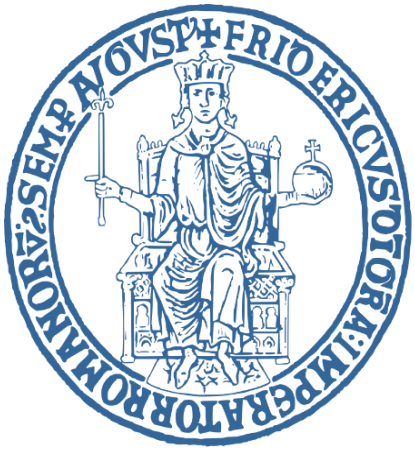
2026



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica
Basi di Dati
a.a. 2025/2026.

PROGETTO BASI DI DATI: SOCIAL

SISTEMA PER LA GESTIONE E L'ANALISI DI RETI COMMERCIALI, UTENTI E INTERAZIONI SOCIAL.

DOCENTE: PROF. VINCENZO MOSCATO

PRIMARY KEY TEAM

GIOVANNI COZZOLINO	N46007828
GRANATO ANNA PAOLA	N46007702
RUSSO ANGELO	N46007775
SAVI GIADA	D16000568

1. INTRODUZIONE

Il progetto, sviluppato nell'ambito del corso di **Basi di Dati** (a.a. 2025/2026), riguarda la progettazione e l'implementazione di una base di dati relazionale per una piattaforma dedicata alla gestione, memorizzazione e analisi delle interazioni tra utenti registrati e attività commerciali. Il sistema è concepito in modo analogo ai moderni sistemi di recensione e raccomandazione online (come Yelp o TripAdvisor) e ha l'obiettivo di integrare in un'unica struttura coerente informazioni eterogenee relative a profili utente, attività, contenuti generati dagli utenti e interazioni tra di essi, supportando sia le funzionalità operative della piattaforma sia l'esecuzione di analisi statistiche sui dati raccolti.

1.2 OBIETTIVI

L'obiettivo principale del progetto è la realizzazione di una base di dati efficiente e coerente, in grado di rappresentare in modo accurato un contesto applicativo reale. Particolare attenzione è rivolta alla corretta modellazione delle entità e delle relazioni, al rispetto dei vincoli di integrità e all'ottimizzazione delle prestazioni mediante l'uso di indici e meccanismi di controllo della concorrenza.

La finalità del progetto è dimostrare la capacità di applicare le metodologie di progettazione concettuale, logica e fisica di una base di dati, nonché di utilizzare il linguaggio SQL per implementare operazioni avanzate di gestione e analisi dei dati.

1.2 GESTIONE DEI DATI

Il progetto prevede la gestione di un sistema di dati interconnessi che descrivono le dinamiche tra attività e utenti. I dati richiesti possono essere raggruppati in quattro pilastri fondamentali:

1. **Attività**
   - **Anagrafica e Geolocalizzazione**: Ogni locale è identificato da un codice univoco e descritto da attributi spaziali (Indirizzo, Città, CAP, Latitudine e Longitudine) e operativi (Nome, Stato di apertura).
   - Le attività sono classificate in una o più categorie (es. Bar, Pizzeria) e descritte da attributi specifici come la presenza di Wi-Fi o parcheggio libero.
   - Si memorizzano i turni di apertura di ogni singolo giorno della settimana.

2. **Utenti**
   - Ogni utente ha un profilo che traccia il nome, la data di iscrizione, il volume di contenuti prodotti e una valutazione media basata sui feedback ricevuti.
   - **Social**: Il sistema deve gestire le relazioni di amicizia tra utenti e un sistema di "complimenti" (tra utenti) qualitativi suddivisi per tipologia (es. "funny", "cool").

3. Contenuti

- **Recensioni e Suggerimenti**: Gli utenti possono pubblicare recensioni (voto in stelle, testo e data) o suggerimenti rapidi (brevi testi per consigli veloci).
- **Voti**: Il database deve registrare i voti della community sulla qualità delle recensioni (utili, divertenti, interessanti) e i complimenti sui suggerimenti.
- **Fotografie**: Supporto al caricamento di foto categorizzate da etichette (es. "food", "outside") e accompagnate da una didascalia.

4. Check-in

- Registrazione dei check-in presso le attività, associando ogni evento a una o più date e orari specifici.

1.3 OPERZIONI SUL DATABASE

1. Trigger

Il database deve reagire automaticamente a determinati eventi per garantire la coerenza dei dati:

- **Aggiornamento Statistiche**: Ogni volta che viene inserita una nuova recensione (o eliminata), il sistema deve ricalcolare in automatico il numero totale di recensioni e la valutazione media (in stelle) sia per l'attività commerciale che per l'utente.
- **Integrità Sociale**: Impedire operazioni non logiche, come un utente che stringe amicizia con sé stesso o che si auto-assegna dei complimenti.
- **Coerenza Temporale**: Verificare che la data di un check-in non sia nel futuro rispetto al momento dell'inserimento.

2. Query e Viste

Le query e viste realizzate permettono di:

- Individuare le attività con la valutazione media più alta, filtrandole per città o per categoria.
- Determinare quali sono gli utenti più influenti della piattaforma in base al numero di recensioni scritte o ai complimenti ricevuti.
- Studiare la distribuzione dei check-in nel tempo per capire i picchi di affluenza (ad esempio, analizzando i dati specifici dell'anno 2021).

3. Procedure

Operazioni complesse che incapsulano il codice per renderlo riutilizzabile e sicuro:

- **Procedure di Inserimento**: Gestire l'aggiunta di nuovi utenti, business o contenuti multimediali assicurando che tutte le relazioni (chiavi esterne) siano rispettate.
- **Gestione della Moderazione**: Procedure dedicate alla rimozione o modifica dei contenuti da parte dei moderatori, garantendo che l'eliminazione di una recensione comporti il ripristino corretto delle medie precedenti.

4. Ottimizzazione e Sicurezza

- **Indici**: Definizione di indici (B-Tree o Bitmap) per velocizzare le ricerche più frequenti (come la ricerca per città o categoria).
- **Controllo della Concorrenza**: Gestire gli accessi simultanei per evitare che due utenti che scrivono una recensione nello stesso istante mandino in errore il calcolo della media.
- **Strategie di Protezione**: Implementazione di piani di backup e recovery per garantire che i dati non vadano persi in caso di guasto.

1.4 SCELTE PROGETTUALI: MODELLO THREE TIER

L'architettura **Three-Tier** (o a tre livelli) è un modello di progettazione software che suddivide un'applicazione in tre strati logici e fisici distinti, garantendo che ogni componente sia indipendente dagli altri. Nel contesto della piattaforma sociale sviluppata, questa struttura permette di gestire separatamente l'interfaccia utente, le regole di business e la persistenza dei dati

Presentation Tier (Livello di Presentazione)

Il primo livello dell'architettura è rappresentato dall'interfaccia utente, realizzata attraverso la piattaforma **Oracle Application Express (APEX)**. Questo strato funge da ponte diretto tra il sistema e gli utenti, permettendo la visualizzazione dinamica delle dashboard statistiche e dei grafici analitici richiesti dalla traccia. La sua funzione principale è quella di raccogliere gli input, come l'inserimento di nuove recensioni o il caricamento di fotografie, e di presentare i risultati delle query. Grazie a questo livello, la complessità delle tabelle e delle relazioni sottostanti viene mascherata, consentendo ai diversi ruoli (Admin, Moderator e User) di operare in un ambiente grafico semplificato e sicuro.

Logic Tier (Livello Logico o Applicativo)

Il secondo livello costituisce il nucleo operativo del sistema e contiene tutta la logica necessaria per elaborare le informazioni. In questo progetto, lo strato logico è implementato direttamente all'interno del DBMS tramite **Stored Procedure** e **Trigger**. Questi componenti automatizzano processi critici, come l'aggiornamento in tempo reale della valutazione media di un'attività dopo ogni nuova recensione o il controllo di coerenza sui check-in. Questo livello garantisce che le interazioni tra utenti e attività commerciali siano validate secondo le regole stabilite.

Data Tier (Livello dei Dati)

Il terzo livello è rappresentato dal database **Oracle** vero e proprio, deputato alla memorizzazione persistente e sicura di tutto le informazioni. In questo strato risiedono le tabelle fisiche che ospitano i dati anagrafici delle attività, i profili degli utenti e le loro interazioni (recensioni, foto e amicizie). Oltre all'archiviazione, il Data Tier gestisce l'integrità referenziale, la creazione di indici per l'ottimizzazione delle prestazioni e le strategie di backup e recovery necessarie per la protezione dei dati. La gestione di questo livello è affidata alla figura del **DBA**, che assicura che le fondamenta del sistema siano performanti e privi di errori.

1.5 POLITICA DI SICUREZZA

La sicurezza del sistema è garantita da una politica di controllo degli accessi basata sui ruoli. Questa struttura permette di separare le responsabilità e limitare il raggio d'azione di ogni utente, riducendo drasticamente il rischio di manomissioni del database. Le figure previste sono le seguenti:

- **DBA (Database Administrator):** È il garante dell'intera infrastruttura tecnologica. Opera al livello più basso e critico del sistema (Data Tier). Le sue competenze includono la gestione dell'istanza Oracle, la creazione degli schemi, l'ottimizzazione delle performance tramite indici, le politiche di backup, il ripristino dei dati (recovery) e la gestione della memoria. Il DBA non si occupa dei contenuti sociali, ma assicura che il "motore" del database sia sempre efficiente, sicuro e disponibile.
- **Admin (Amministratore Applicativo):** Rappresenta la figura di gestione superiore a livello di business. Ha il controllo totale sulla logica applicativa e sulle anagrafiche. Può censire nuove attività commerciali, modificarne gli attributi, gestire i profili degli utenti e configurare le impostazioni generali della piattaforma. È l'unico ruolo autorizzato a modificare i parametri strutturali delle categorie e dei servizi offerti.
- **Moderator (Moderatore):** È una figura intermedia focalizzata sul mantenimento della qualità dei contenuti generati dagli utenti, è il supervisore dei flussi informativi: dispone di permessi speciali per la revisione delle recensioni, dei suggerimenti e delle fotografie. Ha il potere di intervenire sui contenuti segnalati come inappropriati, potendoli modificare o rimuovere definitivamente per garantire il rispetto delle regole della community.
- **User (Utente Registrato):** Rappresenta l'utente finale della piattaforma e opera con i privilegi minimi necessari. Ogni utente può gestire esclusivamente il proprio profilo personale e i contenuti da lui stesso prodotti (le proprie recensioni, foto e check-in). Non ha permessi di scrittura sui dati di altri utenti o sulle informazioni generali delle attività commerciali. Può inoltre stabilire relazioni di amicizia e assegnare complimenti, ma sempre entro i limiti dei vincoli di coerenza imposti dal sistema (es. divieto di auto-interazione).

Questa architettura gerarchica garantisce che ogni operazione sia tracciabile e che i dati critici siano protetti da manipolazioni non autorizzate, assicurando l'integrità e l'affidabilità dell'intera base di dati.

2. PROGETTAZIONE CONCETTUALE

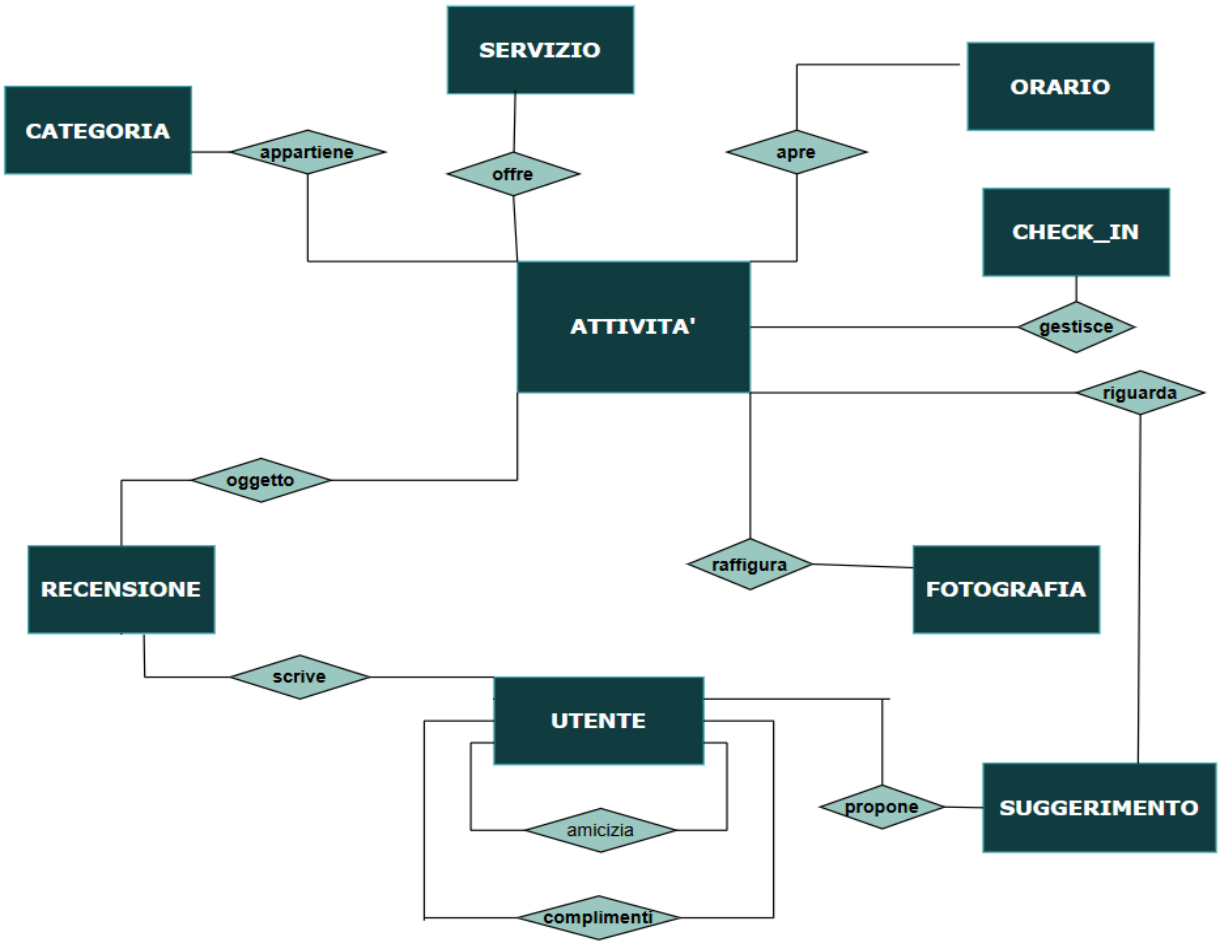
2.1 ENTITA' E RELAZIONI

L'entità principale del sistema è l'**Attività Commerciale**, caratterizzata da attributi descrittivi (nome, indirizzo, coordinate geografiche e valutazione) e **Orari** di apertura (considerati per ogni singolo giorno della settimana); essa appartiene ad una (o più) **Categoria** e offre ai propri clienti dei **Servizi** (come Wi-Fi o parcheggio), traducendo ciò in due relazioni molti-a-molti. Inoltre, essa può gestire più **Check-in** per il monitoraggio dell'affluenza.

L'**Utente**, identificato univocamente e caratterizzato da attributi descrittivi, interagisce con le attività attraverso le relazioni chiave **Recensioni** (che includono valutazione in stelle, testo, ecc....) e **Suggerimenti** rapidi.

Infine, il sistema gestisce contenuti multimediali tramite l'entità **Fotografia**, associata alle attività, e relazioni sociali riflessive tra gli utenti stessi per la gestione degli **Amici**, dei voti e dei **Complimenti** ricevuti.

A questo punto è possibile tracciare il seguente **schema portante**:

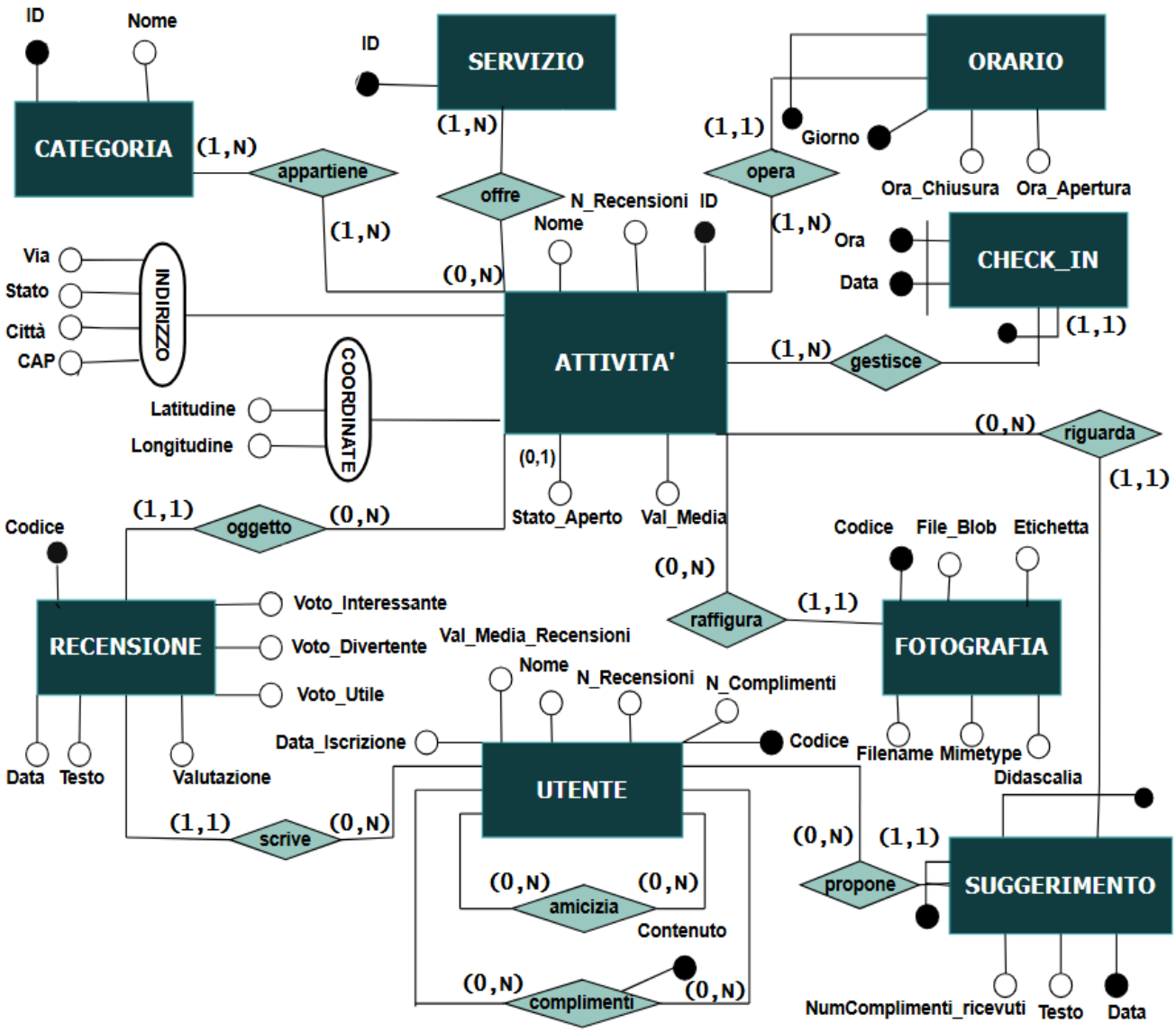


2.2 ASSOCIAZIONI E CARDINALITA'

Valgono le seguenti associazioni:

- Un'attività APPARTIENE ad una o più categorie (1, N) mentre una stessa categoria può essere specifica di una o più attività (1, N)
    - **ASSOCIAZIONE: MOLTI -A- MOLTI**
- Un'attività può OFFRIRE più servizi (o nessuno) (0, N) mentre un servizio può essere offerto da più attività (1, N)
    - **ASSOCIAZIONE: MOLTI -A- MOLTI**
- Un'attività può essere OGGETTO di più recensioni (o nessuna) (0, N), mentre una determinata recensione ha come oggetto una ed un'unica attività (1, 1)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Un'utente può SCRIVERE più recensioni (o nessuna) (0, N), mentre una determinata recensione è scritta da un unico utente (1, 1)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Una fotografia RAFFIGURA un'unica attività (1, 1) mentre un'attività può essere raffigurata in più fotografie (o nessuna) (0, N)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Un' attività può OPERARE in più orari diversi (nei diversi giorni della settimana) (1, N) mentre un orario specifico è descrittivo di un'unica attività (1, 1)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Un'attività può GESTISCE uno o più check-in (1, N) mentre un singolo check-in è gestito da un'unica attività (1, 1)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Un suggerimento RIGUARDA un'unica attività (1, 1) mentre un'attività può essere (o meno) oggetto (riguardare) di più suggerimenti (0, N)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Un utente può PROPORRE zero o più suggerimenti (0, N) mentre un determinato suggerimento può essere proposto da un unico utente (1, 1)
    - **ASSOCIAZIONE: UNO -A- MOLTI**
- Un utente può avere zero o più rapporti di AMICIZIA, cioè amici a loro volta utenti
    - **ASSOCIAZIONE (ricorsiva): MOLTI -A- MOLTI**
- Un utente può ricevere zero o più COMPLIMENTI da altri utenti
    - **ASSOCIAZIONE (ricorsiva): MOLTI -A- MOLTI**

2.3 SCHEMA ER


ID
Nome
CATEGORIA
(1,N)
appartiene
(1,N)
ID
SERVIZIO
(1,N)
offre
(0,N)
N_Recensioni
Nome
ID
ORARIO
(1,1)
opera
(1,N)
Giorno
Ora_Chiusura
Ora_Apertura
Ora
Data
CHECK_IN
(1,1)
Via
Stato
Città
CAP
INDIRIZZO
ATTIVITA'
(1,N)
gestisce
Latitudine
Longitudine
COORDINATE
(0,N)
riguarda
(1,1)
(0,1)
Stato_Aperto
Val_Media
Codice
(1,1)
oggetto
(0,N)
(0,N)
raffigura
(1,1)
Codice
File_Blob
Etichetta
FOTOGRAFIA
RECENSIONE
Voto_Interessante
Voto_Divertente
Voto_Utile
Val_Media_Recensioni
Nome
N_Recensioni
N_Complimenti
Data_Iscrizione
Codice
Filename
Mimetype
Didascalia
Data
Testo
Valutazione
UTENTE
(1,1)
scrive
(0,N)
(0,N)
(1,1)
propone
SUGGERIMENTO
(0,N)
amicizia
(0,N)
Contenuto
(0,N)
complimenti
(0,N)
NumComplimenti_ricevuti
Testo
Data

3. PROGETTAZIONE LOGICA

La fase di Progettazione Logica rappresenta l'anello di congiunzione tra la realtà astratta descrittiva del modello concettuale e quella implementativa del database.

L'obiettivo di questa fase è stato quello di tradurre lo schema ER in un Modello Relazionale ottimizzato, capace di garantire l'efficienza delle operazioni di manipolazione dei dati e di eliminare potenziali ridondanze che potrebbero portare ad anomalie.

Nella costruzione del modello logico presentato di seguito, sono stati seguiti criteri rigorosi per assicurare la robustezza della base di dati:

- **Mapping delle Entità**: Ogni entità è stata trasformata in una tabella.
- **Entità Indipendenti**: Le entità principali come UTENTE, ATTIVITA, CATEGORIE e SERVIZI mantengono i propri attributi originali. Per ciascuna di esse è stata definita una chiave primaria univoca (Codice_Utente, ID_Attivita, ecc.) per garantire l'identificazione certa dei record.
- **Gestione delle Associazioni**:
    - **Relazioni 1: N** sono state tradotte nello schema logico aggiungendo alla relazione che traduce l'entità dal lato UNO eventuali attributi di associazione e l'identificatore dell'entità del lato MOLTI dell'associazione. Esiste un vincolo di integrità referenziale tra quest'ultimo e il corrispettivo attributo dell'entità da cui discende.
    - **Relazioni N: N** sono state tradotte in relazioni nello schema logico, creando dunque tabelle di associazione dedicate, aventi come chiave primaria l'insieme degli identificatori delle entità che la compongono, le quali hanno un vincolo di integrità referenziale con il corrispondente attributo dell'entità da cui discendono.
    - **Associazioni Ricorsive:** Le relazioni di **AMICIZIA** e **COMPLIMENTI** tra utenti sono state tradotte in tabelle dove entrambi i campi puntano alla stessa tabella di origine (**UTENTI**)
- **Normalizzazione**: Lo schema è stato raffinato per soddisfare la Terza Forma Normale (3NF). Ogni attributo non chiave dipende in maniera funzionale, diretta e completa dalla chiave primaria, minimizzando il rischio di inconsistenze e ridondanza dei dati.

3.1 SCHEMA LOGICO FINALE

Di seguito viene riportata la struttura analitica delle tabelle.

Le Chiavi Primarie (PK) definiscono l'unicità di ogni record, mentre le Chiavi Esterne (FK) stabiliscono i vincoli di integrità tra le diverse entità del sistema.

UTENTI (Codice_Utente, Nome, Data_Iscrizione, Valutazione_Media_Recensioni, Num_Recensioni, Num_Complimenti)

• Chiave Primaria: Codice_Utente

ATTIVITA (ID_Attivita, Nome, Valutazione_Media, Numero_Recensioni, Via, Stato, Citta, CAP, Latitudine, Longitudine, Stato_Aperto)

• Chiave Primaria: ID_Attivita

RECENSIONI (Codice_Recensione, Utente, Attività, Valutazione, Data, Testo, Voti_Utili, Voti_Divertenti, Voti_Interessanti)

• Chiave Primaria: Codice_Recensione

• Chiave Esterna: Utente riferisce UTENTI(Codice_Utente)

• Chiave Esterna: Attività riferisce ATTIVITA(ID_Attivita)

FOTOGRAFIE (Codice_Fotografia, Attività, Etichetta, Didascalia, Mimetype, Filename, Foto_Blob)

• Chiave Primaria: Codice_Fotografia

• Chiave Esterna: Attività riferisce ATTIVITA(ID_Attivita)

CHECK_IN (Data, Attività)

• Chiave Primaria: (Data, Ora, Attività)

• Chiave Esterna: Attività riferisce ATTIVITA(ID_Attivita)

CATEGORIE (ID_Categoria, Nome)

• Chiave Primaria: ID_Categoria

SERVIZI (ID_Servizio, Nome)

• Chiave Primaria: ID_Servizio

SUGGERIMENTI (Utente, Attività, Data_Pubblicazione, Num_Complimenti_Ricevuti, Testo)

• Chiave Primaria: (Utente, Attività, Data_Pubblicazione)

• Chiave Esterna: Utente riferisce UTENTI(Codice_Utente)

• Chiave Esterna: Attivita riferisce ATTIVITA(ID_Attivita)

ORARI (Attivita, Giorno, Ora_Apertura, Ora_Chiusura)

• Chiave Primaria: (Attivita, Giorno)

• Chiave Esterna: Attivita riferisce ATTIVITA(ID_Attivita)

TABELLE DI ASSOCIAZIONE (RELAZIONI N: N)

APPARTIENE_A_CATEGORIA (ID_Attivita, ID_Categoria)

• Chiave Primaria: (ID_Attivita, ID_Categoria)

• Chiave Esterna: ID_Attivita riferisce ATTIVITA(ID_Attivita)

• Chiave Esterna: ID_Categoria riferisce CATEGORIE(ID_Categoria)

OFFRE_SERVIZIO (ID_Attivita, ID_Servizio)

• Chiave Primaria: (ID_Attivita, ID_Servizio)

• Chiave Esterna: ID_Attivita riferisce ATTIVITA(ID_Attivita)

• Chiave Esterna: ID_Servizio riferisce SERVIZI(ID_Servizio)

RELAZIONE_AMICIZIA (Codice_Utente1, Codice_Utente2)

• Chiave Primaria: (Codice_Utente1, Codice_Utente2)

• Chiave Esterna: Codice_Utente1 riferisce UTENTI(Codice_Utente)

• Chiave Esterna: Codice_Utente2 riferisce UTENTI(Codice_Utente)

COMPLIMENTI (Utente1, Utente2, Contenuto)

• Chiave Primaria: (Utente1, Utente2, Contenuto)

• Chiave Esterna: Utente1 riferisce UTENTI(Codice_Utente)

• Chiave Esterna: Utente2 riferisce UTENTI(Codice_Utente)

4. PROGETTAZIONE FISICA

La fase di Progettazione Fisica costituisce l'ultimo stadio del ciclo di vita della base di dati.

Mentre la progettazione logica si occupa di definire "cosa" memorizzare in modo indipendente dal software, la progettazione fisica si concentra sul "come" i dati verranno effettivamente organizzati e gestiti sul DBMS Oracle; in questa fase le strutture astratte definite nel modello logico vengono trasformate in oggetti fisici ottimizzati per l'ambiente di destinazione, garantendo efficienza e corretta gestione delle risorse di memoria, spostando l'attenzione dall'organizzazione concettuale all'efficienza implementativa.

Vengono selezionati i tipi di dato nativi (domini), definiti i vincoli di integrità referenziale e pianificate le strategie di memorizzazione. L'obiettivo primario è la creazione di uno schema che non solo preservi la coerenza dei dati, ma che massimizzi le prestazioni in fase di interrogazione e aggiornamento, riducendo al minimo i colli di bottiglia computazionali.

La progettazione fisica è divisa in più fasi:

- **Dimensionamento della base di dati**: Stima del volume di memoria occupato dalle tabelle in base alla dimensione dei record e al numero di occorrenze previste.
- **Creazione della base di dati**: Configurazione dell'ambiente fisico e degli spazi di archiviazione destinati a ospitare i dati.
- **Definizione delle politiche di sicurezza e creazione di utenti e ruoli**: Gestione degli accessi tramite la creazione di account e l'assegnazione di privilegi specifici per la protezione dei dati.
- **Creazione degli oggetti della base di dati**: Traduzione dello schema logico in oggetti fisici tramite istruzioni DDL, selezionando i tipi di dato nativi del DBMS.
- **Definizione dei vincoli**: Implementazione dei vincoli di integrità (PK, FK, Check) per garantire la coerenza automatica della base di dati.

- **Creazione di un'istanza (Popolamento)**: Inserimento dei dati iniziali nel database per testare la struttura e la validità dei vincoli implementati.

4.1 DIMENSIONAMENTO

Il dimensionamento fisico rappresenta l'analisi quantitativa volta a determinare l'impatto della base di dati sulle risorse hardware (memoria secondaria). Per ciascuna entità del sistema, è stata condotta una stima meticolosa dell'occupazione di memoria basata sulla lunghezza media dei record (Record Size), calcolata come somma algebrica dei byte allocati per ogni attributo. Attraverso la proiezione delle cardinalità previste a regime, è possibile derivare il volume di archiviazione complessivo mediante la formula:

Volume Totale = (Dimensione Media Record in Byte) × (Numero di Occorrenze Stimate)

Tale stima tiene conto della gestione specifica dei tipi di dato Oracle (come i 7 byte per i campi cronologici e i 3 byte per le medie numeriche) e dell'allocazione esterna per i contenuti testuali massivi gestiti tramite puntatori CLOB.

UTENTI

ATTRIBUTO	TIPO	BYTE
Codice_Utente	VARCHAR2(50)	50
Nome	VARCHAR2(100)	100
Data_Iscrizione	TIMESTAMP	7
Valutazione_Media_Recensioni	NUMBER (3,2)	3
Num_Recensioni	INTEGER	4
Num_Complimenti	INTEGER	4

$$D_{utenti} = \left(D_{codice_{utente}} + D_{nome} + D_{data_{iscrizione}} + D_{valutazione_{media}} + D_{num_{recensioni}} + D_{num_{complimenti}}\right) \cdot N_{utenti} = (50 + 100 + 7 + 3 + 4 + 4) \cdot N_{utenti} = 168 \cdot N_{utenti}$$

ATTIVITA'

ATTRIBUTO	TIPO	BYTE
Id_Attivita	VARCHAR2(50)	50
Nome	VARCHAR2(150)	150
Valutazione_Media	NUMBER (3,2)	3
Num_Recensioni	INTEGER	4
Via	VARCHAR2(200)	200
Città	VARCHAR2(100)	100
Stato	VARCHAR2(50)	50
Cap	VARCHAR2(20)	20
Latitudine	NUMBER	7
Longitudine	NUMBER	7
Stato_Aperto	VARCHAR2(10)	10

$$D_{attività} = (D_{id\_att} + D_{nome} + D_{val\_media} + D_{num\_rec} + D_{via} + D_{città} + D_{stato} + D_{cap} + D_{lat} + D_{long} + D_{stato\_ap}) \cdot N_{attivita}$$
$$= (50 + 150 + 3 + 4 + 200 + 100 + 50 + 20 + 7 + 7 + 10) \cdot N_{attivita}$$
$$= 601\ byte \cdot N_{attivita}$$

RECENSIONI

ATTRIBUTO	TIPO	BYTE
Codice_Recensione	VARCHAR2(50)	50
Valutazione	NUMBER	2
Data	TIMESTAMP	7
Testo	CLOB	4 (puntatore al testo)
Utente	VARCHAR2(50)	50
Attività	VARCHAR2(50)	50

$$D_{recensioni} = (D_{codice_{rec}} + D_{valutazione} + D_{data} + D_{testo} + D_{utente} + D_{attività}) \cdot N_{recensioni}$$
$$= (50 + 2 + 7 + 4 + 50 + 50) \cdot N_{recensioni} = 163\ byte \cdot N_{recensioni}$$

FOTOGRAFIE

ATTRIBUTO	TIPO	BYTE
Codice_Fotografia	VARCHAR2(50)	50
Etichetta	VARCHAR2(50)	50
Didascalia	CLOB	4 (puntatore al testo)
Attività	VARCHAR2(50)	50
Mimetype	VARCHAR2(255)	255
Filename	VARCHAR2(255)	255
Foto_Blob	BLOB	4 (puntatore al file binario)

$$D_{fotografie} = (D_{codice_{foto}} + D_{etichetta} + D_{didascalia} + D_{attività}) \cdot N_{fotografie}$$
$$= (50 + 50 + 4 + 50 + 255 + 255 + 4) \cdot N_{fotografie} = 668\ byte \cdot N_{fotografie}$$

CHECK-IN

ATTRIBUTO	TIPO	BYTE
Data	TIMESTAMP	7
Attività	VARCHAR2(50)	50

$$D_{check_{in}} = (D_{data} + D_{attività}) \cdot N_{check_{in}} = (7 + 50) \cdot N_{check_{in}} = 57\ byte \cdot N_{check_{in}}$$

CATEGORIE

ATTRIBUTO	TIPO	BYTE
Id_Categoria	VARCHAR2(100)	100
Nome	VARCHAR2(100)	100

$$D_{categorie} = (D_{id_{cat}} + D_{nome}) \cdot N_{categorie} = (100 + 100) \cdot N_{categorie} = 200\ byte \cdot N_{categorie}$$

SERVIZI

ATTRIBUTO	TIPO	BYTE
Id_Servizio	V ARCHAR2(200)	200

$$D_{servizi} = \left(D_{id_{serv}}\right) \cdot N_{servizi} = (200) \cdot N_{servizi} = 150\ byte \cdot N_{servizi}$$

SUGGERIMENTI

ATTRIBUTO	TIPO	BYTE
Data_Pubblicazione	TIMESTAMP	7
Num_Complimenti_Ricevuti	INTEGER	4
Testo	CLOB	4 (puntatore al testo)
Utente	VARCHAR2(50)	50
Attività	VARCHAR2(50)	50

$$D_{suggerimenti} = \left(D_{data_{pubbl}} + D_{complimenti} + D_{testo} + D_{utente} + D_{attività}\right) \cdot N_{suggerimenti} = (7 + 4 + 4 + 50 + 50) \cdot N_{suggerimenti} = 115\ byte \cdot N_{suggerimenti}$$

ORARI

ATTRIBUTO	TIPO	BYTE
Giorno	VARCHAR2(15)	15
Ora_Apertura	TIMESTAMP	7
Ora_Chiusura	TIMESTAMP	7
Attività	VARCHAR2(50)	50

$$D_{orari} = \left(D_{giorno} + D_{apertura} + D_{chiusura} + D_{attività}\right) \cdot N_{orari} = (15 + 7 + 7 + 50) \cdot N_{orari} = 79\ byte \cdot N_{orari}$$

COMPLIMENTI

ATTRIBUTO	TIPO	BYTE
Contenuto	VARCHAR2(50)	50
Utente1	VARCHAR2(50)	50
Utente2	VARCHAR2(50)	50

$$D_{complimenti} = (D_{contenuto} + D_{utente1} + D_{utente2}) \cdot N_{complimenti} = (50 + 50 + 50) \cdot N_{complimenti} = 150\ byte \cdot N_{complimenti}$$

APPARTIENE_A_CATEGORIA

ATTRIBUTO	TIPO	BYTE
Id_Attivita	VARCHAR2(50)	50
Id_Categoria	VARCHAR2(100)	100

$$D_{app_{cat}} = \left(D_{id_{att}} + D_{id_{cat}}\right) \cdot N_{app_{cat}} = (50 + 100) \cdot N_{app_{cat}} = 150\ byte \cdot N_{app_{cat}}$$

OFFRE_SERVIZIO

ATTRIBUTO	TIPO	BYTE
Id_Attivita	VARCHAR2(50)	50
Id_Servizio	VARCHAR2(50)	50

$$D_{offre_{serv}} = (D_{id_{att}} + D_{id_{serv}}) \cdot N_{offre_{serv}} = (50 + 50) \cdot N_{offre_{serv}} = 100\ byte \cdot N_{offre_{serv}}$$

RELAZIONE_AMICIZIA

ATTRIBUTO	TIPO	BYTE
Codice_Utente1	VARCHAR2(50)	50
Codice_Utente2	VARCHAR2(50)	50

$$D_{amicizia} = (D_{utente1} + D_{utente2}) \cdot N_{amicizia} = (50 + 50) \cdot N_{amicizia} = 100\ byte \cdot N_{amicizia}$$

4.2 CREAZIONE DELLE TABELLE

```
-- 2. CREAZIONE TABELLE

-- Tabella: UTENTI
CREATE TABLE UTENTI (
    Codice_Utente VARCHAR2(50),
    Nome VARCHAR2(100),
    Data_Iscrizione DATE DEFAULT SYSDATE,
    Valutazione_Media_Recensioni NUMBER (3,2),
    Num_Recensioni INTEGER DEFAULT 0,
    Num_Complimenti INTEGER DEFAULT 0,

    CONSTRAINT PK_UTENTI PRIMARY KEY(Codice_Utente),
    CONSTRAINT Check_Valutazione CHECK (Valutazione_Media_Recensioni >= 0 AND Valutazione_Media_Recensioni <= 5)
);

-- Tabella: ATTIVITA
CREATE TABLE ATTIVITA (
    ID_Attivita VARCHAR2(50),
    Nome VARCHAR2(150) NOT NULL,
    Valutazione_Media NUMBER (3,2),
    Numero_Recensioni INTEGER DEFAULT 0,
    Via VARCHAR2(200),
    Stato VARCHAR2(50),
    Citta VARCHAR2(100) NOT NULL,
    CAP VARCHAR2(20),
    Latitudine NUMBER(9,6),
    Longitudine NUMBER(9,6),
    Stato_Aperto VARCHAR2(10) DEFAULT '1',

    CONSTRAINT PK_ATTIVITA PRIMARY KEY(ID_Attivita),
    CONSTRAINT Check_Stato CHECK (Stato_Aperto IN ('Aperto', 'Chiuso', '0', '1')),
    CONSTRAINT Check_Valutazione2 CHECK (Valutazione_Media >= 0 AND Valutazione_Media <= 5)
);
```

```
-- Tabella: RECENSIONI
CREATE TABLE RECENSIONI(
    Codice_Recensione VARCHAR2(50),
    Utente VARCHAR2(50) NOT NULL,
    Attivita VARCHAR2(50) NOT NULL,
    Valutazione INTEGER NOT NULL,
    Data DATE DEFAULT SYSDATE NOT NULL,
    Testo CLOB,

    Voti_Utili INTEGER DEFAULT 0,
    Voti_Divertenti INTEGER DEFAULT 0,
    Voti_Interessanti INTEGER DEFAULT 0,

    CONSTRAINT PK_RECENSIONI PRIMARY KEY(Codice_Recensione),
    CONSTRAINT Check_Valutazione3 CHECK (Valutazione >= 0 AND Valutazione <= 5)
);

-- Tabella: FOTOGRAFIE
CREATE TABLE FOTOGRAFIE(
    Codice_Fotografia VARCHAR2(50),
    Attivita VARCHAR2(50) NOT NULL,
    Etichetta VARCHAR2(50),
    Didascalia CLOB,
    Foto_Blob BLOB,
    Mimetype VARCHAR2(255 BYTE),
    Filename VARCHAR2(255 BYTE),

    CONSTRAINT PK_FOTOGRAFIE PRIMARY KEY(Codice_Fotografia)
);
```

```
-- Tabella: CHECK_IN
CREATE TABLE CHECK_IN(
    Data TIMESTAMP,
    Attivita VARCHAR2(50),

    CONSTRAINT PK_CHECK_IN PRIMARY KEY(Data, Attivita)
);

-- Tabella: CATEGORIE
CREATE TABLE CATEGORIE(
    ID_Categoria VARCHAR2(100),
    Nome VARCHAR2(100) NOT NULL,

    CONSTRAINT PK_CATEGORIE PRIMARY KEY(ID_Categoria)
);

-- Tabella: SERVIZI
CREATE TABLE SERVIZI(
    ID_Servizio VARCHAR2(200),

    CONSTRAINT PK_SERVIZI PRIMARY KEY(ID_Servizio)
);

-- Tabella: SUGGERIMENTI
CREATE TABLE SUGGERIMENTI (
    Data_Pubblicazione TIMESTAMP DEFAULT SYSDATE NOT NULL,
    Num_Complimenti_Ricevuti INTEGER DEFAULT 0,
    Testo CLOB,
    Utente VARCHAR2(50) NOT NULL,
    Attivita VARCHAR2(50) NOT NULL,

    CONSTRAINT PK_SUGGERIMENTI PRIMARY KEY(Utente, Attivita, Data_Pubblicazione)
);
```

```
-- Tabella: ORARI
CREATE TABLE ORARI (
    Attivita VARCHAR2(50),
    Giorno VARCHAR2(15),
    Ora_Apertura VARCHAR2(10),
    Ora_Chiusura VARCHAR2(10),

    CONSTRAINT PK_ORARI PRIMARY KEY(Attivita, Giorno),
    CONSTRAINT Check_Giorni CHECK (Giorno IN ('Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday', 'Sunday'))
);

-- Tabella Associazione: COMPLIMENTI
CREATE TABLE COMPLIMENTI (
    Utente1 VARCHAR2(50),
    Utente2 VARCHAR2(50),
    Contenuto VARCHAR2 (50),

    CONSTRAINT PK_COMPLIMENTI PRIMARY KEY (Utente1, Utente2, Contenuto)
);
```

```
-- Tabelle Associazione: APPARTIENE_A_CATEGORIA
CREATE TABLE APPARTIENE_A_CATEGORIA (
    ID_Attivita VARCHAR2(50),
    ID_Categoria VARCHAR2(100),

    CONSTRAINT PK_APPARTENENZA PRIMARY KEY(ID_Attivita, ID_Categoria)
);

-- Tabelle Associazione: OFFRE_SERVIZIO
CREATE TABLE OFFRE_SERVIZIO (
    ID_Attivita VARCHAR2(50),
    ID_Servizio VARCHAR2(200),

    CONSTRAINT PK_SERVIZIO PRIMARY KEY(ID_Attivita, ID_Servizio)
);

-- Tabella Associazione: RELAZIONE_AMICIZIA
CREATE TABLE RELAZIONE_AMICIZIA (
    Codice_Utente1 VARCHAR2(50),
    Codice_Utente2 VARCHAR2(50),

    CONSTRAINT PK_AMICIZIA PRIMARY KEY (Codice_Utente1, Codice_Utente2)
);
```

4.3 VINCOLI DI INTEGRITA' REFERENZIALE

I **Vincoli di Integrità referenziale** sono fondamentali nello sviluppo di un Database Relazionale; essi garantiscono l'integrità dei dati memorizzati tra due tabelle collegate tra loro. Si implementano attraverso l'utilizzo delle **Foreign Key**, dove un attributo di una tabella referenziante, referenzia un attributo Primary Key (quindi sicuramente non NULL) referenziato.

Tutte le Foreign Key sono state aggiunte in seguito alla creazione delle tabelle tramite dei comandi di **ALTER TABLE**, per una questione di pulizia dello script.

```
-- 3. VINCOLI INTEGRITA (Foreign Keys)

-- RECENSIONI
ALTER TABLE RECENSIONI ADD CONSTRAINT FK_REC_ATTIVITA FOREIGN KEY (Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;
ALTER TABLE RECENSIONI ADD CONSTRAINT FK_REC_UTENTE FOREIGN KEY (Utente)
REFERENCES UTENTI(Codice_Utente) ON DELETE CASCADE;

-- FOTOGRAFIE
ALTER TABLE FOTOGRAFIE ADD CONSTRAINT FK_FOTO_ATTIVITA FOREIGN KEY (Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;

-- CHECK_IN
ALTER TABLE CHECK_IN ADD CONSTRAINT FK_CHECK_ATTIVITA FOREIGN KEY (Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;

-- SUGGERIMENTI
ALTER TABLE SUGGERIMENTI ADD CONSTRAINT FK_SUGG_ATTIVITA FOREIGN KEY (Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;
ALTER TABLE SUGGERIMENTI ADD CONSTRAINT FK_SUGG_UTENTE FOREIGN KEY (Utente)
REFERENCES UTENTI(Codice_Utente) ON DELETE CASCADE;

-- ORARI
ALTER TABLE ORARI ADD CONSTRAINT FK_ORARI_ATTIVITA FOREIGN KEY (Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;
```

```
-- APPARTIENE_A_CATEGORIA
ALTER TABLE APPARTIENE_A_CATEGORIA ADD CONSTRAINT FK_APP_ATTIVITA FOREIGN KEY (ID_Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;
ALTER TABLE APPARTIENE_A_CATEGORIA ADD CONSTRAINT FK_APP_CATEGORIA FOREIGN KEY (ID_Categoria)
REFERENCES CATEGORIE(ID_Categoria) ON DELETE CASCADE;

-- OFFRE_SERVIZIO
ALTER TABLE OFFRE_SERVIZIO ADD CONSTRAINT FK_OFFERTA_ATTIVITA FOREIGN KEY (ID_Attivita)
REFERENCES ATTIVITA(ID_Attivita) ON DELETE CASCADE;
ALTER TABLE OFFRE_SERVIZIO ADD CONSTRAINT FK_OFFERTA_SERVIZIO FOREIGN KEY (ID_Servizio)
REFERENCES SERVIZI(ID_Servizio) ON DELETE CASCADE;

-- RELAZIONE_AMICIZIA
ALTER TABLE RELAZIONE_AMICIZIA ADD CONSTRAINT FK_AMICO_UTENTE1 FOREIGN KEY (Codice_Utente1)
REFERENCES UTENTI(Codice_Utente) ON DELETE CASCADE;
ALTER TABLE RELAZIONE_AMICIZIA ADD CONSTRAINT FK_AMICO_UTENTE2 FOREIGN KEY (Codice_Utente2)
REFERENCES UTENTI(Codice_Utente) ON DELETE CASCADE;

-- COMPLIMENTI
ALTER TABLE COMPLIMENTI ADD CONSTRAINT FK_COMPLIMENTI_UTENTE1 FOREIGN KEY (Utente1)
REFERENCES UTENTI(Codice_Utente) ON DELETE CASCADE;
ALTER TABLE COMPLIMENTI ADD CONSTRAINT FK_COMPLIMENTI_UTENTE2 FOREIGN KEY (Utente2)
REFERENCES UTENTI(Codice_Utente) ON DELETE CASCADE;
```

- È stata applicata per tutte le Foreign Keys una politica di **ON DELETE CASCADE** per le operazioni di Rimozione (DELETE) sulla tabella “Padre”, per garantire la coerenza tra i dati ed evitare dati “orfani”.

4.4 POLITICA DI SICUREZZA E CREAZIONE DEI RUOLI

Sono stati implementati tre diversi **Ruoli** applicativi per gestire gli accessi al Database:

- **ROLE_ADMIN**: il ruolo Admin possiede tutti i privilegi sulle tabelle (SELECT, INSERT, UPDATE, DELETE);
- **ROLE_MODERATOR**: il ruolo Moderatore possiede privilegi di SELECT, UPDATE, DELETE sulle tabelle RECENSIONI, SUGGERIMENTI, FOTOGRAFIE, e solo privilegi di SELECT su tutte le altre tabelle;
- **ROLE_USER**: il ruolo User possiede privilegi di INSERT, UPDATE, DELETE sulle tabelle RECENSIONI, SUGGERIMENTI, FOTOGRAFIE, RELAZIONE_AMICIZIA, COMPLIENTI, e solo operazioni di SELECT su tutte le altre tabelle.

- *RUOLO: ADMIN*

```
-- 4. CREAZIONE RUOLI E UTENTI FISICI

-- A. ADMIN
CREATE ROLE ROLE_ADMIN;

GRANT ALL PRIVILEGES ON UTENTI TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON ATTIVITA TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON CATEGORIE TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON SERVIZI TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON RECENSIONI TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON FOTOGRAFIE TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON CHECK_IN TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON SUGGERIMENTI TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON ORARI TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON COMPLIMENTI TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON RELAZIONE_AMICIZIA TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON OFFRE_SERVIZIO TO ROLE_ADMIN;
GRANT ALL PRIVILEGES ON APPARTIENE_A_CATEGORIA TO ROLE_ADMIN;

CREATE USER Admin_Progetto IDENTIFIED BY sono_admin_del_progetto;
GRANT CREATE SESSION TO Admin_Progetto;
GRANT UNLIMITED TABLESPACE TO Admin_Progetto;
GRANT ROLE_ADMIN TO Admin_Progetto;
```

- *RUOLO: MODERATORE*

```
-- B. MODERATORE
CREATE ROLE ROLE_MODERATOR;

GRANT SELECT ON UTENTI TO ROLE_MODERATOR;
GRANT SELECT ON ATTIVITA TO ROLE_MODERATOR;
GRANT SELECT ON CATEGORIE TO ROLE_MODERATOR;
GRANT SELECT ON SERVIZI TO ROLE_MODERATOR;
GRANT SELECT ON CHECK_IN TO ROLE_MODERATOR;
GRANT SELECT ON ORARI TO ROLE_MODERATOR;
GRANT SELECT ON COMPLIMENTI TO ROLE_MODERATOR;
GRANT SELECT ON RELAZIONE_AMICIZIA TO ROLE_MODERATOR;
GRANT SELECT ON OFFRE_SERVIZIO TO ROLE_MODERATOR;
GRANT SELECT ON APPARTIENE_A_CATEGORIA TO ROLE_MODERATOR;

GRANT SELECT, UPDATE, DELETE ON RECENSIONI TO ROLE_MODERATOR;
GRANT SELECT, UPDATE, DELETE ON SUGGERIMENTI TO ROLE_MODERATOR;
GRANT SELECT, UPDATE, DELETE ON FOTOGRAFIE TO ROLE_MODERATOR;

CREATE USER Moderator_Progetto IDENTIFIED BY sono_moderatore_del_progetto;
GRANT CREATE SESSION TO Moderator_Progetto;
GRANT UNLIMITED TABLESPACE TO Moderator_Progetto;
GRANT ROLE_MODERATOR TO Moderator_Progetto;
```

- *RUOLO: USER*

```
-- C. USER
CREATE ROLE ROLE_USER;

GRANT INSERT, UPDATE, DELETE ON RECENSIONI TO ROLE_USER;
GRANT INSERT, UPDATE, DELETE ON SUGGERIMENTI TO ROLE_USER;
GRANT INSERT, UPDATE, DELETE ON FOTOGRAFIE TO ROLE_USER;
GRANT INSERT, UPDATE, DELETE ON RELAZIONE_AMICIZIA TO ROLE_USER;
GRANT INSERT, UPDATE, DELETE ON COMPLIMENTI TO ROLE_USER;

GRANT SELECT ON UTENTI TO ROLE_USER;
GRANT SELECT ON ATTIVITA TO ROLE_USER;
GRANT SELECT ON CATEGORIE TO ROLE_USER;
GRANT SELECT ON SERVIZI TO ROLE_USER;
GRANT SELECT ON ORARI TO ROLE_USER;
GRANT SELECT ON OFFRE_SERVIZIO TO ROLE_USER;
GRANT SELECT ON APPARTIENE_A_CATEGORIA TO ROLE_USER;
GRANT SELECT ON RECENSIONI TO ROLE_USER;
GRANT SELECT ON SUGGERIMENTI TO ROLE_USER;
GRANT SELECT ON FOTOGRAFIE TO ROLE_USER;
GRANT SELECT ON CHECK_IN TO ROLE_USER;
GRANT SELECT ON RELAZIONE_AMICIZIA TO ROLE_USER;
GRANT SELECT ON COMPLIMENTI TO ROLE_USER;

CREATE USER User_Progetto IDENTIFIED BY sono_user_del_progetto;
GRANT CREATE SESSION TO User_Progetto;
GRANT UNLIMITED TABLESPACE TO User_Progetto;
GRANT ROLE_USER TO User_Progetto;
```

4.5 POPOLAMENTO BASE DATI

Per popolare il Database sono stati utilizzati tutti i dati forniti nel formato JSON; attraverso l'esecuzione di uno script python (*conversion_JSON_CSV.py*), i dati sono stati processati e convertiti in formato CSV, compatibile con le opzioni di inserimento su SQL Developer. Prima di essere inseriti nel progetto, i dati sono stati processati ulteriormente attraverso l'esecuzione di un altro script python (*filtering_CSV.py*) per l'estrazione degli attributi multi-valore per popolare in particolare le tabelle CATEGORIE, CHECK_IN, RELAZIONE_AMICIZIA, ORARI.

È stata inoltre effettuata un'attenta analisi dei domini degli attributi dei file CSV e confrontati con i domini definiti tramite i comandi DDL, per un corretto popolamento.

Successivamente sono stati utilizzati altri due script python (*extract_services.py* e *extract_offre_servizio.py*) che hanno permesso di estrarre dal file business .CSV l'attributo multi-valore "attributi" (che contengono tutti i servizi offerti da un'attività), generando un file .sql con tutte le operazioni di INSERT necessarie, in modo da popolare la tabella SERVIZI con tutti gli attributi distinti disponibili, e OFFRE_SERVIZIO, tutti servizi in corrispondenza di tutte le attività presenti.

Infine, tramite SQL Developer sono state popolate le tabelle con tutti i dati disponibili.

A seguito di una valutazione dei dati disponibili, abbiamo scelto di aggiungere degli attributi alla tabella FOTOGRAFIE (Foto_Blob, Filename, Mimetype) che permettono la memorizzazione nel Database delle immagini fornite (che hanno come intestazione il codice già registrato nel Database) direttamente in formato .jpg con una conversione in byte.

Ciò è stato possibile grazie ad un altro script python (importa_foto.py) che ha effettuato automaticamente un'operazione di INSERT, confrontando CODICE_FOTOGRAFIA con il percorso della Fotografia associata all'interno della cartella che le contiene. Nel Database avremmo solo una sequenza di byte nel campo Foto_Blob, la visualizzazione fisica dell'immagine sarà possibile nella applicazione web sviluppata **PrimaryKeyApp**.

5. QUERY

- ***Che cos'è una Query?***

Nel contesto di un database relazionale, una Query (o interrogazione) rappresenta lo strumento fondamentale attraverso il quale gli utenti o le applicazioni interagiscono con i dati memorizzati. Se il database è il "magazzino" delle informazioni, la query è la "richiesta specifica" rivolta al magazziniere per depositare, prelevare o organizzare la merce. Queste richieste sono formulate utilizzando l'SQL (Structured Query Language), il linguaggio standard per la gestione dei database relazionali.

- ***Perché sono necessarie?***

Un database, per quanto ricco di dati, è privo di valore se non possiede un meccanismo efficiente per recuperare e manipolare tali informazioni.

Le query sono necessarie per trasformare i dati grezzi (numeri, stringhe, date) in informazione utile per il processo decisionale.

- ***L'importanza nel Modello Relazionale***

In un database relazionale progettato correttamente, le informazioni sono "normalizzate", ovvero suddivise in molteplici tabelle (Utenti, Attività, Recensioni, Foto) per evitare ridondanze e garantire l'integrità dei dati. Le query svolgono qui un ruolo cruciale di collegamento: attraverso i meccanismi di JOIN, esse sono in grado di ricostruire le relazioni tra le entità, unendo dati provenienti da tabelle diverse in un'unica vista coerente.

Ad esempio, permettono di associare il testo di una recensione al nome dell'utente che l'ha scritta e all'insegna del locale a cui si riferisce, benché questi dati risiedano in tabelle separate.

- ***Applicazione nel Progetto***

Nel contesto specifico della piattaforma di recensioni e raccomandazione oggetto di questo progetto, le query non sono semplici comandi tecnici, ma la traduzione diretta delle funzionalità di business. Ogni azione descritta nella traccia — dalla ricerca di un ristorante per "valutazione media", al controllo dei permessi di un moderatore, fino all'analisi dei picchi di check-in — corrisponde a una specifica interrogazione SQL. La loro efficienza determina direttamente la reattività della piattaforma e la qualità dell'esperienza utente.

5.1 Individuazione del giorno di picco assoluto di affluenza (Anno 2021)

Questa interrogazione ha lo scopo di analizzare lo storico dei check-in registrati sulla piattaforma durante l'anno 2021.

L'obiettivo è identificare, per ogni singola attività commerciale, la data specifica in cui si è verificato il numero massimo di visite. Questo tipo di informazione è cruciale per l'analisi storica delle performance, permettendo ai gestori di individuare eventi o periodi stagionali di particolare successo.

```sql
-- 6. QUERY
-- Trovare per ogni attività, il giorno in cui c'è stato il maggior numero di check in nel 2021
-- Tramite Vista Materializzata (necessita di aggiornamento manuale in caso di nuovi check-in
CREATE MATERIALIZED VIEW GIORNI_CHECK_IN
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND AS
SELECT
    A.ID_Attivita,
    A.Nome AS Nome_Attivita,
    TRUNC(C.Data) AS Data_Check_In,
    COUNT(*) AS Num_Check_In
FROM
    ATTIVITA A
JOIN
    CHECK_IN C ON A.ID_Attivita = C.Attivita
WHERE
    C.Data >= TO_DATE('2021-01-01', 'YYYY-MM-DD')
    AND C.Data <= TO_DATE('2021-12-31', 'YYYY-MM-DD')
GROUP BY
    A.ID_Attivita,
    A.Nome,
    TRUNC(C.Data);

SELECT
    G1.Nome_Attivita,
    G1.Data_Check_In,
    G1.Num_Check_In
FROM
    GIORNI_CHECK_IN G1
WHERE
    G1.Num_Check_In = (

        SELECT MAX(G2.Num_Check_In)
        FROM GIORNI_CHECK_IN G2
        WHERE G2.ID_Attivita = G1.ID_Attivita
    )
ORDER BY
    G1.Num_Check_In DESC;
```

5.2 Classifica "Top Rated" per Città con Soglia di Affidabilità

Questa interrogazione ha lo scopo di generare delle classifiche di eccellenza locale ("Best of"), individuando le 5 migliori attività commerciali per ogni singola città registrata nella piattaforma.

A differenza di una semplice classifica basata sulla media voti pura, questa query introduce un **criterio di qualità e affidabilità**: vengono prese in considerazione esclusivamente le attività che hanno ricevuto almeno 50 recensioni. Questo filtro è fondamentale per evitare distorsioni statistiche (ad esempio, un'attività con un'unica recensione da 5 stelle non dovrebbe apparire prima di una con 4.8 stelle basata su centinaia di opinioni), garantendo all'utente risultati solidi e comprovati dalla community.

```sql
-- Trovare per ogni città, le 5 attività con la valutazione media più alta
--(decrescente) che abbiano almeno 50 recensioni

WITH Classifica_Citta AS (
    SELECT
        ID_Attivita,
        Nome,
        Citta,
        Valutazione_Media,
        Numero_Recensioni,
        RANK() OVER (
            PARTITION BY Citta
            ORDER BY Valutazione_Media DESC
        ) as Posizione
    FROM
        ATTIVITA
    WHERE
        Numero_Recensioni >= 50
)
SELECT
    Nome,
    Citta,
    Valutazione_Media,
    Numero_Recensioni
FROM
    Classifica_Citta
WHERE
    Posizione <= 5
ORDER BY
    Citta, Posizione;
```

5.3 Identificazione degli "Utenti Elite" e Influencer

Lo scopo è isolare, all'interno della vasta base di utenti registrati, i cosiddetti "Power Users" o "Elite". La query non premia solo la quantità (chi scrive molto), ma incrocia questo dato con la qualità e l'impatto sociale (chi riceve approvazione dagli altri). Vengono selezionati solo gli utenti che superano contemporaneamente due soglie critiche: un'alta produttività (> 100 recensioni) e una forte reputazione sociale (> 1000 complimenti ricevuti). Identificare questi profili permette all'azienda di inviare loro promozioni esclusive, badge speciali o inviti a eventi, incentivando ulteriormente la creazione di contenuti di qualità.

```sql
-- Trovare per ogni utente, quelli che hanno scritto più di 100 recensioni
-- i cui complimenti ricevuti (numero di complimenti totali all'utente) superano
-- 1000 (scopo identificare utenti elite)

SELECT
    Nome,
    Num_Recensioni,
    Num_Complimenti
FROM
    UTENTI
WHERE
    Num_Recensioni > 100
    AND Num_Complimenti > 1000
ORDER BY
    Num_Complimenti DESC;
```

5.4 Ricerca Mirata di Attività per Categoria e Qualità (Area Los Angeles)

Questa interrogazione simula il caso d'uso più frequente e centrale per l'utente finale della piattaforma: la ricerca puntuale di un servizio specifico. L'obiettivo non è mostrare tutte le attività genericamente, ma restringere il campo su tre dimensioni simultanee:

- **Tipologia:** Solo attività di ristorazione ("Ristoranti").
- **Geografia:** Solo situate nell'area di "Los Angeles".
- **Qualità:** Solo quelle che garantiscono un'esperienza eccellente (Valutazione media > 4).

```sql
-- Selezionare i ristoranti di Los Angeles (LA) con valutazione media > 4

SELECT
    A.Nome,
    A.Via,
    A.Citta,
    A.Valutazione_Media
FROM
    ATTIVITA A
JOIN APPARTIENE_A_CATEGORIA K ON K.ID_Attivita = A.ID_Attivita
JOIN CATEGORIE C ON K.ID_Categoria = C.ID_Categoria
WHERE
    A.Stato = 'LA'
    AND C.Nome = 'Restaurants'
    AND A.Valutazione_Media > 4
ORDER BY Valutazione_Media DESC;
```

5.5 Verifica disponibilità in tempo reale ("Aperto Ora")

Questa interrogazione abilita una funzionalità critica per l'esperienza utente in mobilità: filtrare le attività non solo in base a *cosa* sono o *dove* sono, ma in base alla loro effettiva **accessibilità nel momento presente**. L'obiettivo è confrontare l'istante esatto della richiesta dell'utente (giorno della settimana e orario corrente) con la tabella degli orari di apertura di ciascuna attività, restituendo solo

quelle attualmente operative. Questo evita la frustrazione dell'utente che si reca fisicamente in un luogo per trovarlo chiuso.

```sql
-- Trovare tutte le attività aperte ora (controllo diretto sull'orario attuale)

SELECT
    A.Nome,
    O.Ora_Apertura,
    O.Ora_Chiusura
FROM
    ATTIVITA A
JOIN
    ORARI O ON A.ID_Attivita = O.Attivita
WHERE
    -- 1. Controlla il giorno (Forziamo la lingua INGLESE/AMERICAN)
    TRIM(UPPER(O.Giorno)) = TRIM(UPPER(TO_CHAR(SYSDATE, 'DAY', 'NLS_DATE_LANGUAGE=AMERICAN')))

    AND

    -- 2. Controlla l'orario
    TO_CHAR(SYSDATE, 'HH24:MI') BETWEEN O.Ora_Apertura AND O.Ora_Chiusura;
```

5.6 Gallery Fotografica Tematica (Filtro "Food")

```sql
-- Trovare tutte le foto di un'attività che hanno etichetta food

SELECT
    A.Nome,
    F.Codice_Fotografia
FROM
    FOTOGRAFIE F
JOIN
    ATTIVITA A ON F.Attivita = A.ID_Attivita
WHERE
    F.Etichetta = 'food';
```

5.7 Profilo Utente a 360° (Contenuti e Network Sociale)

Questa interrogazione ha lo scopo di ricostruire l'intera "impronta digitale" di uno specifico utente all'interno della piattaforma (identificato nel caso in esame dal codice univoco - `NycZLw5rPxqwrkKKI-83w`). L'obiettivo è aggregare in un'unica vista o report tutte le interazioni principali del soggetto:

- **Contributi Critici:** Le recensioni dettagliate scritte per le attività.
- **Contributi Rapidi:** I suggerimenti (tips) brevi lasciati al volo.
- **Network Sociale:** La lista delle connessioni di amicizia stabilite con altri utenti. Questa vista è fondamentale sia per la pagina "Il mio Profilo" (lato utente), sia per permettere agli amministratori di valutare l'influenza e il comportamento complessivo di un iscritto.

```sql
-- Estrarre la lista di tutti i contenuti di un utente specifico (codice -NycZLw5rPxqwrkKKI-83w)
-- (suggerimenti, recensioni, lista di amici)

SELECT
    'Recensione' AS Tipo,
    TO_CHAR(SUBSTR(Testo, 1, 2000)) AS Contenuto,
    TO_CHAR(Data, 'DD/MM/YYYY') AS Data
FROM RECENSIONI
WHERE Utente = '-NycZLw5rPxqwrkKKI-83w'

UNION ALL

SELECT 'Suggerimento', TO_CHAR(SUBSTR(Testo, 1, 2000)), TO_CHAR(Data_Pubblicazione, 'DD/MM/YYYY')
FROM SUGGERIMENTI
WHERE Utente = '-NycZLw5rPxqwrkKKI-83w'

UNION ALL

SELECT 'Amico', CAST(U.Nome AS VARCHAR2(2000)), NULL
FROM RELAZIONE_AMICIZIA A
JOIN UTENTI U ON A.Codice_Utente2 = U.Codice_Utente
WHERE A.Codice_Utente1 = '-NycZLw5rPxqwrkKKI-83w';
```

6. TRIGGER

- ***Che cos'è un Trigger?***

Un **Trigger** (letteralmente "grilletto" o "innesco") è una particolare tipologia di stored procedure che viene eseguita **automaticamente** dal Database Management System (DBMS) in risposta a un determinato evento che avviene su una tabella. A differenza delle query standard, che devono essere richiamate esplicitamente dall'utente o dall'applicazione, i trigger rimangono "in ascolto" e si attivano da soli quando vengono modificati i dati, specificamente durante le operazioni di **INSERT**, **UPDATE** o **DELETE**.

- ***A cosa servono e come sono impiegati?***

I trigger sono utilizzati per implementare la cosiddetta logica attiva all'interno del database. Il loro impiego principale si suddivide in tre categorie:

- **Mantenimento della Consistenza dei Dati (Data Integrity)**: Servono per imporre regole complesse che i semplici vincoli (come NOT NULL o FOREIGN KEY) non possono gestire. Ad esempio, impedire l'inserimento di una recensione se la data è futura rispetto a quella odierna.
- **Calcolo di Dati Derivati (Automazione)**: Sono fondamentali per aggiornare automaticamente campi riassuntivi.

- Nel nostro progetto, ad esempio, ogni volta che viene aggiunta una nuova recensione, un trigger può ricalcolare istantaneamente la media stelle dell'attività e incrementare il contatore delle recensioni totali, senza che l'applicazione debba fare calcoli manuali.

- **Auditing e Storicizzazione**: Vengono impiegati per tracciare le modifiche. Se un utente modifica una recensione, un trigger può salvare la vecchia versione in una tabella di "storico" (Audit Log) per scopi di sicurezza e moderazione.

6.1 Sincronizzazione Automatica Metriche Attività (Feedback Loop)

Ecco la presentazione per il primo Trigger. Questa è probabilmente l'automazione più importante di tutto il database, perché garantisce che il voto medio che l'utente vede sia sempre aggiornato senza rallentare il caricamento della pagina.

Obiettivo: Questo trigger ha il compito di mantenere costantemente aggiornato il "biglietto da visita" di ogni attività commerciale. In un sistema di recensioni, due delle informazioni più critiche per l'utente sono il numero totale di recensioni (che indica la popolarità) e la valutazione media in stelle (che indica la qualità). L'obiettivo di questo trigger è far sì che, nell'istante esatto in cui un utente pubblica una nuova recensione, il profilo dell'attività target recepisca immediatamente il cambiamento, incrementando il contatore e ricalcolando la media voti, senza attendere elaborazioni notturne o interventi manuali.

```
-- 8. TRIGGER

-- 1.     Trigger per Aggiornare le Statistiche dell'Attività (Media e Conteggio)

CREATE OR REPLACE TRIGGER TRG_UPDATE_STATS_ATTIVITA
AFTER INSERT OR UPDATE OR DELETE ON RECENSIONI
FOR EACH ROW
BEGIN
   -- Caso INSERIMENTO di una nuova recensione
   IF INSERTING THEN
       UPDATE ATTIVITA
       SET
           -- Incrementiamo il contatore (usiamo NVL per sicurezza)
           Numero_Recensioni = NVL(Numero_Recensioni, 0) + 1,

           -- Ricalcoliamo la media ponderata
           Valutazione_Media = (
               (NVL(Valutazione_Media, 0) * NVL(Numero_Recensioni, 0)) + :NEW.Valutazione
           ) / (NVL(Numero_Recensioni, 0) + 1)
       WHERE ID_Attivita = :NEW.Attivita;

   -- Caso CANCELLAZIONE di una recensione
   ELSIF DELETING THEN
       UPDATE ATTIVITA
       SET
           Numero_Recensioni = GREATEST(NVL(Numero_Recensioni, 0) - 1, 0), -- Evitiamo numeri negativi

           Valutazione_Media = CASE
               -- Se stiamo cancellando l'unica recensione rimasta, la media torna a 0
               WHEN (NVL(Numero_Recensioni, 0) - 1) <= 0 THEN 0
               ELSE (
                   (NVL(Valutazione_Media, 0) * NVL(Numero_Recensioni, 0)) - :OLD.Valutazione
               ) / (NVL(Numero_Recensioni, 0) - 1)
           END
       WHERE ID_Attivita = :OLD.Attivita;

   -- Caso AGGIORNAMENTO del voto
   -- Scatta solo se cambia la colonna Valutazione o cambia l'attività (spostamento recensione)
   ELSIF UPDATING('Valutazione') THEN
       UPDATE ATTIVITA
       SET Valutazione_Media = (
           (NVL(Valutazione_Media, 0) * NVL(Numero_Recensioni, 0)) - :OLD.Valutazione + :NEW.Valutazione
       ) / NULLIF(Numero_Recensioni, 0) -- NULLIF evita divisioni per zero se i dati fossero corrotti
       WHERE ID_Attivita = :NEW.Attivita;
   END IF;
END;
/
```

6.2 Aggiornamento Real-time del Profilo Utente (Metriche di Partecipazione)

Questo automatismo è essenziale per la gestione del sistema di reputazione della piattaforma. Come specificato nei requisiti, gli utenti sono caratterizzati dal numero di recensioni scritte, un dato fondamentale per determinare il loro status (es. Utente "Elite", "Principiante", ecc.) e l'autorevolezza del loro profilo agli occhi della community.

Obiettivo: garantire che ogni volta che un utente pubblica un nuovo contributo, il contatore nel suo profilo personale venga incrementato istantaneamente. Questo feedback immediato è cruciale per la

gratificazione dell'utente (gamification) e assicura che le query di ricerca degli "Influencer" (vedi Query 3) operino sempre su dati aggiornati al secondo.

```
-- 2.     Trigger per Aggiornare le Statistiche dell'Utente (Recensioni scritte)

CREATE OR REPLACE TRIGGER TRG_UPDATE_STATS_UTENTE
AFTER INSERT OR UPDATE OR DELETE ON RECENSIONI
FOR EACH ROW
BEGIN
    -- Caso INSERIMENTO
    IF INSERTING THEN
        UPDATE UTENTI
        SET
            Num_Recensioni = NVL(Num_Recensioni, 0) + 1,

            -- Formula: (MediaVecchia * NumVecchio + NuovoVoto) / (NumVecchio + 1)
            Valutazione_Media_Recensioni = (
                (NVL(Valutazione_Media_Recensioni, 0) * NVL(Num_Recensioni, 0)) + :NEW.Valutazione
            ) / (NVL(Num_Recensioni, 0) + 1)
        WHERE Codice_Utente = :NEW.Utente;

    -- Caso CANCELLAZIONE
    ELSIF DELETING THEN
        UPDATE UTENTI
        SET
            -- Usiamo GREATEST per evitare che il contatore vada sotto zero per errore
            Num_Recensioni = GREATEST(NVL(Num_Recensioni, 0) - 1, 0),

            Valutazione_Media_Recensioni = CASE
                -- Se il numero di recensioni sta per diventare 0 (o lo è già), la media torna a 0
                WHEN (NVL(Num_Recensioni, 0) - 1) <= 0 THEN 0
                ELSE (
                    (NVL(Valutazione_Media_Recensioni, 0) * NVL(Num_Recensioni, 0)) - :OLD.Valutazione
                ) / (NVL(Num_Recensioni, 0) - 1)
            END
        WHERE Codice_Utente = :OLD.Utente;

    -- Caso AGGIORNAMENTO (Solo se cambia il voto)
    ELSIF UPDATING ('Valutazione') THEN
        UPDATE UTENTI
        SET
            Valutazione_Media_Recensioni = (
                (NVL(Valutazione_Media_Recensioni, 0) * NVL(Num_Recensioni, 0)) - :OLD.Valutazione + :NEW.Valutazione
            ) / NULLIF(NVL(Num_Recensioni, 0), 0) -- NULLIF protegge dalla divisione per zero
        WHERE Codice_Utente = :NEW.Utente;
    END IF;
END;
/
```

6.3 Aggiornare i complimenti ricevuti

```
-- 3.     Trigger per Aggiornare i Complimenti Ricevuti

CREATE OR REPLACE TRIGGER TRG_UPDATE_COMPLIMENTI
AFTER INSERT OR DELETE ON COMPLIMENTI
FOR EACH ROW
BEGIN
    -- Se viene fatto un complimento (INSERT)
    IF INSERTING THEN
        UPDATE UTENTI
        SET Num_Complimenti = NVL(Num_Complimenti, 0) + 1
        WHERE Codice_Utente = :NEW.Utente2; -- Aggiorniamo il ricevente

    -- Se viene rimosso un complimento (DELETE)
    ELSIF DELETING THEN
        UPDATE UTENTI
        SET Num_Complimenti = GREATEST(NVL(Num_Complimenti, 0) - 1, 0) -- Evitiamo numeri negativi
        WHERE Codice_Utente = :OLD.Utente2;
    END IF;
END;
/
```

6.4 Salvaguardia dell'Unicità (Prevenzione Recensioni Multiple)

L'esempio successivo è fondamentale perché illustra un uso diverso dei trigger: non per aggiornare dati (come i precedenti), ma per bloccare un'operazione non valida (Integrità).

Questo automatismo funge da "guardiano" per la qualità e la correttezza etica della piattaforma. Una regola di business fondamentale nei sistemi di recensione è il principio "una persona, un voto" per ogni singola attività. Se si permettesse allo stesso utente di recensire il medesimo ristorante più volte, si aprirebbe la porta a comportamenti scorretti come lo spam, il review bombing (inondare un'attività di voti negativi) o l'auto-promozione fraudolenta.

Obiettivo: impedire fisicamente che ciò accada: se l'utente X prova a inserire una seconda recensione per l'attività Y, il sistema deve rifiutare l'operazione e restituire un errore.

```
-- 4.     Trigger di Integrità: Prevenire Doppie Recensioni

CREATE OR REPLACE TRIGGER TRG_CHECK_DUPLICATE_REVIEW
BEFORE INSERT ON RECENSIONI
FOR EACH ROW
DECLARE
    v_count INTEGER;
BEGIN
    -- Controlla se esiste già una recensione per questa coppia Utente/Attività
    SELECT COUNT(*)
    INTO v_count
    FROM RECENSIONI
    WHERE Utente = :NEW.Utente
      AND Attivita = :NEW.Attivita;

    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Errore: Questo utente ha già recensito questa attività.');
    END IF;
END;
/
```

6.5 Trigger di integrità per Categorie

Questo automatismo serve a mantenere pulita e coerente la classificazione delle attività commerciali all'interno della piattaforma. In un sistema complesso, è fondamentale che le categorie (es. "Ristorante", "Hotel", "Bar") appartengano a una lista controllata e standardizzata. Se si permettesse l'inserimento libero di categorie testuali, il database si riempirebbe rapidamente di duplicati, errori di battitura o varianti non necessarie (es. "Pizzeria", "pizzerie", "Pizza Place"), rendendo impossibili le ricerche filtrate o le analisi statistiche.

Obiettivo: verificare, all'atto dell'inserimento o della modifica di un'attività, che le categorie associate esistano effettivamente nell'elenco ufficiale del sistema.

```
-- 5.     Trigger per gestione automatica delle Categorie (verifica se esiste la categoria inserita per una nuova attività)

CREATE OR REPLACE TRIGGER TRG_AUTO_INSERT_CATEGORIA
BEFORE INSERT ON APPARTIENE_A_CATEGORIA
FOR EACH ROW
DECLARE
    v_esiste INTEGER;
BEGIN
    -- Controlla se la categoria esiste già nella tabella madre
    SELECT COUNT(*) INTO v_esiste
    FROM CATEGORIE
    WHERE ID_Categoria = :NEW.ID_Categoria;

    -- Se non esiste (count = 0), inseriRE prima di proseguire
    IF v_esiste = 0 THEN
        -- Dobbiamo inserire anche il campo 'Nome' perché è NOT NULL nel database.
        -- Usiamo lo stesso valore dell'ID come Nome di default.
        INSERT INTO CATEGORIE (ID_Categoria, Nome)
        VALUES (:NEW.ID_Categoria, :NEW.ID_Categoria);

        DBMS_OUTPUT.PUT_LINE('Nuova categoria creata automaticamente: ' || :NEW.ID_Categoria);
    END IF;
END;
/
```

6.6 Garanzia di Coerenza Temporale (Validazione Check-in)

Questo automatismo agisce come filtro logico per preservare la qualità e il realismo dei dati raccolti. Il "Check-in" rappresenta, per definizione, la conferma di una presenza fisica avvenuta presso un'attività. Concettualmente, è impossibile attestare di "essere stati" in un luogo in una data futura (es. "Domani alle 15:00 sono stato qui").
Obiettivo: intercettare errori di inserimento (es. utente che sbaglia a selezionare l'anno nel calendario) o tentativi di manipolazione dati, impedendo che vengano registrati eventi cronologicamente impossibili che inquinerebbero le analisi statistiche sui flussi di visita.

```
-- 6.     Trigger di Coerenza Temporale (Check-in)

CREATE OR REPLACE TRIGGER TRG_CHECKIN_FUTURE
BEFORE INSERT OR UPDATE ON CHECK_IN
FOR EACH ROW
BEGIN
    -- Se la data del check-in è nel futuro rispetto all'orologio del server
    IF :NEW.Data > SYSTIMESTAMP THEN
        RAISE_APPLICATION_ERROR(-20003, 'Errore: Non è possibile effettuare un check-in nel futuro.');
    END IF;
END;
/
```

6.7 Integrità Sociale (Prevenzione Auto-interazioni)

Obiettivo: prevenire atteggiamenti che potrebbero danneggiare la qualità della piattaforma. Infatti, di base un utente non può stringere amicizia con sé stesso, sarebbe un assurdo, e non può nemmeno farsi dei complimenti da solo, altrimenti andrebbe a “gonfiare” il proprio profilo in modo illecito.

```
-- 7.     Trigger Sociale (Prevenzione auto-interazione)

CREATE OR REPLACE TRIGGER TRG_NO_SELF_INTERACTION
BEFORE INSERT OR UPDATE ON RELAZIONE_AMICIZIA
FOR EACH ROW
BEGIN
    -- Controllo se l'utente sta provando a stringere amicizia con se stesso
    IF :NEW.Codice_Utente1 = :NEW.Codice_Utente2 THEN
        RAISE_APPLICATION_ERROR(-20005, 'Errore: Non puoi stringere amicizia con te stesso.');
    END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_NO_SELF_COMPLIMENT
BEFORE INSERT OR UPDATE ON COMPLIMENTI
FOR EACH ROW
BEGIN
    -- Controllo se l'utente sta provando a farsi un complimento da solo
    IF :NEW.Utente1 = :NEW.Utente2 THEN
        RAISE_APPLICATION_ERROR(-20006, 'Errore: Non puoi farti i complimenti da solo.');
    END IF;
END;
/
```

7. PROCEDURE

- ***Che cos'è una Stored Procedure?***

Una Stored Procedure (procedura memorizzata) è un insieme di istruzioni SQL e logica procedurale (come cicli IF/ELSE, WHILE, dichiarazione di variabili) che viene salvato, compilato e conservato direttamente all'interno del database. Possiamo immaginarla come una "funzione" o un "sottoprogramma" nella programmazione tradizionale: invece di scrivere lunghe sequenze di codice SQL all'interno dell'applicazione (sito web o app), si incapsula tutta la logica in un unico oggetto nel database, che può essere richiamato semplicemente per nome.

- ***Differenza tra Query e Trigger***

Per comprendere appieno il loro ruolo, è utile distinguerle dagli strumenti analizzati in precedenza:

Procedura e Query: una query è solitamente un'interrogazione statica per leggere o scrivere dati. Una procedura è dinamica: accetta parametri in ingresso (input), può restituire valori (output) e può eseguire decine di operazioni diverse in sequenza.

Procedura e Trigger: Un trigger scatta automaticamente in risposta a un evento (es. dopo un inserimento). Una procedura, invece, deve essere invocata esplicitamente (dall'utente, da un amministratore o dall'applicazione) quando serve eseguire una specifica operazione di business.

7.1 Inserimento Controllato Recensioni (Astrazione e Semplicità)

Questa procedura serve a semplificare e standardizzare il processo di creazione di una nuova recensione da parte delle applicazioni client (sito web o app mobile). Invece di obbligare lo sviluppatore dell'applicazione a scrivere complesse istruzioni INSERT manuali (dovendo conoscere a memoria i nomi delle colonne, il formato delle date e come generare gli ID univoci), si fornisce un'interfaccia semplice e "pulita". L'obiettivo è il principio dell'incapsulamento: l'applicazione deve solo fornire i dati essenziali (Chi, Dove, Voto, Testo), mentre il database si occupa di tutti i dettagli tecnici "dietro le quinte", garantendo che ogni recensione sia registrata nel formato corretto.

```
-- 7.  PROCEDURE

-- 1.   Inserimento "Sicuro" di una Recensione (User Experience
-- Questa procedura semplifica l'inserimento di una recensione. Invece di scrivere una
-- INSERT manuale, l'applicazione chiama questa procedura.
-- Essa gestisce automaticamente la data e l'ID (usando SYS_GUID se non fornito o gestendolo internamente).

CREATE OR REPLACE PROCEDURE SP_AGGIUNGI_RECENSIONE (
    p_Codice_Utente IN VARCHAR2,
    p_ID_Attivita   IN VARCHAR2,
    p_Voto          IN INTEGER,
    p_Testo         IN CLOB
) AS
    v_Nuovo_ID VARCHAR2(50);
BEGIN
    -- Genera un ID univoco
    v_Nuovo_ID := SYS_GUID();

    -- Inserimento
    INSERT INTO RECENSIONI (Codice_Recensione, Utente, Attivita, Valutazione, Testo, Data)
    VALUES (v_Nuovo_ID, p_Codice_Utente, p_ID_Attivita, p_Voto, p_Testo, SYSDATE);

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Recensione inserita con successo! ID: ' || v_Nuovo_ID);

EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Errore: ID duplicato.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Errore durante l''inserimento: ' || SQLERRM);
        ROLLBACK;
END;
```

7.2 Gestione Moderazione e Penalizzazione

Questa procedura è uno strumento dedicato ai ruoli con privilegi elevati (Admin e Moderator) per la gestione della qualità dei contenuti. Quando una recensione viene identificata come offensiva, spam o fraudolenta, non è sufficiente rimuoverla; spesso è necessario intraprendere un'azione correttiva contro l'autore per scoraggiare comportamenti futuri. Il moderatore deve poter dire "Elimina questa recensione e penalizza l'autore". Il sistema si occuperà di ripulire il contenuto e contemporaneamente abbassare il "Trust Score" (punteggio di affidabilità) dell'utente o segnalarlo, garantendo che le policy della piattaforma siano applicate in modo uniforme.

```
-- 2.   Moderazione: Rimozione Recensione e Penalizzazione (Admin/Moderator)
-- Quando un moderatore cancella una recensione offensiva, potrebbe voler anche abbassare il
-- "trust score" dell'utente (o semplicemente loggare l'evento).
-- Questa procedura cancella la recensione (i trigger aggiorneranno le medie automaticamente) e stampa un alert.

CREATE OR REPLACE PROCEDURE SP_RIMUOVI_RECENSIONE_MOD (
    p_Codice_Recensione IN VARCHAR2,
    p_Motivo            IN VARCHAR2
) AS
    v_Utente VARCHAR2(50);
BEGIN
    -- Recupera l'utente prima di cancellare per notificarlo (simulato)
    SELECT Utente INTO v_Utente
    FROM RECENSIONI
    WHERE Codice_Recensione = p_Codice_Recensione;

    -- Cancellazione (I trigger aggiorneranno automaticamente le medie voti su UTENTI e ATTIVITA)
    DELETE FROM RECENSIONI
    WHERE Codice_Recensione = p_Codice_Recensione;

    IF SQL%ROWCOUNT > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Recensione rimossa. Utente ' || v_Utente || ' segnalato per: ' || p_Motivo);
        COMMIT;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Recensione non trovata.');
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Recensione inesistente.');
END;
/
```

7.3 Controllo sulle modifiche dei contenuti da parte degli Utenti

```
-- 3.   Controllo sulle modifiche dei contenuti da parte degli Utenti

CREATE OR REPLACE PROCEDURE MODIFICA_MIA_RECENSIONE (
    p_id_recensione   IN VARCHAR2,
    p_nuovo_voto      IN NUMBER,
    p_nuovo_testo     IN CLOB,
    p_utente_loggato  IN VARCHAR2 -- L'ID dell'utente che sta provando a modificare
) IS
    v_owner VARCHAR2(50);
BEGIN
    -- 1. Trova chi è il proprietario della recensione
    -- Corretto ID_Recensione in Codice_Recensione
    SELECT Utente INTO v_owner
    FROM RECENSIONI
    WHERE Codice_Recensione = p_id_recensione;

    -- 2. Se l'utente corrisponde, esegui l'update
    IF v_owner = p_utente_loggato THEN
        UPDATE RECENSIONI
        SET Valutazione = p_nuovo_voto,
            Testo = p_nuovo_testo,
            Data = SYSDATE -- Aggiorniamo la data alla modifica corrente
        WHERE Codice_Recensione = p_id_recensione; -- Corretto nome colonna

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Recensione modificata con successo.');
    ELSE
        RAISE_APPLICATION_ERROR(-20001, 'Errore: Non puoi modificare recensioni non tue.');
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20002, 'Recensione non trovata.');
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE_APPLICATION_ERROR(-20003, 'Errore imprevisto: ' || SQLERRM);
END;
-- Assegna il permesso al ruolo ROLE_USER
GRANT EXECUTE ON MODIFICA_MIA_RECENSIONE TO ROLE_USER;
```

8. OTTIMIZZAZIONE DELLA BASE DI DATI

8.1 INDICI

Gli Indici sono strumenti essenziali per ottimizzare i tempi di lettura del Database, a fronte di un lieve overhead nelle operazioni di scrittura. Abbiamo applicato gli indici in modo strategico sulle tabelle a maggior volume di dati, ottenendo una riduzione drastica dei tempi di esecuzione delle query più complesse.

Per bilanciare le performance, abbiamo creato indici specifici non solo sulle Chiavi Primarie (automatici), ma anche sulle colonne utilizzate frequentemente nelle clausole WHERE e JOIN. Questo ci ha permesso di mantenere le query veloci anche dopo il popolamento massivo della base dati.

```
-- 5. INDICI

-- Indici per velocizzare la ricerca sui Check-in per Attività e Data
CREATE INDEX IDX_CHECKIN_ATTIVITA ON CHECK_IN (ATTIVITA);
CREATE INDEX IDX_CHECKIN_DATA ON CHECK_IN (DATA);

-- Indici per velocizzare la ricerca sulle Recensioni su utente e attività (per join)
CREATE INDEX IDX_RECENSIONI_UTENTE ON RECENSIONI (UTENTE);
CREATE INDEX IDX_RECENSIONI_ATTIVITA ON RECENSIONI (ATTIVITA);

-- Indice per velocizzare la ricerca sulle foto delle Attività
CREATE INDEX IDX_FOTO_ATTIVITA ON FOTOGRAFIE(ATTIVITA);

-- Indice per velocizzare la ricerca sui nomi delle Attività
CREATE INDEX IDX_ATTIVITA_NOME ON ATTIVITA(NOME);

-- Indice per velocizzare la ricerca sui Servizi associati alle Attività
CREATE INDEX IDX_OFFRE_SERVIZIO_SERVIZI ON OFFRE_SERVIZIO(ID_SERVIZIO);
```

8.2 GESTIONE DELLA CONCORRENZA (Concurrency Control)

É possibile (e necessario) gestire la concorrenza in questo progetto, e sia il metodo 2PL (Two-Phase Locking) che il Timestamp Ordering sono applicabili.

Essendo un sistema multiutente (Admin, Moderator, User) con dati aggregati (es. numero recensioni, media voti), i conflitti di concorrenza sono inevitabili. Prima di scegliere il metodo, individuiamo dove avvengono i conflitti critici (Race Conditions) nel nostro sistema:

- **Aggiornamento Contatori**: Due utenti scrivono una recensione per la stessa pizzeria nello stesso istante. Entrambi leggono Numero_Recensioni = 100. Entrambi incrementano a 101 e salvano. Risultato: 101 invece di 102. (Dato perso).
- **Calcolo Media Voti**: Simile a sopra, ma per Valutazione_Media.
- **Moderazione vs Modifica**: Un utente sta modificando la sua recensione mentre un moderatore la sta cancellando.
- **Check-in Multipli**: Tanti utenti fanno check-in nello stesso momento durante un evento affollato.

In questo progetto è molto più conveniente applicare il metodo 2PL per gestire la concorrenza, questo perché il database contiene molti dati aggregati (Numero_Recensioni, Valutazione_Media, Num_Complimenti). Questi campi sono "hotspot": vengono aggiornati molto spesso da utenti diversi contemporaneamente. Il Locking (2PL) gestisce meglio questi scenari di contesa rispetto al Timestamp, che causerebbe troppi riavvii delle transazioni.

In questo progetto il 2PL agirebbe in questo modo:

1. ***Fase 1 (Growing - Acquisizione Lock)***

La transazione di Utente X vuole aggiornare ATTIVITA. Richiede un Lock Esclusivo (Write Lock) sulla riga del "Bar Centrale".
Se ottiene il lock, la transazione di Utente Y deve attendere (si mette in coda).
Operazione: Utente x inserisce la recensione, calcola la nuova media voti e aggiorna la tabella ATTIVITA.

2. ***Fase 2 (Shrinking - Rilascio Lock)***

Utente X fa COMMIT. I lock vengono rilasciati.
Ora Utente Y ottiene il lock, legge i dati aggiornati da Utente X (es. recensioni passate da 10 a 11), fa il suo calcolo e aggiorna a 12.

```
create or replace PROCEDURE INSERISCI\_RECENSIONE (
    p_Codice_Recensione IN VARCHAR2,
    p_Utente            IN VARCHAR2,
    p_Attivita          IN VARCHAR2,
    p_Valutazione       IN INTEGER,
    p_Testo             IN CLOB
) AS
    v_Vecchia_Media NUMBER(3,2);
    v_Nuovo_Num_Rec INTEGER;
BEGIN
    -- \*\*\* INIZIO TRANSAZIONE (Fase di Crescita - Acquisizione Lock) \*\*\*

    -- 1. Inserimento della Recensione
    -- Oracle acquisisce un LOCK ESCLUSIVO sulla nuova riga in RECENSIONI
    INSERT INTO RECENSIONI (Codice_Recensione, Utente, Attivita, Valutazione, Testo, Data)
    VALUES (p_Codice_Recensione, p_Utente, p_Attivita, p_Valutazione, p_Testo, SYSDATE);
```

```
    -- 2. Aggiornamento dell'Attività (Lock sulla riga dell'attività)
    -- Per evitare letture sporche, blocchiamo la riga dell'attività per aggiornare i contatori
    -- Questo update ricalcola la media in modo incrementale per efficienza
    UPDATE ATTIVITA
    SET
        -- Formula matematica per aggiornare la media senza rileggere tutte le recensioni
        Valutazione_Media = CASE
            WHEN Numero_Recensioni = 0 THEN p_Valutazione
            ELSE ((Valutazione_Media * Numero_Recensioni) + p_Valutazione) / (Numero_Recensioni + 1)
        END,
        Numero_Recensioni = Numero_Recensioni + 1
    WHERE ID_Attivita = p_Attivita;
```

```
    -- 3. Aggiornamento dell'Utente (Lock sulla riga dell'utente)
    UPDATE UTENTI
    SET Num_Recensioni = Num_Recensioni + 1
    WHERE Codice_Utente = p_Utente;


    -- \*\*\* FINE TRANSAZIONE (Fase di Decrescita - Rilascio Lock) \*\*\*
    -- Il COMMIT rilascia istantaneamente tutti i lock acquisiti sopra.
    COMMIT;


    DBMS\_OUTPUT.PUT\_LINE('Recensione inserita e statistiche aggiornate con successo.');


EXCEPTION
    WHEN OTHERS THEN
        -- Se qualcosa va storto, ROLLBACK annulla tutto e rilascia i lock
        ROLLBACK;
        DBMS\_OUTPUT.PUT\_LINE('Errore durante l inserimento: ' || SQLERRM);
        RAISE; -- Rilancia l'errore per farlo vedere a SQL Developer
END;
/
```

Si potrebbe applicare la gestione della concorrenza anche alle seguenti operazioni:
Check-in: Aggiornare la tabella check-in (se ci sono contatori)
Aggiunta Foto: Aggiornare eventuali contatori foto.
Complimenti: Aggiornare il contatore complimenti dell'utente.

8.3 STRATEGIE DI AFFIDABILITA' E GESTIONE DEI GUASTI

Per garantire la robustezza della piattaforma di recensioni e assicurare le proprietà ACID (in particolare Atomicità e Persistenza) delle transazioni, il sistema adotta un rigido protocollo di controllo di affidabilità. L'obiettivo è ripristinare il corretto stato del database (recovery) a valle di possibili malfunzionamenti hardware o software, siano essi accidentali o intenzionali.

La strategia implementata si basa sulla gestione gerarchica delle memorie: memoria centrale (volatile), memoria di massa (persistente ma soggetta a guasti) e memoria stabile (indistruttibile per astrazione, realizzata tramite ridondanza).

8.3.1 Il Log (Diario di Bordo) e le Regole di Scrittura

Il cuore del sistema di affidabilità è il File di Log, registrato su memoria stabile, che traccia sequenzialmente tutte le operazioni svolte dalle transazioni (Begin, Insert, Update, Commit, Abort).

Ogni record di log contiene l'identificativo della transazione (**T**), il timestamp (**ts**), l'azione svolta, l'oggetto coinvolto (**O**), il valore prima della modifica (Before-Image, **BI**) e il valore dopo la modifica (After-Image, **AI)**.

Per garantire la corretta ripresa, il sistema impone due protocolli fondamentali:

- **Write Ahead Log (WAL):** Prima che una transazione (es. l'inserimento di una recensione) scriva effettivamente nel database su disco, deve scrivere il relativo record nel Log. Questo consente, in caso di guasto, di disfare (UNDO) le azioni non completate, poiché il valore precedente (BI) è salvo nel log.
- **Commit Precedenza**: Il sistema scrive sul log tutti i record relativi a una transazione (incluso il record di commit) prima di confermare il successo all'utente. Questo consente di rifare (REDO) le azioni perse dalla memoria volatile, poiché il nuovo valore (**AI**) è persistente nel log.

8.3.2 Strategie di Backup e Ottimizzazione

Per evitare che il file di log cresca indefinitamente e per velocizzare le operazioni di ripristino, vengono adottate due tecniche periodiche:

- **Checkpoint (Sincronizzazione):** periodicamente, il sistema esegue un checkpoint per creare un "punto della situazione" certo.
  Durante questa fase:
  - Si sospende l'accettazione di nuove operazioni.
  - Si forzano su memoria di massa tutte le pagine "sporche" (modificate) presenti nel buffer.
  - Si scrive un record di checkpoint sul log elencando le transazioni attive in quel momento.
    Questo garantisce che, fino a quel punto, tutte le transazioni committate siano fisicamente salvate su disco.

- **Dump (Backup Completo):** pianificato nei momenti di minore attività (es. notte), viene prodotto un Dump, ovvero una copia completa del contenuto della base di dati su memoria stabile. Questo è essenziale per il ripristino in caso di danni fisici ai dischi (guasti hard).

8.3.3 Procedure di Recovery (Ripristino)

Il sistema distingue le procedure di ripristino in base alla tipologia di guasto (modello fail-stop):

- **Ripresa a Caldo (Warm Restart)**

Applicata in caso di guasti soft (es. crash del sistema operativo, caduta di tensione) dove si perde solo il contenuto della memoria centrale (RAM).

Il gestore dell'affidabilità esegue i seguenti passi:

- Ripercorre il log a ritroso fino all'ultimo record di Checkpoint.
- Classifica le transazioni in due insiemi:
  - Insieme UNDO: Transazioni attive al momento del guasto o abortite. Queste devono essere disfatte.
  - Insieme REDO: Transazioni che hanno effettuato il commit prima del guasto ma i cui dati potrebbero non essere stati scritti su disco. Queste devono essere rifatte.

  Esegue le operazioni di Undo (ripristinando i valori **BI** all'indietro) e successivamente le operazioni di Redo (riapplicando i valori **AI** in avanti).

- **Ripresa a Freddo (Cold Restart)**

Applicata in caso di guasti hard (es. rottura del disco rigido o danneggiamento del filesystem).

La procedura prevede:

- Ripristino dei dati partendo dall'ultimo Dump (backup) disponibile.
- Esecuzione di tutte le operazioni registrate nel log a partire dal dump fino all'istante del guasto.
- Esecuzione finale di una ripresa a caldo per allineare lo stato delle transazioni attive.

8.3.4 Replicazione e Memoria Stabile

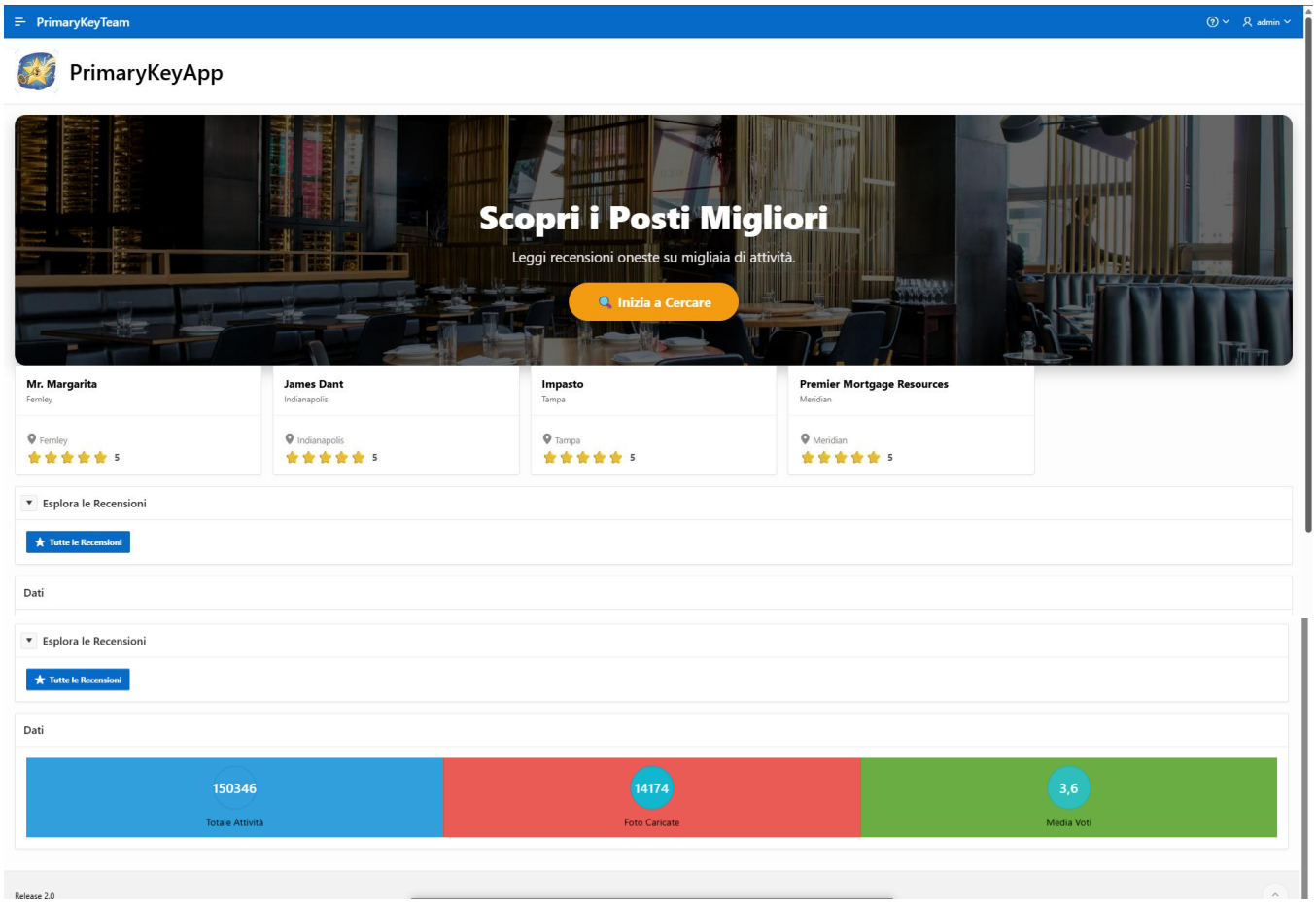
Per implementare il concetto teorico di "memoria stabile" (che non può danneggiarsi) e garantire la disponibilità del servizio anche in caso di guasto di un singolo supporto, il sistema utilizza tecnologie di ridondanza hardware.

Nello specifico, lo storage secondario è configurato utilizzando repliche RAID di dischi. Questa strategia di replicazione assicura che il fallimento di un'unità fisica non comporti la perdita di dati, permettendo al sistema di continuare a operare o di effettuare il recovery senza ricorrere necessariamente al Dump.

9. INTERFACCIA UTENTE (applicazione APEX)

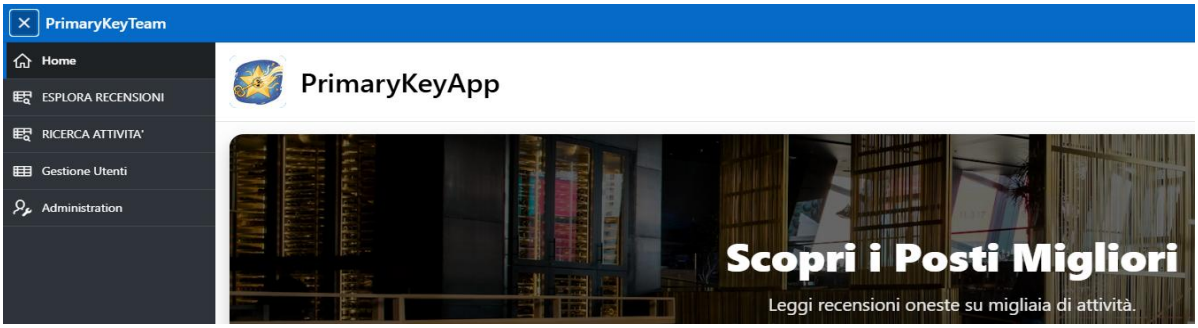
L'Applicazione Web **PrimaryKeyApp** è stata realizzata con gli strumenti di **APEX** (*Oracle Application Express*). L'obbiettivo è ottenere una piattaforma di Gestione delle Recensioni delle Attività registrate nel Database, attraverso un'interfaccia Utente moderna e interattiva.

La Pagina **HOME** si presenta come segue:

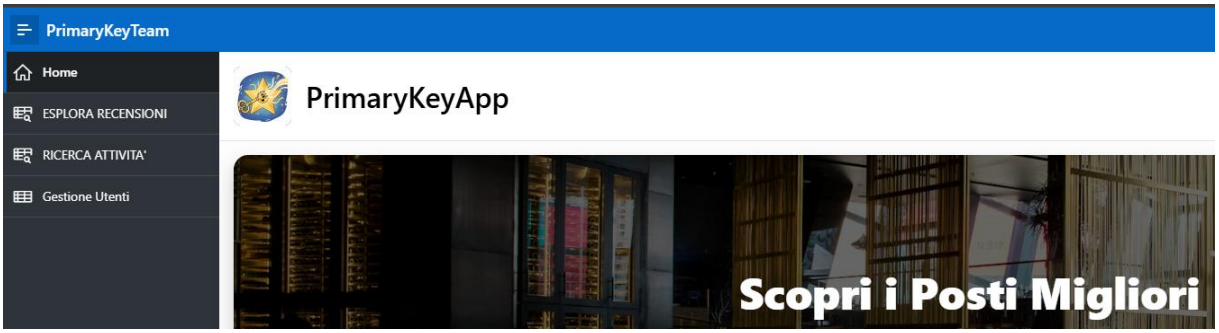


Sulla piattaforma sono stati registrati quattro tipi di utente (con i nomi dei componenti del Team) con ruoli diversi, in modo da risaltare i diversi privilegi e la differente visione della stessa applicazione con Login diversi:

- **Admin** (Angelo Russo) - visualizza tutte le pagine disponibili (HOME, ESPLORA RECENSIONI, RICERCA ATTIVITA', GESTIONE UTENTI, ADMINISTRATION);



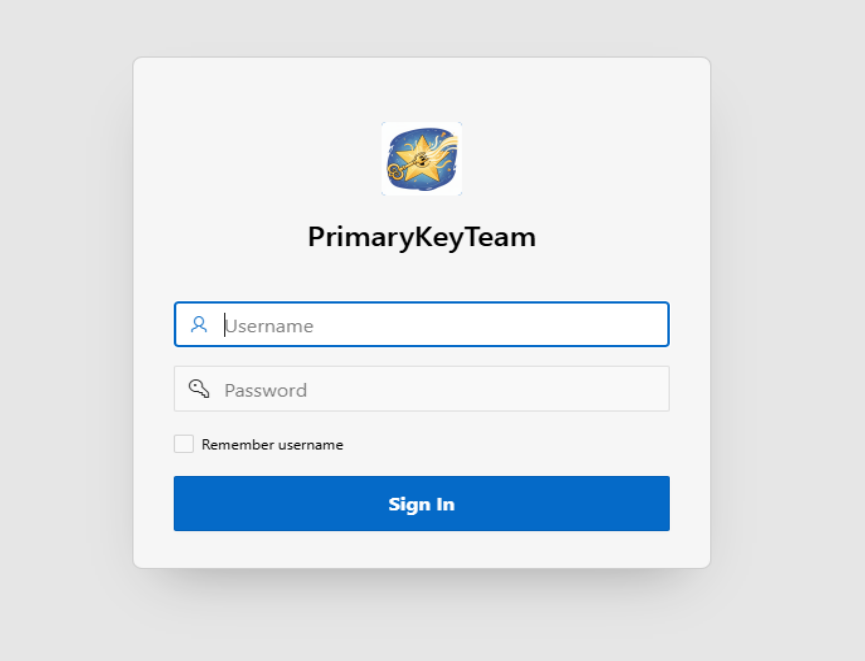
- **Moderator** (Anna Paola Granato) - visualizza tutte le pagine tranne ADMINISTRATION (in accesso esclusivo per l'admin); a differenza degli Users, possono visualizzare l'elenco di tutti gli utenti registrati alla piattaforma;



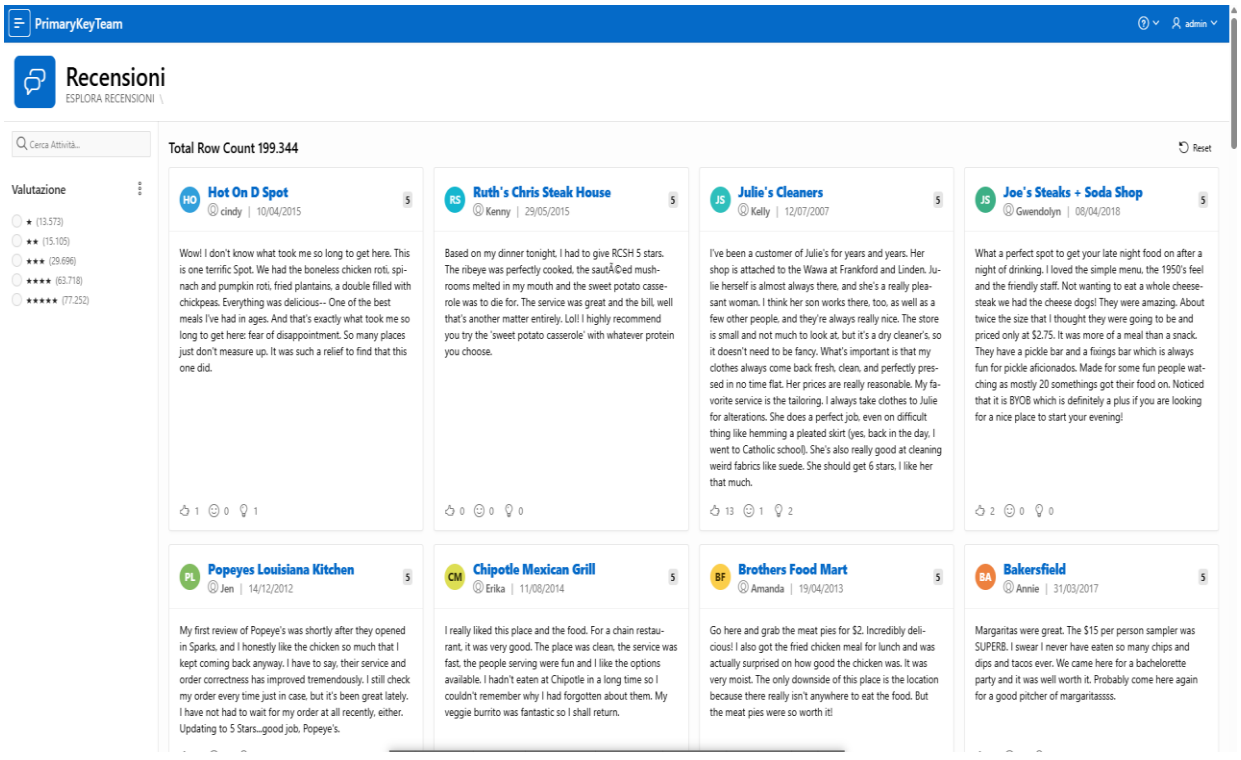
- **User1, User2** (rispettivamente Giada Savi e Giovanni Cozzolino) - visualizzano solo la pagina HOME, ESPLORA RECENSIONI, RICERCA ATTIVITA', non hanno in alcun modo accesso alle pagine di amministrazione e moderazione

Di seguito la Panoramica sulle altre pagine che compongono l'applicazione:

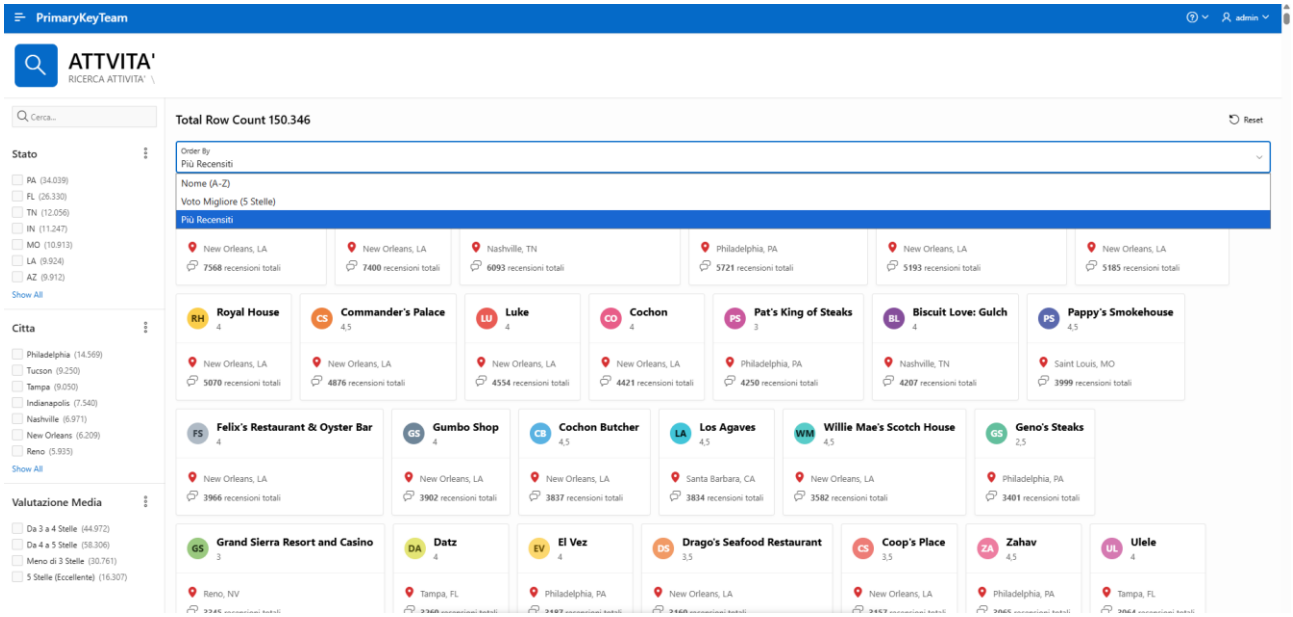
- **Schermata di ACCESSO**:



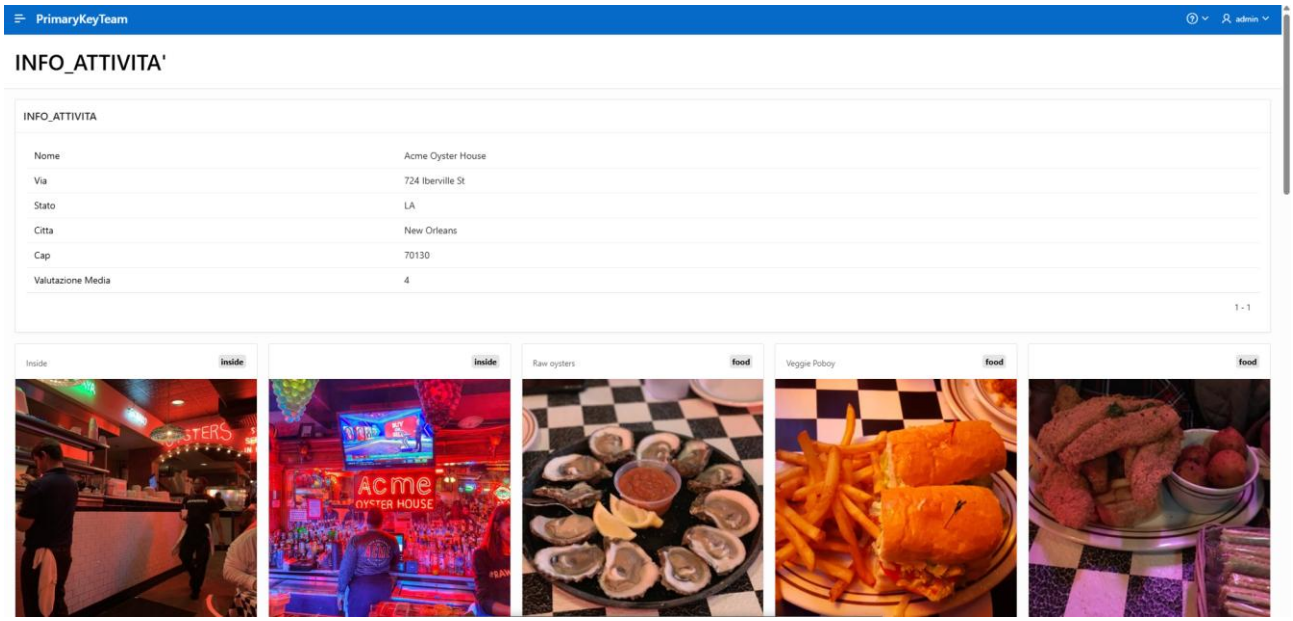
- **ESPLORA RECENSIONI** (accessibile tramite Button):



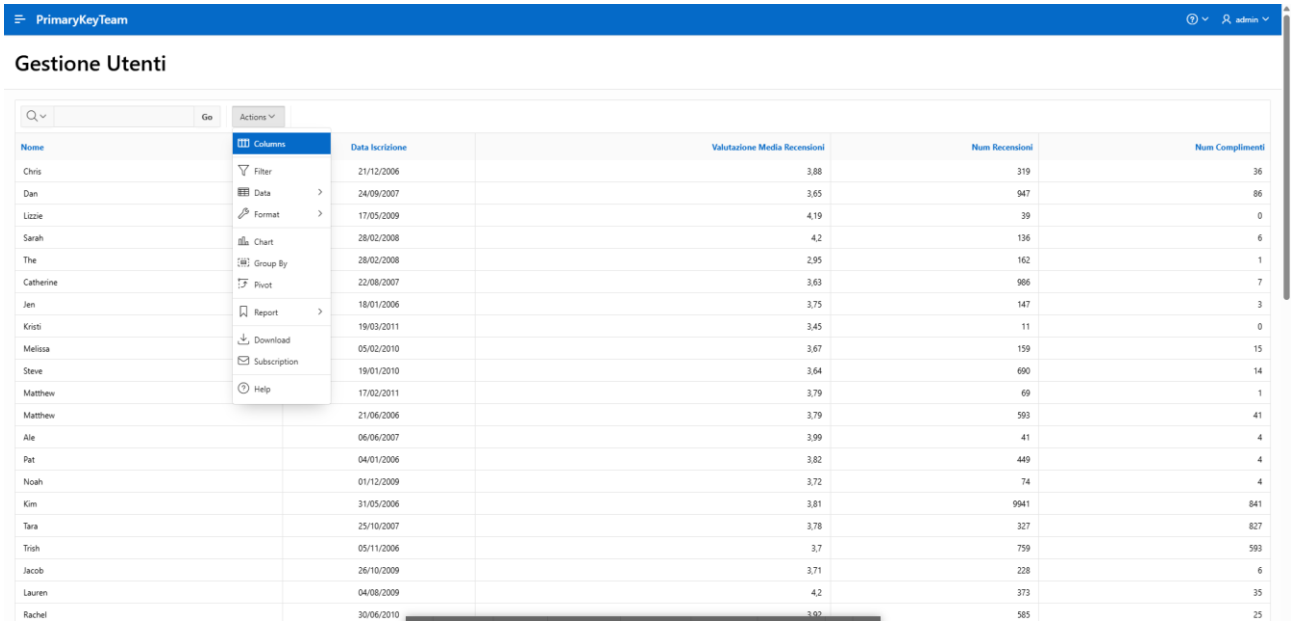
- **RICERCA ATTIVITA'**:



Se si clicca su una qualsiasi attività si ottengono tutte le informazioni su tale attività e le foto ad essa associate:

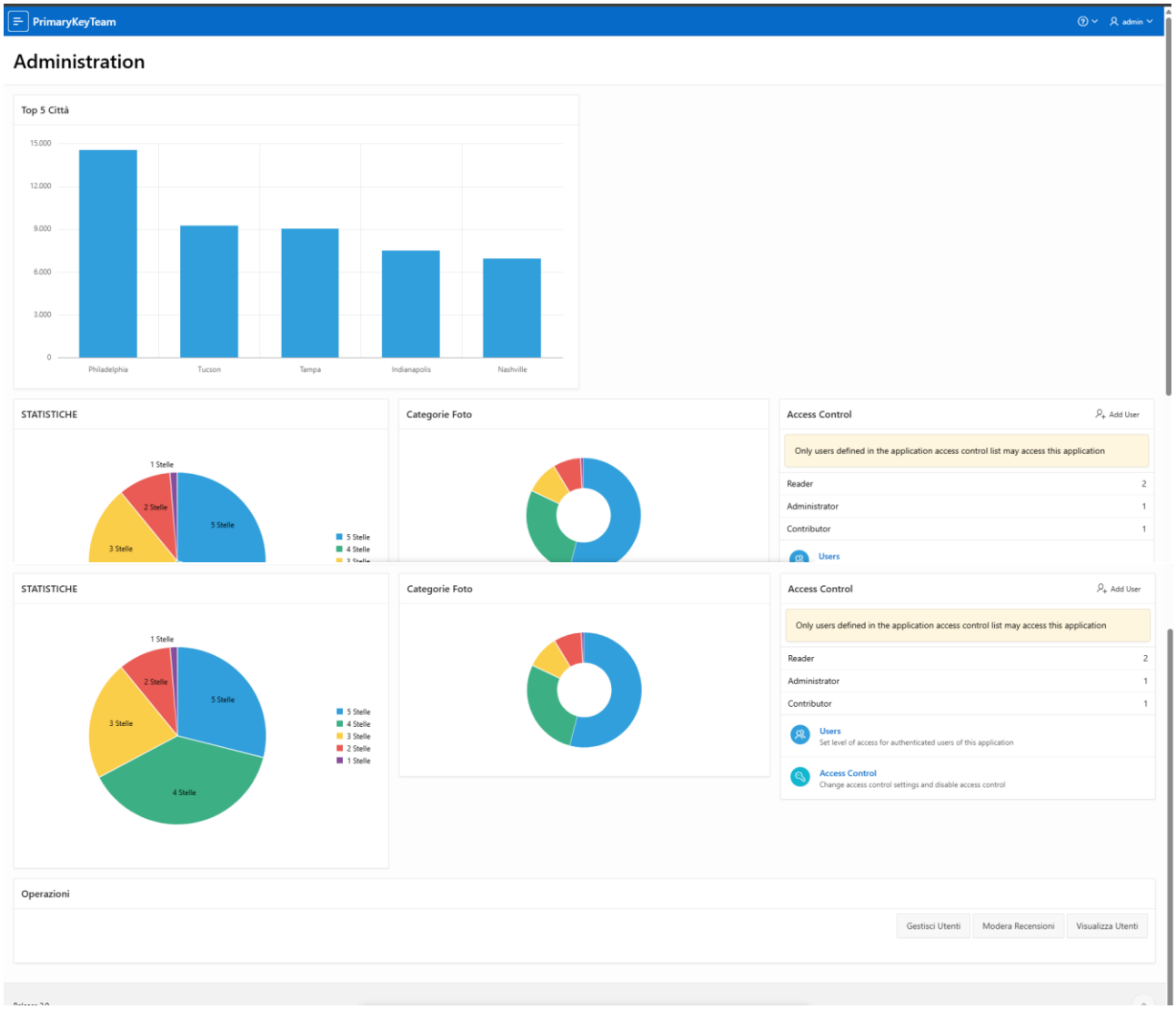


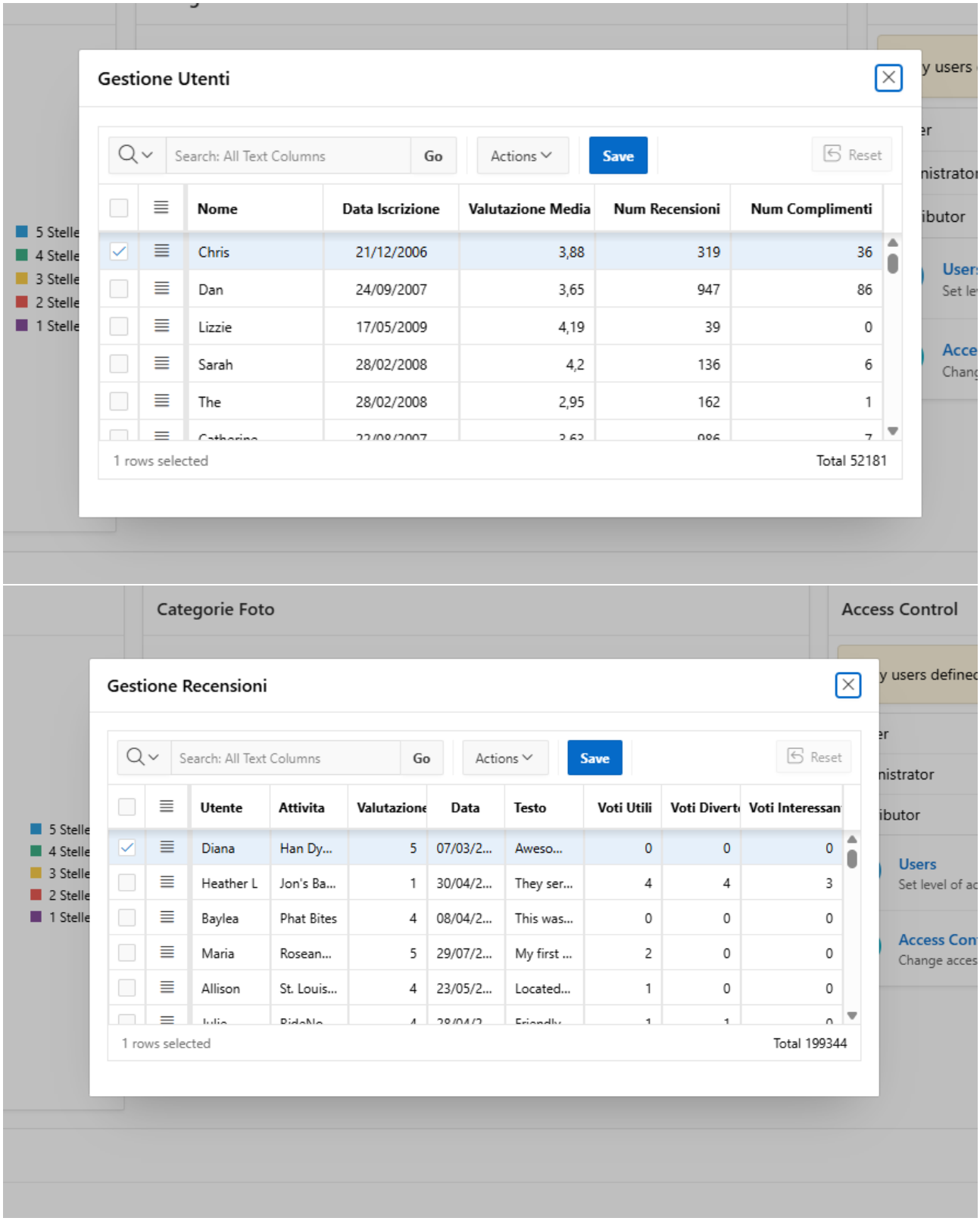
- **GESTIONE UTENTI** (elenco di tutti gli utenti iscritti alla piattaforma):



Nome	Data Iscrizione	Valutazione Media Recensioni	Num Recensioni	Num Complimenti
Chris	21/12/2006	3,88	319	36
Dan	24/09/2007	3,65	947	86
Lizzie	17/05/2009	4,19	39	0
Sarah	28/02/2008	4,2	136	6
The	28/02/2008	2,95	162	1
Catherine	22/08/2007	3,63	986	7
Jen	18/01/2006	3,75	147	3
Kristi	19/03/2011	3,45	11	0
Melissa	05/02/2010	3,67	159	15
Steve	19/01/2010	3,64	690	14
Matthew	17/02/2011	3,79	69	1
Matthew	21/06/2006	3,79	593	41
Ale	06/06/2007	3,99	41	4
Pat	04/01/2006	3,82	449	4
Noah	01/12/2009	3,72	74	4
Kim	31/05/2006	3,81	9941	841
Tara	25/10/2007	3,78	327	827
Trish	05/11/2006	3,7	759	593
Jacob	26/10/2009	3,71	228	6
Lauren	04/08/2009	4,2	373	35
Rachel	30/06/2010	3,92	585	25

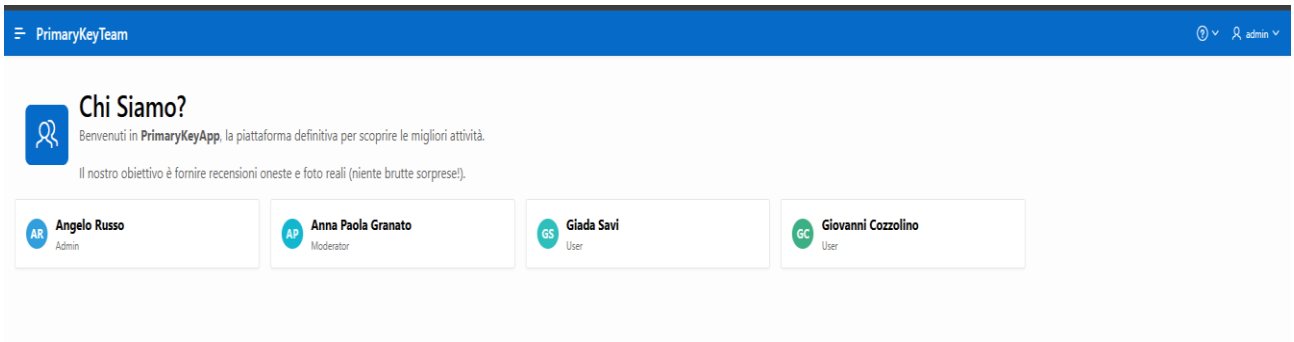
- ADMINISTRATION:





Con la possibilità esclusivamente per l'admin di rimuovere Utenti e Recensioni.

- **ABOUT PAGE**:



10. LINGUAGGIO NATURALE

Abbiamo sviluppato un'interfaccia in linguaggio naturale che abilita l'esecuzione di Query tramite un approccio Text-to-SQL. L'obiettivo è democratizzare l'accesso ai dati, permettendo anche a utenti non tecnici di interrogare il Database senza conoscere il linguaggio SQL.

Inizialmente, è stata valutata l'adozione di Select AI, il motore nativo incluso nel pacchetto DBMS_CLOUD_AI di Oracle. Tuttavia, dato il vincolo architetturale che prevedeva l'utilizzo di Oracle Database 21c in locale (tramite SQL Developer) e per evitare la migrazione dell'intera infrastruttura su Cloud, abbiamo optato per una soluzione diversa.

La soluzione implementata sfrutta uno script Python (*main.py*) che agisce da ponte tra il Database locale e le API di Cohere AI (il modello LLM scelto per la traduzione Text-to-SQL). L'utente può interagire con il sistema direttamente da riga di comando: digitando la richiesta in italiano, lo script genera ed esegue la query SQL corrispondente, restituendo i risultati in tempo reale.

Esempi di interrogazioni:

```
C:\Users\angio\Desktop\MioProgettoAI>python main.py
--- 🔌 Connessione al Database in corso... ---

✅ CONNESSIONE RIUSCITA!
Collegato come utente: PROG1
Il sistema è pronto. Scrivi la tua domanda (o 'esci' per chiudere).
--------------------------------------------------

🧠 Domanda: elencami tutte le tabelle presenti nel Database
C:\Users\angio\Desktop\MioProgettoAI\main.py:55: LangChainDeprecationWarning: The method `Chain.run` was deprecated in langchain-classic 0.1.0 and will be removed in 1.0. Use `invoke` instead.
  response = db_chain.run(user_input)


> Entering new SQLDatabaseChain chain...
elencami tutte le tabelle presenti nel Database. Rispondi solo con la query SQL. NON mettere il punto e virgola finale (;). NON aggiungere caratteri dopo la query.
SQLQuery:SELECT table_name
FROM user_tables
SQLResult: [('UTENTI',), ('ATTIVITA',), ('RECENSIONI',), ('FOTOGRAFIE',), ('CHECK_IN',), ('CATEGORIE',), ('SERVIZI',), ('SUGGERIMENTI',), ('ORARI',), ('COMPLIMENTI',), ('APPARTIENE_A_CATEGORIA',), ('OFFRE_SERVIZIO',), ('RELAZIONE_AMICIZIA',), ('GIORNI_CHECK_IN',)]
Answer:UTENTI
ATTIVITA
RECENSIONI
FOTOGRAFIE
CHECK_IN
CATEGORIE
SERVIZI
SUGGERIMENTI
ORARI
COMPLIMENTI
APPARTIENE_A_CATEGORIA
OFFRE_SERVIZIO
RELAZIONE_AMICIZIA
GIORNI_CHECK_IN
> Finished chain.

🤖 Risposta: UTENTI
ATTIVITA
RECENSIONI
FOTOGRAFIE
CHECK_IN
CATEGORIE
SERVIZI
SUGGERIMENTI
ORARI
COMPLIMENTI
APPARTIENE_A_CATEGORIA
OFFRE_SERVIZIO
RELAZIONE_AMICIZIA
GIORNI_CHECK_IN
```

```
🧠 Domanda: fai l'elenco dei 10 stati/nazioni con piu attività di tipo "Restaurant"


> Entering new SQLDatabaseChain chain...
fai l'elenco dei 10 stati/nazioni con piu attività di tipo "Restaurant". Rispondi solo con la query SQL. NON mettere il punto e virgola finale (;). NON aggiungere caratteri dopo la query.
SQLQuery:SELECT stato, COUNT(id_attivita) AS num_attivita
FROM attivita
WHERE nome LIKE '%Restaurant%'
GROUP BY stato
ORDER BY num_attivita DESC
FETCH FIRST 10 ROWS ONLY
SQLResult: [('PA', 1012), ('FL', 644), ('NJ', 335), ('TN', 308), ('IN', 283), ('MO', 281), ('LA', 280), ('AB', 235), ('AZ', 158), ('DE', 104)]
Answer:[('PA', 1012), ('FL', 644), ('NJ', 335), ('TN', 308), ('IN', 283), ('MO', 281), ('LA', 280), ('AB', 235), ('AZ', 158), ('DE', 104)]
> Finished chain.

🤖 Risposta: [('PA', 1012), ('FL', 644), ('NJ', 335), ('TN', 308), ('IN', 283), ('MO', 281), ('LA', 280), ('AB', 235), ('AZ', 158), ('DE', 104)]
--------------------------------------------------
```

```
🧠 Domanda: dammi un informazione su un utente a piacere tuo


> Entering new SQLDatabaseChain chain...
dammi un informazione su un utente a piacere tuo. Rispondi solo con la query SQL. NON mettere il punto e virgola finale (;). NON aggiungere caratteri dopo la query.
SQLQuery:SELECT * FROM utenti WHERE codice_utente = '582iTyqKkoHeBFPMJuQEJw' FETCH FIRST 1 ROWS ONLY
SQLResult: [('582iTyqKkoHeBFPMJuQEJw', 'Chris', datetime.datetime(2006, 12, 21, 3, 58, 10), Decimal('3.88'), 319, 36)]
Answer:{'codice_utente': '582iTyqKkoHeBFPMJuQEJw', 'nome': 'Chris', 'data_iscrizione': datetime.datetime(2006, 12, 21, 3, 58, 10), 'valutazione_media_recensioni': Decimal('3.88'), 'num_recensioni': 319, 'num_complimenti': 36}
> Finished chain.

🤖 Risposta: {'codice_utente': '582iTyqKkoHeBFPMJuQEJw', 'nome': 'Chris', 'data_iscrizione': datetime.datetime(2006, 12, 21, 3, 58, 10), 'valutazione_media_recensioni': Decimal('3.88'), 'num_recensioni': 319,
'num_complimenti': 36}
--------------------------------------------------

🧠 Domanda: esci

C:\Users\angio\Desktop\MioProgettoAI>
```

```
🧠 Domanda: Fammi la classifica delle prime 5 attività per valutazione media con più di 500 recensioni


> Entering new SQLDatabaseChain chain...
Fammi la classifica delle prime 5 attività per valutazione media con più di 500 recensioni. Rispondi solo con la query SQL. NON mettere il punto e virgola finale (;). NON aggiungere caratteri dopo la query.
SQLQuery:SELECT attivita.nome, attivita.valutazione_media
FROM attivita
WHERE attivita.numero_recensioni > 500
ORDER BY attivita.valutazione_media DESC
FETCH FIRST 5 ROWS ONLY
SQLResult: [('Blues City Deli', Decimal('5')), ('SUGARED + BRONZED', Decimal('5')), ('Yats', Decimal('5')), ('Free Tours By Foot', Decimal('5')), ('Tumerico', Decimal('5'))]
Answer:1. Blues City Deli - 5
2. SUGARED + BRONZED - 5
3. Yats - 5
4. Free Tours By Foot - 5
5. Tumerico - 5
> Finished chain.

🤖 Risposta: 1. Blues City Deli - 5
2. SUGARED + BRONZED - 5
3. Yats - 5
4. Free Tours By Foot - 5
5. Tumerico - 5
--------------------------------------------------
```