OfferGo!

Corso di Basi di Dati - Prof. Vincenzo Moscato

Nome:
Coppola Chiara
Costantino Alessandro

Matricola:
N46004138
N46003730

Università degli Studi di Napoli Federico II
Facoltà di Ingegneria Informatica
Corso A-I Anno 2018/2019

1. Introduzione

Un’azienda vuole realizzare una piattaforma di informatica per raccogliere le promozioni/offerte da parte di negozi e supermercati su varie tipologie di prodotto sul territorio nazionale. Di contro ogni cliente potrà visualizzare le offerte presenti per area geografica e categoria di prodotti.
Dunque i principali servizi che deve offrire la piattaforma sono:

- Registrazione di clienti e negozi/supermercati
- Inserimento di promozioni/offerte sui prodotti con la relativa validità temporale
- Consultazione delle promozioni/offerte attive per area geografica e per categorie
- Inserimento di commenti/voti per le offerte/promozioni da parte di utenti che poi hanno acquistato o meno i prodotti

Per l’implementazione del database è stata utilizzata la distribuzione Oracle 11g XE e il tool Oracle Application Express per la creazione della web-application. Oracle Application Express (APEX) è una piattaforma di sviluppo low-code che consente di creare applicazioni enterprise scalabili e sicure, con funzioni di primissimo livello, che possono essere distribuite ovunque.
Il team di sviluppo ha proceduto nel seguente ordine:

- Progettazione concettuale/logica/fisica e implementazione del database
- Creazione della web application con l’apposito tool

2. Specifiche

2.1 Specifiche sui dati

Il committente richiede che si gestiscano le seguenti informazioni:

-Per i clienti: username, password, nome, cognome, comune di residenza, email ed opzionalmente una foto

-Per i negozi: username, password,nome, tipologia, descrizione, indirizzo completo, coordinate GPS, recapiti telefonici, fax, partita iva, l'appartenenza o meno ad un centro commerciale e se è una catena o un negozio singolo.

-Per le offerte: codice univoco all'interno del sistema(ID), il/i prodotto/i, lo stato, il periodo di validità (con data inizio e data fine dell'offerta), il prezzo (con lo sconto rispetto al prezzo originale di vendita in negozio) ed opzionalmente una breve descrizione e/o delle note aggiuntive che caratterizzano l'offerta.

-Per i prodotti: codice univoco all'interno del sistema(ID), nome, descrizione, categoria, sottocategoria e una galleria immagini.

In seguito ad analisi da parte della dirigenza della compagnia, ci si aspetta di gestire nel primo anno di esercizio del servizio:

- L'anagrafica di circa 10.000 utenti (2000 negozi e 8000 clienti)
- 8000 comuni
- Circa 20.000 offerte e 100.000 prodotti

Al fine di evitare un rallentamento progressivo delle prestazioni dovuto all'incremento dei dati, si è concordato con il committente di utilizzare un archivio storico per la gestione delle informazioni relative alle offerte concluse, dei relativi feedback e dei prodotti non più disponibili.

2.2 Specifiche sulle operazioni

In seguito alla registrazione degli utenti e all'inserimento delle informazioni riguardanti il catalogo dei prodotti e dei comuni da parte dell'azienda committente, si richiede per la piattaforma:

- che consenta la registrazione degli utenti (siano essi clienti o negozi)
- che consenta l'inserimento dei prodotti mancanti da parte dei negozi
- che consenta l'aggiunta di feedback da parte dei clienti sulle offerte
- che consenta la consultazione delle offerte disponibili
- che consenta la ricerca delle offerte per area geografica e per categoria.

Inoltre sono richieste alcune funzionalità utili a supportare analisi di tipo statistiche come il numero di offerte per città, regione o tipo di prodotto etc.

2.3 Specifiche sugli utenti della base di dati

Per la tipologia del servizio richiesto, si è optato per la creazione di un utente DBA della base dati che si occupa, tra le altre cose, di registrare gli utenti e di inserire tutte le informazioni.

Tra le possibili misure di sicurezza, meglio discusse in seguito, si sarebbe dovuto prevedere un ulteriore utente per l'applicazione web che avesse i soli privilegi strettamente necessari allo svolgimento delle operazioni sulla base dati ma dato che per lo sviluppo della web application è stato scelto il tool APEX che è un ambiente integrato con il DB Oracle non è stato necessario.

3. Progettazione concettuale

3.1 Schema E/R portante

In questa fase si è dapprima definito un modello E/R portante che cogliesse gli attori principali della nostra realtà di interesse, e successivamente si è operato per raffinamenti successivi ed estensioni, inserendo quindi gli attributi delle entità nel dettaglio e aggiungendo delle classi figlie che specificassero alcune differenze nonchè alcune entità aggiuntive utili a rappresentare al meglio la realtà di interesse .
Nel modello E/R portante sono state individuate tre entità chiave: *Utente*, *Offerta*, *Prodotto*.
Alla luce di queste semplici considerazioni iniziali uno schema portante è il seguente:

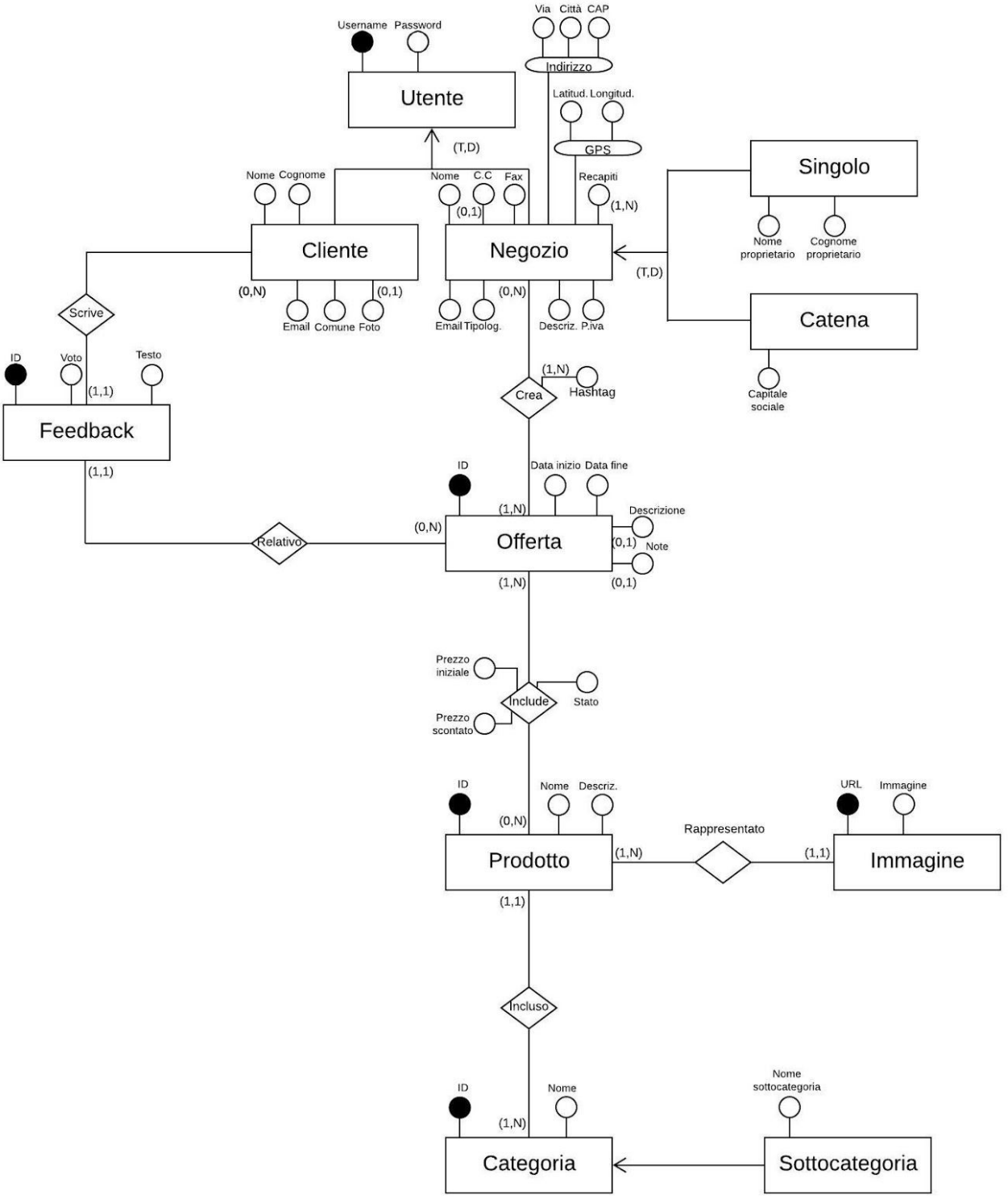
Infatti un prodotto può essere interessato da più offerte che vengono create dagli utenti. Una volta individuato il fulcro del modello E/R possiamo procedere ad una analisi più dettagliata.

3.2 Schema E/R completo

Riesaminando nel dettaglio le specifiche è immediato accorgersi del fatto che è necessario fare una distinzione, date le operazioni che possono effettuare sul sistema, tra i clienti, che visualizzano le offerte, e i negozi, creatori di offerte. Questi ultimi si dividono in negozi singoli o catene. È evidente, dunque, che l'entità *cliente* e l'entità *negozio* sono due specializzazioni di una più generale entità utente. Tale gerarchia risulta essere totale e disgiunta (un utente o è un cliente o è un negozio). *Singolo* e *Catena* sono due specializzazioni dell'entità negozio. Anche in questo caso la gerarchia risulta essere totale e disgiunta (un negozio o è un negozio singolo in proprio o fa parte di una catena). A tal proposito sono stati aggiunti gli attributi "nome proprietario" e "cognome proprietario" per il negozio singolo e "capitale sociale" per la catena. Come previsto dal committente, ogni cliente è in grado di lasciare un feedback relativo ad un'offerta, a prescindere dall'acquisto del prodotto. Ciò prevede la creazione di una nuova entità *feedback* che è caratterizzata da un ID, un voto e del testo. Un cliente può lasciare più feedback (cardinalità minima zero: il feedback non è obbligatorio) mentre un feedback è scritto da uno ed un solo cliente. Inoltre un feedback riguarda una ed una sola offerta mentre le offerte possono essere soggette a più feedback (cardinalità minima zero: un'offerta può non avere feedback). Le offerte sono relative al singolo negozio che le crea ma i negozi possono creare o nessuna (in caso di sola registrazione ho cardinalità minima zero) o più offerte. Quando un negozio crea un'offerta, queste vengono corredate di hashtag che aiutano la ricerca. Ogni offerta può essere relativa ad uno o a più prodotti mentre un prodotto può essere presente o meno in più offerte (cardinalità minima zero: un prodotto può non essere interessato da alcuna offerta in un particolare momento).

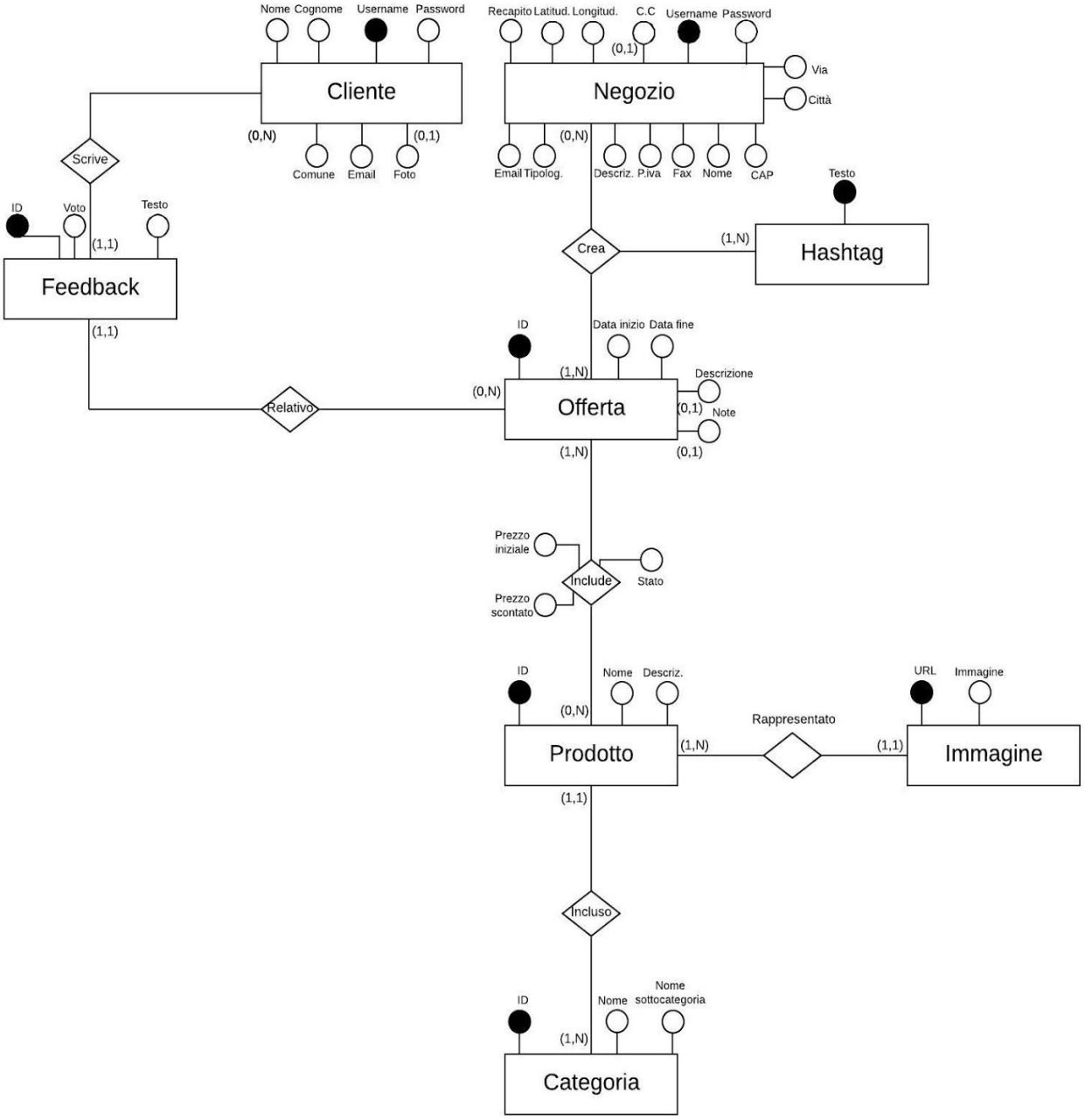
Nel momento in cui viene creata un'offerta relativa a dei prodotti, va specificato il prezzo iniziale, il prezzo scontato e lo stato del prodotto considerato. I negozi possono caricare opzionalmente delle immagini relative ai prodotti. A tal proposito è necessaria un'entità *immagine* che è collegata al prodotto. L'immagine viene memorizzata sia fisicamente che tramite il suo URL costruendo così la galleria immagini richiesta dalla società. Ogni prodotto può avere nessuna o più immagini, mentre un'immagine è relativa ad un solo prodotto. Infine, dato che il catalogo di prodotti fornito dal committente è organizzato in categorie e sottocategorie, il prodotto appartiene ad un'unica categoria mentre ad una categoria appartengono più prodotti. Ogni entità del sistema è univocamente identificata con un codice ID, tranne le immagini che vengono distinte con il loro URL.

Di seguito il modello E/R raffinato:


Username
Password
Utente
Via
Città
CAP
Indirizzo
Latitud.
Longitud.
GPS
(T,D)
Nome
Cognome
Cliente
Nome
C.C
Fax
(0,1)
Recapiti
(1,N)
Negozio
Singolo
Nome proprietario
Cognome proprietario
(T,D)
Catena
Capitale sociale
(0,N)
Email
Comune
Foto
(0,1)
(0,N)
Email
Tipolog.
Descriz.
P.iva
Scrive
ID
Voto
Testo
(1,1)
Feedback
(1,N)
Hashtag
Crea
(1,1)
ID
Data inizio
Data fine
(1,N)
Descrizione
(0,N)
Relativo
Offerta
(0,1)
Note
(0,1)
(1,N)
Prezzo iniziale
Include
Stato
Prezzo scontato
ID
Nome
Descriz.
(0,N)
Rappresentato
URL
Immagine
Prodotto
(1,N)
(1,1)
Immagine
(1,1)
Incluso
ID
Nome
(1,N)
Nome sottocategoria
Categoria
Sottocategoria

3.3 Schema E/R trasformato

A questo punto non resta che trasformare il modello E/R per poterlo successivamente tradurre nel modello logico-relazionale. La trasformazione consiste nella rimozione di quei costrutti che non sono direttamente traducibili nel modello logico, ovvero gli attributi composti, gli attributi multivalore e i costrutti di generalizzazione-specializzazione.
Nello specifico: l'attributo multivalore Hashtag dell'associazione tra Offerta e Negozio è stato tradotto in una nuova entità che ha come chiave primaria il testo dell'hashtag e che partecipa all'associazione ternaria che ne risulta con cardinalità (1,N) mentre sono stati scomposti gli attributi *indirizzo* e *GPS* della figlia Negozio e collegati alla relativa entità. Si è deciso di considerare telefono non multivalore (un solo recapito telefonico a negozio) allo scopo di velocizzare le interrogazioni sulla base di dati dato che un attributo multivalore avrebbe comportato la necessità di fare join. Inoltre dato che le specifiche non indicano attributi particolari per i negozi singoli e le catene, supponiamo che sia una specializzazione priva di attributi in modo che accorpando le figlie nel padre non ci sono valori nulli da gestire. Un'alternativa sarebbe quella di accorpare il padre nelle figlie, avendo così Clienti, Singolo e Catena come entità ma dato che le specifiche non richiedono particolare attenzione su questo punto, si sceglie la prima soluzione per semplificare. Dunque la generalizzazione/specializzazione Negozi - Singolo/Catena viene trasformata accorpando le figlie nel padre e aggiungendo l'attributo tipo che permette di distinguere a quale delle due figlie appartiene l'occorrenza. La generalizzazione/specializzazione Utenti - Clienti/Negozi viene trasformata accorpando il padre nelle figlie, in modo da avere due entità Clienti e Negozi entrambe con gli attributi del padre al loro interno, username e password. Infine la generalizzazione/specializzazione Categoria - Sottocategoria accorpando la figlia nel padre, ovvero aggiungendo l'attributo "Nome sottocategoria" all'entità Categoria. Essendo l'unica figlia non c'è bisogno dell'attributo tipo per distinguere le occorrenze e non ci sono valori nulli.
Il risultato è il seguente schema E/R trasformato:


Nome
Cognome
Username
Password
Cliente
(0,N)
(0,1)
Comune
Email
Foto
Recapito
Latitud.
Longitud.
C.C
(0,1)
Username
Password
Negozio
Via
Città
(0,N)
Email
Tipolog.
Descriz.
P.iva
Fax
Nome
CAP
Scrive
ID
Voto
Testo
(1,1)
Feedback
(1,1)
Testo
Crea
(1,N)
Hashtag
ID
Data inizio
Data fine
(1,N)
(0,N)
Relativo
Offerta
Descrizione
(0,1)
Note
(0,1)
(1,N)
Prezzo iniziale
Include
Stato
Prezzo scontato
ID
Nome
Descriz.
(0,N)
Prodotto
(1,N)
Rappresentato
(1,1)
URL
Immagine
Immagine
(1,1)
Incluso
ID
Nome
Nome sottocategoria
(1,N)
Categoria

4 Progettazione logica

4.1 Dallo schema E/R al modello logico-relazionale

Giunti a questo punto sono state seguite le regole di traduzione per costruire il modello logico-relazionale. Sono state create le relazioni (con i nomi al plurale delle entità), aventi come chiave primaria i relativi identificatori scelti in fase di progettazione. Per quanto riguarda la traduzione delle associazioni si è proceduto come segue:

- L'associazione (N,1) "*Scrive"* tra Cliente e Feedback viene tradotta eliminando l'associazione e l'identificatore lato molti (Username di Cliente) va nell'entità Feedback. Il fatto che Cliente partecipa con cardinalità minima pari a zero non influisce in alcun modo perché un feedback esiste in quanto scritto da un cliente in seguito ad una sua esperienza. Se il cliente non crea alcun feedback (non partecipa all'associazione), il feedback non esiste quindi non ho valori nulli da gestire.
- L'associazione (N,1) "*Relativo"* tra Feedback e Offerta viene tradotta eliminando l'associazione e l'identificatore lato molti (ID di Offerta) va nell'entità Feedback. Non importa la cardinalità minima zero di Offerta perché un feedback è sempre relativo ad un'offerta quindi se un'offerta non ha feedback non ci sarà la tupla all'interno della relazione (dunque non ci sono problemi di valori nulli da gestire).
- L'associazione ternaria *"Crea"* tra Negozio-Hashtag-Offerta viene tradotta creando una nuova relazione che chiameremo Neg_Off_Hash la cui chiave primaria è costituita dagli identificatori delle entità che partecipano con cardinalità massima N. Nel nostro caso tutte le entità partecipano con cardinalità massima N quindi la chiave primaria sarà Username:Negozio, ID:Offerta e Testo:Hashtag.
- L'associazione (N,N) *"Include"* tra Offerta e Prodotto viene tradotta creando una nuova relazione Offerta_Prodotti che ha come chiave primaria i due identificatori ID_Offerta e ID_Prodotto e tutti gli attributi dell'associazione.
- L'associazione (N,1) "*Rappresentato"* tra Prodotto e Immagine viene tradotta eliminando l'associazione e l'identificatore lato molti (ID di Prodotto) va nell'entità Immagine. In questo modo la galleria immagini sarà composta da un URL, dall'immagine vera e propria e dall'ID del prodotto che è presente nell'immagine
- L'associazione (1,N) "*Incluso"* tra Prodotto e Categoria viene tradotta eliminando l'associazione e l'identificatore lato molti (ID di Categoria) va nell'entità Prodotto. Secondo le specifiche, ogni prodotto è caratterizzato da una categoria (identificata con l'ID) e da una sottocategoria che è inclusa in categoria.

Il risultato di quanto detto è il seguente modello logico-relazionale:

Clienti (Username, Password, Nome, Cognome, Comune:Comuni, Email, Foto)

Feedback (ID, Cliente:Clienti, Offerta:Offerte, Voto, Testo)

Negozi (Username, Password, Nome, Recapito, Email, Fax, P.iva, Via, Città, Comune:Comuni, CAP, Latitudine, Longitudine, Centro_Comm, Tipologia, Descrizione)

Hashtag (Testo)

Offerte (ID, Data_inizio, Data_fine, Descrizione, Note)

Neg_Off_Hash (Negozio:Negozio, Offerta:Offerta, Hashtag:Hashtag)

Offerte_Prodotti (Offerta:Offerte, Prodotto:Prodotti, Prezzo_iniziale, Prezzo_scontato, Stato)

Prodotti (ID, Nome, Descrizione, Categoria:Categorie)

Immagini (URL, Prodotto:Prodotti, Immagine)

Categorie (ID, Nome, Nome_sottocategoria)

A queste tabelle derivate dalla traduzione da modello E/R a modello logico va aggiunta la tabella comuni, direttamente inserita nella base dati.

Comuni (Codice, Nome)

5 Progettazione fisica

5.1 Premesse

Dopo aver creato il modello logico-relazionale si è passati all'implementazione della base di dati sulla distribuzione Oracle 11g XE. Si è, dunque:

• Dimensionato le tabelle ed i tablespace

• Creato l'utente DBA della base di dati

• Creato le tabelle e in seguito aggiunti i relativi vincoli

• Creato i trigger e le stored procedure per automatizzare delle procedure e rendere la base dati attiva.

• Definito gli indici sulle tabelle

• Definito le politiche per la gestione della concorrenza e dell'affidabilità

Come accennato in precedenza si fa uso di quattro tabelle aggiuntive: una per lo storico delle offerte in modo da tenere traccia di tutte le offerte passate, una per lo storico dei relativi feedback ed una per ospitare le richieste di inserimento prodotti in attesa. Successivamente sono state aggiunte due ulteriori tabelle, Registrazioni_Clienti e Registrazioni_Negozi, per mantenere le richieste di registrazione da parte degli utenti (ad esempio da applicazione web). Da qui il DBA stesso potrà decidere di accettare le registrazioni (spostando gli utenti, rispettivamente, nelle tabelle Clienti e Negozi tramite le stored procedure apposite), eventualmente mandando una email di conferma e di benvenuto all'utente.

5.2 Dimensionamento tabelle e tablespace

CLIENTI

Username	VARCHAR2(20)	20 byte
Password	VARCHAR2(20)	20 byte
Nome	VARCHAR2(100)	100 byte
Cognome	VARCHAR2(100)	100 byte
Comune	CHAR(2)	2 byte
Email	VARCHAR2(50)	50 byte
Foto	BLOB	4 byte

TOT: 296 byte * 8000 clienti = 3 MB

COMUNI

Codice	CHAR(2)	2 byte
Nome	VARCHAR2(50)	50 byte

TOT: 52 byte * 8000 comuni = 1 MB

HASHTAG

Testo	VARCHAR2(100)	100 byte

TOT: 100 byte * 20000 = 2 MB

FEEDBACK

ID	NUMBER(38)	38 byte
Cliente	VARCHAR2(20)	20 byte
Offerta	NUMBER(38)	38 byte
Voto	NUMBER(1)	1 byte
Testo	CLOB	4 byte

TOT: 101 byte * 24000 clienti = 3 MB

NEGOZI

Username	VARCHAR2(20)	20 byte
Password	VARCHAR2(20)	20 byte
Nome	VARCHAR2(100)	100 byte
Recapito	CHAR(10)	10 byte
Email	VARCHAR2(50)	50 byte
Fax	CHAR(10)	10 byte
P.iva	CHAR(11)	11 byte
Via	VARCHAR2(100)	100 byte
Città	VARCHAR2(50)	50 byte
Comune	CHAR(2)	2 byte
CAP	CHAR(5)	5 byte
Latitudine	VARCHAR2(22)	22 byte
Longitudine	VARCHAR2(22)	22 byte
Centro comm.	NUMBER(1)	1 byte
Tipologia	VARCHAR2(50)	50 byte
Descrizione	CLOB	4 byte

TOT: 477 byte * 2000 = 1,2 MB

OFFERTE

ID	NUMBER(38)	38 byte
Data inizio	DATE	7 byte
Data fine	DATE	7 byte
Descrizione	CLOB	4 byte
Note	VARCHAR2(255)	255 byte

TOT: 311 byte * 20000 = 7 MB

NEG_OFF_HASH

Negozio	VARCHAR2(20)	20 byte
Offerta	NUMBER(38)	38 byte
Hashtag	VARCHAR2(100)	100 byte

TOT: 158 byte * 20000 = 4 MB

OFFERTE_PRODOTTI

Offerta	NUMBER(38)	38 byte
Prodotto	NUMBER(38)	38 byte
Prezzo iniziale	NUMBER(6,2)	8 byte
Prezzo scontato	NUMBER(6,2)	8 byte
Stato	VARCHAR2(20)	20 byte

TOT: 112 byte * 20000 = 2,5 MB

PRODOTTI

ID	NUMBER(38)	38 byte
Nome	VARCHAR2(100)	100 byte
Descrizione	CLOB	4 byte
Categoria	NUMBER(38)	38 byte

TOT: 180 byte * 100000 = 19 MB

IMMAGINI

URL	VARCHAR2(255)	255 byte
Prodotto	NUMBER(38)	38 byte
Immagine	BLOB	4 byte

TOT: 297 byte * 200000 = 61 MB

CATEGORIE

ID	NUMBER(38)	38 byte
Nome	VARCHAR2(100)	100 byte
Nome sottocategoria	VARCHAR2(100)	100 byte

TOT: 238 byte * 35 = 1 MB

Per quanto riguarda le tabelle aggiuntive:

PRODOTTI_DA_INSERIRE

Nome	VARCHAR2(100)	100 byte
Descrizione	CLOB	4 byte
Categoria	NUMBER(38)	38 byte

TOT = 204 byte * 100000 = 15 MB

N.B: I prodotti da inserire nel catalogo non sono ancora dotati di un identificativo univoco per il riconoscimento all'interno del sistema. In questo caso si considera il nome come chiave primaria.

STORICO_FEEDBACK

ID	NUMBER(38)	38 byte
Cliente	VARCHAR2(20)	20 byte
Offerta	NUMBER(38)	38 byte
Voto	NUMBER(1)	1 byte
Testo	CLOB	4 byte

TOT: 101 byte * 24000 clienti = 3 MB

STORICO_OFFERTE

Offerta	NUMBER(38)	38 byte
Prodotto	NUMBER(38)	38 byte
Prezzo iniziale	NUMBER(6,2)	8 byte
Prezzo scontato	NUMBER(6,2)	8 byte
Stato	VARCHAR2(20)	20 byte
Data inizio	DATE	7 byte
Data fine	DATE	7 byte
Descrizione	CLOB	4 byte
Note	VARCHAR2(255)	255 byte

TOT: 385 byte * 20000 = 8 MB

REGISTRAZIONE_CLIENTI

Nome	VARCHAR2(100)	100 byte
Cognome	VARCHAR2(100)	100 byte
Comune	CHAR(2)	2 byte
Email	VARCHAR2(50)	50 byte
Foto	BLOB	4 byte

TOT: 256 byte * 8000 clienti = 3 MB

N.B: I clienti che si stanno registrando al sistema non sono ancora dotati di un identificativo univoco per il riconoscimento all'interno del sistema. In questo caso si considera l'email come chiave primaria. L'email avrà il vincolo di unicità nella tabella dopo l'avvenuta registrazione.

REGISTRAZIONE_NEGOZI

Nome	VARCHAR2(100)	100 byte
Recapito	CHAR(10)	10 byte
Email	VARCHAR2(50)	50 byte
Fax	CHAR(10)	10 byte
P.iva	CHAR(11)	11 byte
Via	VARCHAR2(100)	100 byte
Città	VARCHAR2(50)	50 byte
Comune	CHAR(2)	2 byte
CAP	CHAR(5)	5 byte
Latitudine	NUMBER	22 byte
Longitudine	NUMBER	22 byte
Centro comm.	NUMBER(1)	1 byte
Tipologia	VARCHAR2(50)	50 byte
Descrizione	CLOB	4 byte

TOT: 437 byte * 2000 clienti = 1 MB

N.B: I negozi che si stanno registrando al sistema non sono ancora dotati di un identificativo univoco per il riconoscimento all'interno del sistema. In questo caso si considera l'email

come chiave primaria. L'email avrà il vincolo di unicità nella tabella dopo l'avvenuta registrazione.

Tutte le dimensioni sono arrotondate per eccesso considerando un'aliquota di tolleranza che,tra le altre cose, rende sicure alcune imprecisioni. Oracle archivia i valori del tipo di dati NUMBER in un formato a lunghezza variabile. Il primo byte è l'esponente ed è seguito da 1 a 20 byte mantissa. Arrotondando si risolve il problema della non conoscenza della precisa rappresentazione utilizzata da Oracle in termini di byte. Number ha una precisione che va da 1 a 38. Per indicare gli ID come interi si è scelto NUMBER(38) che equivale al tipo INT secondo l'ANSI. NUMBER privo di valori specificati sottintende una precisione di 22. Riguardo le entità i cui numeri da trattare non sono stati specificati dal committente, sono state fatte delle previsioni approssimative per dare una prima dimensione ai tablespace.

Sono stati, poi, dimensionati i tablespace:

- **offergo_ts** 370 MB, che contiene la maggior parte delle tabelle (la somma degli storage iniziali delle tabelle è di 184,5 MB)
- **offergo_foto_ts** 250 MB, la cui dimensione teorica sarebbe 10 GB (considerando 1 MB a foto), ma si è limitati dalla licenza di Oracle in uso
- **offergo_feedback_ts** 250 MB, che contiene i commenti dei feedback (considerando 250 caratteri a commento * 1.000.000 commenti)
- **offergo_descrizioni_ts** 40 MB, che contiene le descrizioni dei prodotti, delle offerte, e dei negozi

---

```
CREATE TABLESPACE offergo_ts DATAFILE
'D:\oraclexe\app\oracle\oradata\XE\wawa.dbf' SIZE 440 M;

CREATE TABLESPACE offergo_foto_ts DATAFILE
'D:\oraclexe\app\oracle\oradata\XE\wawa_foto.dbf' SIZE 250 M;

CREATE TABLESPACE offergo_feedback_ts DATAFILE
'D:\oraclexe\app\oracle\oradata\XE\wawa_feedback.dbf' SIZE 250 M;

CREATE TABLESPACE offergo_descrizioni_ts DATAFILE
'D:\oraclexe\app\oracle\oradata\XE\wawa_descrizioni.dbf' SIZE 40 M;
```

5.3 Creazione dell'utente DBA e delle tabelle

Le operazioni sulla base di dati di competenza del Database Administrator vengono effettuate tramite l'utente *offergo_dba*, avente gli appositi privilegi.

```
CREATE USER offergo_dba DEFAULT TABLESPACE offergo_ts IDENTIFIED BY 1234;
GRANT DBA, UNLIMITED TABLESPACE TO offergo_dba;
```

Tramite tale utente sono state create, poi, le tabelle precedentemente definite:

```
CREATE SEQUENCE contatoreProdotti;

CREATE TABLE Clienti (
    username VARCHAR2(20),
    password VARCHAR2(20) NOT NULL,
    nome VARCHAR2(100) NOT NULL,
    cognome VARCHAR2(100) NOT NULL,
    comune CHAR(2) NOT NULL,
    email VARCHAR2(50) NOT NULL,
    foto BLOB,

    CONSTRAINT PK_CLIENTI PRIMARY KEY(username),
    CONSTRAINT UC_EMAIL_CLIENTI UNIQUE (email)
)
LOB (foto) STORE AS lob_foto_clienti (TABLESPACE offergo_foto_ts) STORAGE (INITIAL
3 M);

CREATE TABLE Comuni (
    codice CHAR(2),
    nome VARCHAR2(50) NOT NULL,

    CONSTRAINT PK_COMUNI PRIMARY KEY(codice),
    CONSTRAINT UC_NOME_COMUNI UNIQUE (nome)
)
STORAGE (INITIAL 1 M);
```

```
CREATE TABLE Hashtag (
   testo VARCHAR2(100),

   CONSTRAINT PK_HASHTAG PRIMARY KEY(testo)
)
STORAGE (INITIAL 2 M);

CREATE TABLE Feedback(
   id NUMBER(38),
   cliente VARCHAR2(20) NOT NULL,
   offerta NUMBER(38) NOT NULL,
   voto NUMBER(1) NOT NULL,
   testo CLOB,

   CONSTRAINT PK_FEEDBACK PRIMARY KEY(id),
   CONSTRAINT CC_VOTO_FEEDBACK CHECK (voto >= 0 AND voto <= 5)
)
LOB (testo) STORE AS lob_commento_feedback (TABLESPACE offergo_feedback_ts)
STORAGE (INITIAL 3 M);

CREATE TABLE Negozi(
   username VARCHAR2(20),
   password VARCHAR2(20) NOT NULL,
   nome VARCHAR2(100) NOT NULL,
   recapito CHAR(10) NOT NULL,
   email VARCHAR2(50) NOT NULL,
   fax CHAR(10) NOT NULL,
   piva CHAR(11) NOT NULL,
   via VARCHAR2(100) NOT NULL,
   citta VARCHAR2(50) NOT NULL,
   comune CHAR(2) NOT NULL,
   cap CHAR(5) NOT NULL,
   latitudine NUMBER NOT NULL,
   longitudine NUMBER NOT NULL,
   centro_comm NUMBER(1) NOT NULL,
   tipologia VARCHAR2(50) NOT NULL,
   descrizione CLOB NOT NULL,

   CONSTRAINT PK_NEGOZI PRIMARY KEY(username),
   CONSTRAINT CC_CENTRO_COMM CHECK (centro_comm = 0 OR centro_comm = 1),
   CONSTRAINT UC_EMAIL_NEGOZI UNIQUE (email),
   CONSTRAINT UC_RECAPITO_NEGOZI UNIQUE (recapito)
)
LOB (descrizione) STORE AS lob_descrizione_negozi (TABLESPACE
offergo_descrizioni_ts) STORAGE (INITIAL 2 M);
```

```
CREATE TABLE Offerte(
   id NUMBER(38),
   data_inizio DATE NOT NULL,
   data_fine DATE NOT NULL,
   descrizione CLOB,
   note VARCHAR2(255),

   CONSTRAINT PK_OFFERTE PRIMARY KEY(id)
)
LOB (descrizione) STORE AS lob_descrizione_offerte (TABLESPACE
offergo_descrizioni_ts) STORAGE (INITIAL 7 M);

CREATE TABLE Neg_Off_Hash(
   negozio VARCHAR2(20),
   offerta NUMBER(38),
   hashtag VARCHAR2(100),

   CONSTRAINT PK_NEG_OFF_HASH PRIMARY KEY(negozio,offerta,hashtag)
)
STORAGE (INITIAL 4 M);

CREATE TABLE Offerte_Prodotti(
   offerta NUMBER(38),
   prodotto NUMBER(38),
   prezzo_iniziale NUMBER(6,2) NOT NULL,
   prezzo_scontato NUMBER(6,2) NOT NULL,
   stato VARCHAR2(20) NOT NULL,

   CONSTRAINT PK_OFFERTE_PRODOTTI PRIMARY KEY(offerta,prodotto)
)
STORAGE (INITIAL 3 M);

CREATE TABLE Prodotti(
   id NUMBER(38),
   nome VARCHAR2(100) NOT NULL,
   descrizione CLOB,
   categoria NUMBER(38) NOT NULL,

   CONSTRAINT PK_PRODOTTI PRIMARY KEY(id)
)
LOB (descrizione) STORE AS lob_descrizione_prodotti (TABLESPACE
offergo_descrizioni_ts) STORAGE (INITIAL 19 M);
```

```
CREATE TABLE Immagini(
   url VARCHAR2(255),
   prodotto NUMBER(38) NOT NULL,
   immagine BLOB NOT NULL,

   CONSTRAINT PK_IMMAGINI PRIMARY KEY(url)
)
LOB (immagine) STORE AS lob_immagini (TABLESPACE offergo_foto_ts) STORAGE
(INITIAL 61 M);

CREATE TABLE Categorie(
   id NUMBER(38),
   nome VARCHAR2(100) NOT NULL,
   nome_sottocategoria VARCHAR2(100) NOT NULL,

   CONSTRAINT PK_CATEGORIE PRIMARY KEY(id)
)
STORAGE (INITIAL 1 M);
```

Per quanto riguarda le tabelle aggiuntive:

```
CREATE TABLE Prodotti_Da_Inserire(
   nome VARCHAR2(100),
   descrizione CLOB,
   categoria NUMBER(38) NOT NULL,

   CONSTRAINT PK_PRODOTTI PRIMARY KEY(nome)
)
LOB (descrizione) STORE AS lob_descrizione_prodotti_da_inserire (TABLESPACE
offergo_descrizioni_ts) STORAGE (INITIAL 15 M);

CREATE TABLE Storico_Feedback(
   id NUMBER(38),
   cliente VARCHAR2(20) NOT NULL,
   offerta NUMBER(38) NOT NULL,
   voto NUMBER(1) NOT NULL,
   testo CLOB,

   CONSTRAINT PK_STORICO_FEEDBACK PRIMARY KEY(id)
)
LOB (testo) STORE AS lob_commento_storico_feedback (TABLESPACE
offergo_feedback_ts) STORAGE (INITIAL 3 M);



CREATE TABLE Storico_Offerte(
```

```
    offerta NUMBER(38),
    prodotto NUMBER(38),
    prezzo_iniziale NUMBER(6,2) NOT NULL,
    prezzo_scontato NUMBER(6,2) NOT NULL,
    stato VARCHAR2(20) NOT NULL,
    data_inizio DATE NOT NULL,
    data_fine DATE NOT NULL,
    descrizione CLOB,
    note VARCHAR2(255),

    CONSTRAINT PK_STORICO_OFFERTE PRIMARY KEY(offerta,prodotto)
)
LOB (descrizione) STORE AS lob_descrizione_storico_offerte (TABLESPACE
offergo_descrizioni_ts) STORAGE (INITIAL 8 M);

CREATE TABLE Registrazione_Clienti(
    Nome VARCHAR2(100) NOT NULL,
    Cognome VARCHAR2(100) NOT NULL,
    Comune CHAR(2) NOT NULL,
    Email VARCHAR2(50),
    Foto BLOB,

    CONSTRAINT PK_REGISTRAZIONE_CLIENTI PRIMARY KEY(email)
)
LOB (foto) STORE AS lob_foto_clienti (TABLESPACE offergo_foto_ts) STORAGE (INITIAL
3 M);

CREATE TABLE Registrazione_Negozi(
    nome VARCHAR2(100) NOT NULL,
    recapito CHAR(10) NOT NULL,
    email VARCHAR2(50),
    fax CHAR(10) NOT NULL,
    piva CHAR(11) NOT NULL,
    via VARCHAR2(100) NOT NULL,
    città VARCHAR2(50) NOT NULL,
    comune CHAR(2) NOT NULL,
    cap CHAR(5) NOT NULL,
    latitudine NUMBER NOT NULL,
    longitudine NUMBER NOT NULL,
    centro_comm NUMBER(1) NOT NULL,
    tipologia VARCHAR2(50) NOT NULL,
    descrizione CLOB NOT NULL,

    CONSTRAINT PK_REGISTRAZIONE_NEGOZI PRIMARY KEY(email),
    CONSTRAINT CC_REGISTRAZIONE_CENTRO_COMM CHECK (centro_comm = 0 OR
centro_comm = 1),
```

```
    CONSTRAINT UC_REGISTRAZIONE_RECAPITO_NEGOZI UNIQUE (recapito)
)
LOB (descrizione) STORE AS lob_descrizione_registrazione_negozi (TABLESPACE
offergo_descrizioni_ts) STORAGE (INITIAL 1 M);
```

Nei comandi SQL tutti gli attributi sono stati scritto senza spazi, senza punti, senza accenti,in minuscolo e con underscore per facilitare la scrittura dei comandi.

5.4 Aggiunta delle chiavi esterne e politiche di reazione

Le scelte per le politiche di reazione sono state le seguenti:

- Se un comune viene cancellato, le tabelle riferite dei clienti e dei negozi rimangono immutate (NO ACTION). A seguito della cancellazione nella tabella dei comuni, un trigger apposito inserirà un valore di default che segnala la presenza di un errore dovuto alla cancellazione (Si rimanda al punto 5.5 sui trigger).
- Se un negozio viene cancellato, le offerte da esso attivati e i relativi hashtag vengono cancellati dalla tabella che associa negozi, offerte ed hashtag utilizzati.
- Se un'offerta viene cancellata, il negozio che l'ha creata e i relativi hashtag vengono cancellati dalla tabella che associa negozi, offerte ed hashtag utilizzati.
- Se un hashtag viene cancellato, la tabella riferita che lega negozi, offerte e hashtag rimane immutata (NO ACTION). A seguito della cancellazione nella tabella degli hashtag, un trigger apposito inserirà un valore di default che segnala la presenza di un errore dovuto alla cancellazione (Si rimanda al punto 5.5 sui trigger).
- Se un'offerta viene cancellata, i prodotti vengono cancellati dalla tabella che unisce le offerte con i prodotti interessati.
- Se un prodotto viene cancellato, le offerte riguardanti quei prodotti vengono cancellate dalla tabella che unisce le offerte con i prodotti interessati.
- Se un cliente viene cancellato, la tabella riferita dei feedback rimane immutata (NO ACTION). Anche se un utente non fa più parte della piattaforma, il suo feedback è comunque utile per fini statistici e non può essere cancellato. A seguito della cancellazione, un trigger apposito inserirà un valore di default che segnala la presenza di un errore dovuto alla cancellazione (Si rimanda al punto 5.5 sui trigger).
- Se un'offerta viene cancellata, i feedback che riguarda quell'offerta vengono eliminati. (Si intende per offerta cancellata un'offerta che viene appositamente eliminata dagli amministratori del sistema, altrimenti dopo la data di scadenza l'offerta viene spostata all'interno dello storico).
- Se una categoria viene cancellata, anche i prodotti di quella categoria vengono cancellati.
- Se un prodotto viene cancellato, anche le immagini nella galleria vengono cancellate

In SQL:

```
ALTER TABLE Clienti ADD (constraint FK_COMUNE_CLIENTI
```

```
FOREIGN KEY (Comune)
REFERENCES Comuni(Codice));

ALTER TABLE Negozi ADD (constraint FK_COMUNE_NEGOZI
FOREIGN KEY (Comune)
REFERENCES Comuni(Codice));

ALTER TABLE Neg_Off_Hash ADD CONSTRAINT FK_NEG_OFF_HASH_NEGOZIO
FOREIGN KEY (Negozio) REFERENCES Negozi(username)
ON DELETE CASCADE

ALTER TABLE Neg_Off_Hash ADD CONSTRAINT FK_NEG_OFF_HASH_OFFERTA
FOREIGN KEY (Offerta) REFERENCES Offerte(id)
ON DELETE CASCADE

ALTER TABLE Neg_Off_Hash ADD (constraint FK_NEG_OFF_HASH_HASHTAG
FOREIGN KEY (Hashtag)
REFERENCES Hashtag(testo));

ALTER TABLE Offerte_Prodotti ADD CONSTRAINT FK_OFFERTE FOREIGN KEY (Offerta)
REFERENCES Offerte(id)
ON DELETE CASCADE

ALTER TABLE Offerte_Prodotti ADD CONSTRAINT FK_PRODOTTI FOREIGN KEY
(Prodotto) REFERENCES Prodotti(id)
ON DELETE CASCADE

ALTER TABLE Feedback ADD (constraint FK_FEEDBACK_CLIENTE
FOREIGN KEY (Cliente)
REFERENCES Clienti(username));

ALTER TABLE Feedback ADD CONSTRAINT FK_FEEDBACK_OFFERTA FOREIGN KEY
(Offerta) REFERENCES Offerte(id)
ON DELETE CASCADE

ALTER TABLE Prodotti ADD CONSTRAINT FK_PRODOTTI_CATEGORIA FOREIGN KEY
(Categoria) REFERENCES Categorie(id)
ON DELETE CASCADE

ALTER TABLE Immagini ADD CONSTRAINT FK_IMMAGINI_PRODOTTI FOREIGN KEY
(Prodotto) REFERENCES Prodotti(id)
ON DELETE CASCADE
```

5.5 Triggers

Per quanto riguarda i trigger, i quattro trigger sottostanti inseriscono all'interno di un campo della tabella referente, al seguito di una cancellazione nella tabella riferita, un valore d'errore

***. In questo modo è come se implementassimo la politica di reazione SET DEFAULT non prevista da Oracle. Il nostro valore d'errore ci consente di non perdere informazioni per noi importanti (come nel caso della cascade) ma, in caso di query o altri controlli sui dati, di vedere gli asterischi e venire a conoscenza di un errore. I trigger partono in automatico dopo un evento di cancellazione, trasformando la base dati in una base dati attiva. Si nota esplicitamente che il COMMIT delle operazioni è lasciato al chiamante delle procedure, il quale potrebbe voler compiere altre operazioni all'interno della stessa transazione.

```
CREATE OR REPLACE TRIGGER update_comuni_clienti
AFTER DELETE
ON Comuni
FOR EACH ROW
BEGIN
   UPDATE Clienti SET comune = '***'
   WHERE comune=:OLD.codice;
END;

CREATE OR REPLACE TRIGGER update_comuni_negozi
AFTER DELETE
ON Comuni
FOR EACH ROW
BEGIN
   UPDATE Negozi SET comune = '***'
   WHERE comune=:OLD.codice;
END;

CREATE OR REPLACE TRIGGER update_negoffhash
AFTER DELETE
ON Hashtag
FOR EACH ROW
BEGIN
   UPDATE Neg_Off_hash SET hashtag = '***'
   WHERE hashtag=:OLD.testo;
END;

CREATE OR REPLACE TRIGGER update_feedback
AFTER DELETE
ON Clienti
FOR EACH ROW
BEGIN
   UPDATE Feedback SET cliente = '***'
   WHERE cliente=:OLD.username;
END;
```

Successivamente è stato implementato un trigger che controllasse la validità della data di fine offerta. Ovviamente un'offerta di non può avere una data di terminazione più piccola

della data odierna, controllata automaticamente con sysdate. Il trigger lancia un'eccezione se questa condizione se verifica che visualizza a schermo un codice e un messaggio di errore.

```
CREATE OR REPLACE TRIGGER check_offerta_valida
BEFORE INSERT
ON Offerte
FOR EACH ROW
DECLARE
   late exception;
BEGIN
   IF :NEW.data_fine < sysdate
      THEN RAISE late;
   END IF;

   EXCEPTION WHEN late THEN raise_application_error(-20005, 'La data di fine offerta non può essere minore di quella attuale');
END;
```

Infine, come è ben noto, Oracle non implementa gli UPDATE a cascata. Si è allora ritenuto necessario implementare due trigger su Clienti e Negozi per garantire la consistenza dei dati tra le tabelle referenziate e referenzianti (le altre chiavi sono identificatori univoci generati come sequenze dal sistema pertanto non si prevedono modifiche su di esse).

```
CREATE OR REPLACE TRIGGER on_clienti_update AFTER UPDATE ON Clienti FOR EACH ROW
BEGIN
   UPDATE Feedback SET cliente = :new.username WHERE cliente = :old.username;
   UPDATE Storico_Feedback SET cliente = :new.username WHERE cliente = :old.username;
END;
```

```
CREATE OR REPLACE TRIGGER on_negozi_update AFTER UPDATE ON Negozi FOR EACH ROW
BEGIN
   UPDATE Neg_Off_Hash SET negozio = :new.username WHERE negozio = :old.username;
END;
```

5.5 Stored procedures e job

Oltre ai trigger sono state implementate anche delle stored procedure per realizzare alcune funzionalità della base dati e un job per automatizzare la cancellazione delle offerte. Per quanto riguarda la registrazione degli utenti, siano essi clienti o negozi, sono state

implementate due procedure per lo spostamento degli utenti dalle tabelle di registrazione provvisorie alle tabelle Clienti e Negozi. Entrambe le procedure prendono in ingresso un username e password (da assegnare all'utente) e l'email dell'utente che si intende spostare. In questo modo si potrebbe decidere in futuro di implementare un generatore automatico di username e password, nonché di mandare una notifica via email all'utente.

```
CREATE OR REPLACE PROCEDURE registra_cliente (user IN Clienti.username%TYPE,
   pass IN Clienti.password%TYPE, mail IN Clienti.email%TYPE) AS
BEGIN
   DECLARE
      cliente Registrazione_Clienti%ROWTYPE;
   BEGIN
      SELECT * INTO cliente FROM Registrazione_Clienti WHERE email = mail;
      INSERT INTO Clienti VALUES (user, pass, cliente.nome, cliente.cognome,
cliente.comune, mail, cliente.foto);
   END;
END;

CREATE OR REPLACE PROCEDURE registra_negozio (user IN Negozi.username%TYPE,
   pass IN Negozi.password%TYPE, mail IN Negozi.email%TYPE) AS
BEGIN
   DECLARE
      negozio Registrazione_Negozi%ROWTYPE;
   BEGIN
      SELECT * INTO negozio FROM Registrazione_Negozi WHERE email = mail;
      INSERT INTO Negozi VALUES (user, pass, negozio.nome, negozio.recapito, mail,
negozio.fax, negozio.piva, negozio.via, negozio.città, negozio.comune, negozio.cap,
negozio.latitudine, negozio.longitudine, negozio.centro_comm, negozio.tipologia,
negozio.descrizione);
   END;
END;
```

Successivamente è stata implementata una stored procedure che realizza l'inserimento di un prodotto mancante. I prodotti mancanti vengono inseriti, dopo la segnalazione degli utenti, nell'apposita tabella Prodotti_Da_Inserire e da lì dopo controlli sui dati inseriti devono essere spostati nella tabella Prodotti. La procedura prende in ingresso il nome del prodotto da spostare e la sequenza precedentemente creata serve a dare un ID valido al prodotto.

```
CREATE OR REPLACE PROCEDURE inserimento_prodotti_mancanti(nom IN
Prodotti_Da_Inserire.nome%TYPE) AS
BEGIN
   DECLARE
      prodotto Prodotti_Da_Inserire%ROWTYPE;
   BEGIN
      SELECT * INTO prodotto FROM Prodotti_Da_Inserire;
```

```
      INSERT INTO Prodotti VALUES (contatoreProdotti.NEXTVAL, prodotto.nome,
prodotto.descrizione, prodotto.categoria);
      DELETE FROM Prodotti_Da_Inserire WHERE nome = nom;
   END;
END;
```

La seguente stored procedure, invece, sposta le offerte con i relativi feedback nei rispettivi storici dopo che l'offerta è scaduta. Ciò viene realizzato confrontando l'attributo data fine con la data di sistema (sysdate).

```
CREATE OR REPLACE PROCEDURE spostamento_storico AS
BEGIN
   DECLARE
   CURSOR offerte_selez IS SELECT op.offerta, op.prodotto, op.prezzo_iniziale,
op.prezzo_scontato, op.stato, o.data_inizio, o.data_fine, o.descrizione, o.note
                  FROM Offerte o, Offerte_Prodotti op
                  WHERE (op.offerta = o.id) AND (o.data_fine < sysdate);

   CURSOR feedback_selez IS SELECT f.id, f.cliente, f.offerta, f.voto, f.testo
                FROM Feedback f, Offerte o
                WHERE f.offerta = o.id;

   offe Offerte_Prodotti.offerta%TYPE;
   pro Offerte_Prodotti.prodotto%TYPE;
   pri Offerte_Prodotti.prezzo_iniziale%TYPE;
   prs Offerte_Prodotti.prezzo_scontato%TYPE;
   sta Offerte_Prodotti.stato%TYPE;
   dai Offerte.data_inizio%TYPE;
   daf Offerte.data_fine%TYPE;
   des Offerte.descrizione%TYPE;
   note Offerte.note%TYPE;

   ide Feedback.id%TYPE;
   cli Feedback.cliente%TYPE;
   off1 Feedback.offerta%TYPE;
   vot Feedback.voto%TYPE;
   test Feedback.testo%TYPE;

   BEGIN
      OPEN offerte_selez;
      OPEN feedback_selez;

      LOOP
         FETCH offerte_selez INTO offe,pro,pri,prs,sta,dai,daf,des,note;
         EXIT WHEN offerte_selez%NOTFOUND;
```

```
        INSERT INTO Storico_Offerte VALUES (offe,pro,pri,prs,sta,dai,daf,des,note);
        DELETE FROM Offerte WHERE id = offe;
      END LOOP;

      LOOP
        FETCH feedback_selez INTO ide, cli, off1, vot, test;
        EXIT WHEN feedback_selez%NOTFOUND;
        INSERT INTO Storico_Feedback VALUES (ide, cli, off1, vot, test);
        DELETE FROM Feedback WHERE offerta=off1;
      END LOOP;

      CLOSE offerte_selez;
      CLOSE feedback_selez;
   END;
END;
```

Tuttavia le stored procedure, al contrario dei trigger, devono essere attivate manualmente ed è impensabile che qualcuno debba avere il compito di avviare la procedura ogni volta. Oracle mette a disposizione il DBMS_Scheduler, che si occupa di eseguire in automatico un particolare compito (*job*) e di eseguire implicitamente il commit della sua transazione. Implementiamo tale funzionalità nel seguente modo:

```
BEGIN
   DBMS_SCHEDULER.CREATE_JOB(
       job_name => 'JOB_SPOSTAMENTO_OFFERTE_FEEDBACK',
       job_type => 'STORED_PROCEDURE',
       job_action => 'spostamento_storico',
       start_date => current_timestamp,
       repeat_interval => 'FREQ=DAILY;INTERVAL=1;',
       enabled => true);
END;
```

In questo modo la stored procedure viene lanciata in automatico dal DBMS ogni giorno e le offerte scadute vengono archiviate senza dover avviare la procedura manualmente ogni volta.

5.6 Definizione delle viste e raccolta delle statistiche

Sono state in seguito create delle viste (non materializzate) sulle interrogazioni frequenti sulla base di dati per scopi statistici:

-Numero di offerte per ogni città

```
CREATE OR REPLACE VIEW NUM_OFFERTE_PER_CITTA
AS
SELECT N.CITTA, COUNT(*) AS NUM_OFFERTE
FROM NEGOZI N, NEG_OFF_HASH NO
WHERE N.USERNAME = NO.NEGOZIO
GROUP BY N.CITTA
ORDER BY N.CITTA;
```

-Numero di offerte per ogni hashtag

```
CREATE OR REPLACE VIEW NUM_OFFERTE_PER_HASHTAG
AS
SELECT HASHTAG, COUNT(*) AS NUM_OFFERTE
FROM NEG_OFF_HASH
GROUP BY HASHTAG
ORDER BY HASHTAG;
```

-Numero di offerte per ogni categoria di prodotto

```
CREATE OR REPLACE VIEW NUM_OFFERTE_PER_CATEGORIA
AS
SELECT C.NOME AS CATEGORIA, COUNT(*) AS NUM_OFFERTE
FROM CATEGORIE C, PRODOTTI P, OFFERTE_PRODOTTI OP
WHERE OP.PRODOTTO = P.ID AND P.CATEGORIA = C.ID
GROUP BY C.NOME
ORDER BY C.NOME;
```

-Numero di feedback per ogni hashtag

```
CREATE OR REPLACE VIEW NUM_FEEDBACK_PER_HASHTAG
AS
SELECT NG.HASHTAG, COUNT(*) AS NUM_FEEDBACK
FROM NEG_OFF_HASH NG, FEEDBACK F
WHERE NG.OFFERTA = F.OFFERTA
GROUP BY NG.HASHTAG
ORDER BY NG.HASHTAG;
```

-Categoria con più offerte attive

```
CREATE OR REPLACE VIEW CATEGORIA_CON_MAX_OFFERTE
AS
SELECT C.NOME AS CATEGORIA, COUNT(*) AS NUM_OFFERTE
FROM CATEGORIE C, PRODOTTI P, OFFERTE_PRODOTTI OP
WHERE OP.PRODOTTO = P.ID AND P.CATEGORIA = C.ID
```

```
GROUP BY C.NOME
HAVING COUNT(*) >= ALL
(SELECT COUNT(*)
FROM CATEGORIE C1, PRODOTTI P1, OFFERTE_PRODOTTI OP1
WHERE OP1.PRODOTTO = P1.ID AND P1.CATEGORIA = C1.ID
GROUP BY C1.NOME);
```

-Negozio con più offerte in assoluto

```
CREATE OR REPLACE VIEW NEGOZIO_CON_MAX_OFFERTE
AS
SELECT N.USERNAME, COUNT(*) AS NUM_OFFERTE
FROM NEG_OFF_HASH NO, NEGOZI N
WHERE NO.NEGOZIO = N.USERNAME
GROUP BY NEGOZIO
HAVING COUNT(*) >= ALL
(SELECT COUNT(*)
FROM NEG_OFF_HASH
GROUP BY NEGOZIO)
```

-Media dei feedback di ogni negozio

```
CREATE OR REPLACE VIEW AVG_FEEDBACK_PER_NEGOZIO
AS
SELECT N.USERNAME AS NEGOZIO, AVG(F.VOTO) AS MEDIA_FEEDBACK
FROM NEGOZI N, NEG_OFF_HASH NO, FEEDBACK F
WHERE N.USERNAME = NO.NEGOZIO AND F.OFFERTA = NO.OFFERTA
GROUP BY N.USERNAME
ORDER BY N.USERNAME;
```

-Media dei feedback per ogni offerta

```
CREATE OR REPLACE VIEW AVG_FEEDBACK_PER_OFFERTA
AS
SELECT O.ID AS OFFERTA, O.NOTE, AVG(F.VOTO) AS MEDIA_FEEDBACK
FROM OFFERTE O, FEEDBACK F
WHERE O.ID = F.OFFERTA
GROUP BY O.ID, O.NOTE
ORDER BY O.ID;
```

-Numero di offerte di ogni negozio relativo ad ogni categoria

```
CREATE OR REPLACE VIEW OFFERTE_CATEGORIA_NEGOZI
AS
SELECT C.NOME, N.USERNAME, COUNT(*) AS NUM_OFFERTE
FROM CATEGORIE C, PRODOTTI P, OFFERTE_PRODOTTI OP, NEG_OFF_HASH NO,
NEGOZI N
```

```
WHERE C.ID = P.CATEGORIA AND P.ID = OP.PRODOTTO AND NO.OFFERTA =
OP.OFFERTA AND NO.NEGOZIO = N.USERNAME
GROUP BY C.NOME, N.USERNAME;
```

Infine sono state pensate alcune query per la raccolta di semplici dati statistici sull'utilizzo del servizio (in futuro si potrebbe pensare di implementare un vero e proprio *data warehouse*):

QUERY

-Numero di negozi registrati

```
CREATE OR REPLACE VIEW NUM_NEGOZI_REGISTRATI
AS
SELECT COUNT(*) AS NUM_NEGOZI
FROM NEGOZI;
```

-Numero di clienti registrati

```
CREATE OR REPLACE VIEW NUM_CLIENTI_REGISTRATI
AS
SELECT COUNT(*) AS NUM_CLIENTI
FROM CLIENTI;
```

-Numero di negozi per ogni città

```
CREATE OR REPLACE VIEW NUM_NEGOZI_PER_CITTA
AS
SELECT CITTA, COUNT(*) AS NUM_NEGOZI
FROM NEGOZI
GROUP BY CITTA;
```

-Numero di negozi per tipologia

```
CREATE OR REPLACE VIEW NUM_NEGOZI_PER_TIPO
AS
SELECT TIPOLOGIA, COUNT(*) AS NUM_NEGOZI
FROM NEGOZI
GROUP BY CITTA;
```

-Numero di clienti per comune

```
CREATE OR REPLACE VIEW NUM_CLIENTI_PER_COMUNE
AS
SELECT COMUNE, COUNT(*) AS NUM_CLIENTI
FROM CLIENTI
GROUP BY COMUNE;
```

5.7 Definizione degli indici

Un **indice** è una struttura dati ordinata che permette di localizzare un record presente in un file dati. Esso velocizza le selezioni sui campi che compongono la **chiave di ricerca** per l'indice, ovvero una chiave composta da qualunque sottoinsieme dei campi di una relazione, diminuendo così i tempi di risposta di una query. Ad ogni file contenente i record della base di dati può essere associato un **file di indice** che contiene la struttura dati dell'indice stesso.A loro volta gli indici possono essere di tre tipi:

- **Indice primario:** è un indice costruito su di un insieme di campi che include la chiave primaria di una relazione
- **Indice secondario:** è un indice costruito su di un insieme di campi che include la chiave non primaria di una relazione
- **Indice *clustering*:** è un indice costruito su un campo non chiave, in modo che ad ogni valore di campo corrispondono più record.

Il DBMS Oracle crea in automatico degli indici definiti sulle chiavi primarie delle tabelle; a supporto di essi, si è provveduti all'aggiunta di alcuni indici secondari selezionati in base alla frequenza di utilizzo delle query, che sono i seguenti:

1. Categoria.Nome, in quanto le query sulle offerte sono spesso basate sulle categorie di appartenenza dei prodotti
2. Feedback.Offerta, in quanto la ricerca dei feedback relativi ad un'offerta sfrutta la ricerca sul campo autista dei feedback
3. Feedback.Cliente, in quanto la ricerca dei feedback relativi ad un'offerta sfrutta la ricerca sul campo cliente dei feedback
4. Comuni.Nome, dal momento che le ricerche delle offerte legate alla posizione geografica si basano sul nome della città e non sul codice.

Questi indici sono stati implementati con SQL nella seguente maniera:

```
CREATE INDEX INDICE_CATEGORIA_NOME ON CATEGORIE (NOME);
CREATE INDEX INDICE_FEEDBACK_OFFERTA ON FEEDBACK (OFFERTA);
CREATE INDEX INDICE_FEEDBACK_CLIENTE ON FEEDBACK (CLIENTE);
CREATE INDEX INDICE_COMUNI_NOME ON COMUNI (NOME);
```

5.8 Gestione della concorrenza

Il controllo di concorrenza garantisce l'integrità e la consistenza del database in caso di accesso multiplo in parallelo al database per aggiornare lo stato dei dati. Questo processo avviene tramite l'adozione di una politica di gestione delle concorrenze che prevenga anomalie al database e minimizzi i tempi di risposta. In particolare, si vogliono evitare i seguenti fenomeni:

- La **perdita di aggiornamento** di una transazione, quando un dato viene sovrascritto da un'altra transazione perdendo il valore precedente
- **L'aggiornamento fantasma**, quando dei dati che sono stati aggiornati non vengono correttamente visti da transazioni temporalmente sovrapposte
- La **lettura sporca** di un dato, quando una transazione legge un valore modificato da un'altra transazione che è stata abortita
- La **lettura inconsistente**, quando una transazione legge in due istanti diversi due valori differenti di uno stesso oggetto, modificato al contempo da un'altra transazione

Il DBMS oracle adotta il meccanismo 2PL per il controllo della concorrenza.

Locking a Due Fasi (2PL)

Il **Locking a Due Fasi** (***Two Phase Locking*** o **2PL**) è una politica di controllo di concorrenza basata su ***lock***, ovvero ogni oggetto della base di dati è protetto da un apposito blocco che permette l'accesso all'oggetto solo alla transazione che ne richiede per primo l'autorizzazione.Questo significa che dapprima una transazione "acquisisce" il blocco di un oggetto, effettuando tutte le operazioni di lettura e/o di scrittura previste, dopodiché "rilascia" il blocco che ha acquisito permettendo ad un'altra transazione di effettuare le sue operazioni sullo stesso oggetto a cui nel contempo ne aveva richiesto l'autorizzazione.Il lock in lettura su un dato oggetto può essere condiviso da più transazioni; il lock in scrittura su un dato oggetto è invece esclusivo, ovvero può essere acquisito da una sola transazione per volta. Nella tecnica 2PL una transazione si compone di due fasi:

- Nella prima fase detta **crescente**, la transazione acquisisce tutti i lock sugli oggetti su cui dovrà effettuare operazioni di lettura e scrittura.
- Nella seconda fase detta **decrescente**, solo dopo il termine di tutte le operazioni della transazione sul relativo oggetto, vengono gradualmente rilasciati tutti i lock precedentemente acquisiti.

Il 2PL impone che ogni transazione deve proteggere tutte le letture e scritture con lock, e non può acquisire altri lock dopo averne rilasciato uno. E' possibile aggiungere una terza condizione che costringe alle transazioni di rilasciare i lock solo dopo il commit o l'abort,

creando una variante del 2PL detto **2PL stretto**. Il 2PL evita i fenomeni di perdita di aggiornamenti, aggiornamento fantasma e lettura inconsistente, mentre il 2PL stretto evita anche il fenomeno della lettura sporca.

Livello di isolamento

Per le transazioni che effettuano solo letture, è possibile stabilire il cosiddetto **livello di isolamento** richiesto, che può essere uno dei seguenti:

- **READ UNCOMMITTED**: la transazione accetta di leggere dati modificati da una transazione che non ha ancora fatto il commit (ignora i lock esclusivi in scrittura e non acquisisce lock in lettura)
- **READ COMMITTED**: la transazione accetta di leggere dati modificati da una transazione solo se questa ha fatto il commit.
  Se però essa legge due volte lo stesso dato, può trovare dati diversi.
- **REPETABLE READ**: la transazione accetta di leggere dati modificati da una transazione solo se questa ha fatto il commit.
  Inoltre se un dato è letto due volte, si avrà sempre lo stesso risultato.
- **SERIALIZABLE**: produce schedule serializzabili senza anomalie.

Per il seguente progetto si è scelto il livello di isolamento REPETABLE READ in modo da evitare letture inconsistenti e garantire ai clienti una migliore esperienza di utilizzo del servizio. Questa scelta implica tuttavia l'obbligo all'utente di completare la sua transazione in un tempo limite, altrimenti la schedule del DBMS rimarrebbe bloccata troppo lungo. E' stato stabilito un tempo massimo di 10 minuti.

5.8 Controllo dell'affidabilità

Si è scelto come metodo di storage il RAID 1, che mantiene una copia esatta di tutti i dati su almeno due dischi. I vantaggi sono la sua elevata affidabilità (che aumenta linearmente con il numero di copie) e la sua velocità in lettura (che si è ritenuta più importante rispetto alla velocità in scrittura). Si deve, tuttavia, far fronte alla bassa scalabilità del sistema, presupponendo quindi che siano state effettuate corrette analisi a livello direzionale. Si nota esplicitamente che la capacità di scrittura viene ridotta a quella del disco più lento.
Il DBMS Oracle adotta in automatico alcuni meccanismi per quanto riguarda il controllo dell'affidabilità del sistema. La scrittura sulla base di dati è di tipo differito, in quanto vengono conservati i redo file ma non degli undo file. In caso di guasti soft si ricorre la procedura di ripresa a caldo che ripristina il corretto funzionamento del database a partire dall'ultimo checkpoint; in presenza di guasti hard si procede invece alla ripresa a freddo, utilizzando le copie di backup che Oracle ha effettuato (in questo caso sul disco copia) in modo da ritornare, grazie alla lettura del log, all'ultimo checkpoint ed effettuare quindi una ripresa a caldo