuna scelta di luogo e raffinazione o nome locale.

```
SELECT codBusiness, Nome, Indirizzo, Stelle, URL
FROM ricercaLocale( ? , ? )
AS Locali(codBusiness INTEGER ,Nome VARCHAR(50),Indirizzo VARCHAR(100),
          Stelle NUMERIC,URL VARCHAR(1000))
```

4.4.2 Recupera locali da tipo

Questa `SELECT` recupera le informazioni di un locale attraverso una specifica richiesta di `tipo` tra `Attrazione, Ristorante o Alloggio`.

```
SELECT B.codBusiness, B.Nome, B.Indirizzo, B.Stelle, I.Url
FROM Business B, ImmagineProprieta I
WHERE B.codBusiness = I.codBusiness AND
      I.URL = (SELECT URL
               FROM ImmagineProprieta IP
               WHERE IP.codBusiness = B.codBusiness LIMIT 1)
               AND Tipo = ?::tipoBusiness
```

4.4.3 Recupera business da codUtente

Questa `SELECT` recupera tutte le informazioni dei business di un determinato utente.

```
SELECT B.codBusiness, B.Nome, B.Indirizzo, B.Stelle, I.Url
FROM Business B, ImmagineProprieta I
WHERE B.codBusiness = I.codBusiness AND
    I.URL = (SELECT URL
         FROM ImmagineProprieta IP
         WHERE IP.codBusiness = B.codBusiness LIMIT 1)
         AND codUtente = ?
```