Basi di Dati

Progetto A.A. 2024/2025

CUSTOMER RELATIONSHIP MANAGEMENT

0308023

Sofia Tosti

Indice

1. Descrizione del Minimondo

Un sistema di Customer Relationship Management (o gestione delle relazioni con i clienti) è un sistema informativo che verte sulla ***fidelizzazione*** del cliente. Si vuole realizzare un sistema CRM per un'azienda marketing-oriented che intende realizzare relazioni durevoli di breve e lungo periodo con i propri clienti, massimizzando quindi il valore degli stessi. L'azienda contatta periodicamente i suoi clienti, di cui è a conoscenza di tutte le ***informazioni anagrafiche*** di interesse e a cui associa anche un certo numero di ***contatti telefonici e di email***. Inoltre, per ogni cliente, viene memorizzata anche la data di registrazione nel sistema. Gli operatori dell'azienda interagiscono periodicamente con i clienti per proporre nuove offerte commerciali, come sconti su prodotti, saldi o promozioni particolari, per fare alcuni esempi. Queste offerte sono di varia natura e quindi il sistema deve poter permettere alla segreteria di fornire una breve descrizione per ogni offerta. Quando un operatore contatta un cliente mediante uno dei recapiti telefonici forniti, propone al cliente una delle offerte dell'azienda. Poiché lo stesso cliente può essere contattato da più ***utenti***, il sistema deve prevedere un meccanismo per registrare delle ***note*** in cui gli operatori annotano dettagli sul risultato dell'interazione. Gli ***utenti*** possono richiamare l'elenco delle ***note*** di un cliente che stanno contattando. Qualora un cliente accettasse l'offerta, questa deve essere registrata nel sistema, indicando quale utente ha permesso l'accettazione e in quale data. Un cliente potrebbe decidere di richiedere un appuntamento in una delle sedi dell'azienda. In questo caso, alla ***nota*** deve essere allegato un appuntamento, indicando in quale sede e in quale data/ora lo stesso si svolgerà. La segreteria può, in ogni momento, generare un report che mostri, in un intervallo temporale specificato, per tutti i clienti del sistema, quanti sono stati contattati e quante volte, così come quante offerte sono state accettate da ciascuno.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
2	Fidelizzazione	Gestione delle relazioni	Definisce meglio l'obiettivo del CRM che è mantenere e valorizzare i rapporti con gli interlocutori
5	Azienda	Operatori	Chiarisce chi contatti i clienti.
6	Informazioni anagrafiche	Nome, cognome, data di nascita, codice fiscale e indirizzo di residenza	Specifica in modo dettagliato quali dati identificativi del cliente vengono raccolti, evitando ambiguità su quali informazioni anagrafiche siano necessarie.
8	Contatti telefonici/ email	Recapiti di contatto	Chiarisce che si tratta di informazioni di contatto fornite dal cliente.
10, 11	Offerta	Offerta commerciale	Evita ambiguità con il concetto di "promozione" o "sconto", specificando che si tratta di una comunicazione mirata.
19,24	Nota	Feedback dell'interazione/Note dell'interazione	Specifica chiaramente che si tratta di un'annotazione legata al contatto con l'interlocutore.
18,20	Utente	Operatore	Specifica il ruolo di chi interagisce con il CRM, eliminando l'ambiguità.

Specifica disambiguata

Un sistema di **Customer Relationship Management** (o gestione delle relazioni con i clienti) è un sistema informativo che verte sulla **gestione delle relazioni** con il cliente. Si vuole realizzare un sistema CRM per un'azienda **marketing-oriented**, che intende stabilire relazioni durevoli di breve e lungo periodo con i propri **potenziali clienti/interlocutori**, massimizzandone il valore.

Gli **operatori** contattano periodicamente i clienti, di cui raccolgono **nome, cognome, data di nascita, codice fiscale e indirizzo di residenza**, associando anche un certo numero di **recapiti di contatto** (telefonici ed email). Inoltre, per ogni cliente, viene memorizzata la data di registrazione nel sistema.

Gli **operatori aziendali** interagiscono periodicamente con i clienti per proporre nuove **offerte commerciali**, come sconti su prodotti, saldi o promozioni particolari. Queste offerte sono di varia natura e quindi il sistema deve permettere alla segreteria di

fornire una breve descrizione per ciascuna.

Quando un operatore contatta un cliente mediante uno dei **recapiti di contatto** forniti, propone una delle **offerte commerciali** dell'azienda. Poiché lo stesso cliente può essere contattato da più operatori, il sistema deve prevedere un meccanismo per registrare il **feedback dell'interazione/note dell'interazione**, in cui gli operatori annotano dettagli sul risultato della comunicazione. Gli operatori possono consultare l'elenco delle **note dell'interazione** di un cliente durante il contatto.

Se un cliente accetta un'offerta, questa deve essere registrata nel sistema, indicando quale operatore ha permesso l'accettazione e in quale data. Inoltre, un cliente potrebbe richiedere un appuntamento in una delle sedi aziendali. In tal caso, alla **nota dell'interazione** deve essere allegata la registrazione di un appuntamento, indicando la sede e la data/ora dell'incontro.

La segreteria può, in qualsiasi momento, generare un report che mostri, in un intervallo temporale specificato, quanti clienti sono stati contattati e con quale frequenza, nonché quante **offerte commerciali** sono state accettate da ciascuno.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Clienti	Persona registrata nel sistema con cui l'azienda interagisce, di cui vengono memorizzate informazioni personali e contatti telefonici/email.	Interlocutore, Destinatario	Offerta commerciale, Interazione
Operatore	Utente del sistema CRM che interagisce con i contatti per proporre offerte e registrare note sulle interazioni.	Agente, Addetto	Interazione, Offerta commerciale, Nota dell'interazione
Offerta commerciale	Proposta commerciale fatta al contatto	Promozione, Sconto	Operatore, Interazione

	dall'operatore, con descrizione e promozione di prodotti o servizi.		
Interazione	Comunicazione tra l'operatore e il contatto, registrata nel sistema attraverso note che descrivono l'esito e i dettagli della proposta fatta.	Contatto, Comunicazione	Operatore, Nota dell'interazione, Offerta
Nota dell'interazione	Annotazione fatta dall'operatore durante o dopo l'interazione, contenente informazioni sull'esito della comunicazione e possibili richieste di appuntamenti.	Annotazione, Commento, Registro di interazione, Feedback	Interazione, Operatore, Cliente, Offerta
Segreteria	Utente del sistema CRM che permette di creare e aggiornare le offerte, generare report.	Supporto, Amministrazione	Offerta, Cliente
Appuntamento	Rappresenta un evento pianificato in cui un cliente si incontra con un rappresentante dell'azienda (ad esempio, un operatore) in una specifica data e ora, presso una determinata sede aziendale.	Incontro, Colloquio	Operatore, Interazione, Cliente

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Clienti
Gli operatori contattano periodicamente i loro clienti, di cui raccoglie **nome, cognome, data di nascita, codice fiscale e indirizzo di residenza**, associando anche un certo numero di **recapiti di contatto** (telefonici ed email). Inoltre, per ogni cliente, viene memorizzata la data di registrazione nel sistema. Gli **operatori aziendali** interagiscono periodicamente con i clienti per proporre nuove **offerte commerciali**, come sconti su prodotti, saldi o promozioni particolari.

Quando un operatore contatta un cliente mediante uno dei **recapiti di contatto** forniti, propone una delle **offerte commerciali** dell'azienda. Poiché lo stesso cliente può essere contattato da più operatori, il sistema deve prevedere un meccanismo per registrare il **feedback dell'interazione/note dell'interazione**, in cui gli operatori annotano dettagli sul risultato della comunicazione.

Frasi relative a Offerta

Gli **operatori aziendali** interagiscono periodicamente con i clienti per proporre nuove **offerte commerciali**, come sconti su prodotti, saldi o promozioni particolari. Queste offerte sono di varia natura e quindi il sistema deve permettere al **reparto amministrativo** di fornire una breve descrizione per ciascuna.

Se un cliente accetta un'offerta, questa deve essere registrata nel sistema, indicando quale operatore ha permesso l'accettazione e in quale […].

Frasi relative a Operatore

Gli operatori contattano periodicamente i clienti, di cui raccolgono **nome, cognome, data di nascita, codice fiscale e indirizzo di residenza**

Gli **operatori aziendali** interagiscono periodicamente con i clienti per proporre nuove **offerte commerciali**, come sconti su prodotti, saldi o promozioni particolari. Poiché lo stesso cliente può essere contattato da più operatori, il sistema deve prevedere un meccanismo per registrare il **feedback dell'interazione/note dell'interazione**, in cui gli operatori annotano dettagli sul risultato della comunicazione. Gli operatori possono consultare l'elenco delle **note dell'interazione** di un cliente durante il contatto.

Se un cliente accetta un'offerta, questa deve essere registrata nel sistema, indicando quale operatore ha permesso l'accettazione e in quale data.

Frasi relative a Segreteria
Queste offerte sono di varia natura e quindi il sistema deve permettere alla segreteria di fornire una breve descrizione per ciascuna. La segreteria può, in qualsiasi momento, generare un report che mostri, in un intervallo temporale specificato, quanti clienti sono stati contattati e con quale frequenza, nonché quante **offerte commerciali** sono state accettate da ciascuno.

Frasi relative a Appuntamento
Alla **nota dell'interazione** deve essere allegata la registrazione di un appuntamento, indicando la sede e la data/ora dell'incontro.

Frasi relative a Nota
meccanismo per registrare il **feedback dell'interazione/note dell'interazione**, in cui gli operatori annotano dettagli sul risultato della comunicazione. Gli operatori possono consultare l'elenco delle **note dell'interazione** di un cliente durante il contatto. Se un cliente accetta un'offerta, questa deve essere registrata nel sistema, indicando quale operatore ha permesso l'accettazione e in quale data. Inoltre, un cliente potrebbe richiedere un appuntamento in una delle sedi aziendali. In tal caso, alla **nota dell'interazione** deve essere allegata la registrazione di un appuntamento, indicando la sede e la data/ora dell'incontro.

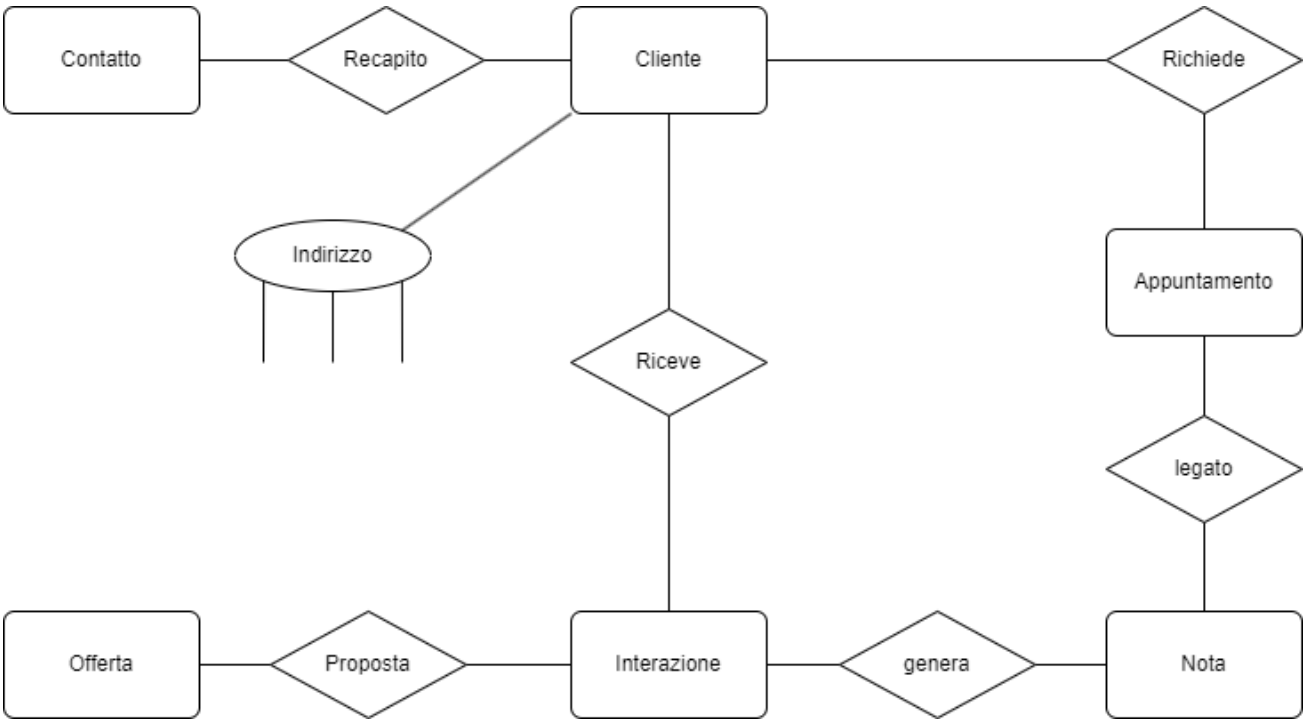
3. Progettazione concettuale

Costruzione dello schema E-R

Per la costruzione del seguente modello concettuale entity-relationship è stata adottata una strategia mista. Sono state identificate le entità fondamentali:

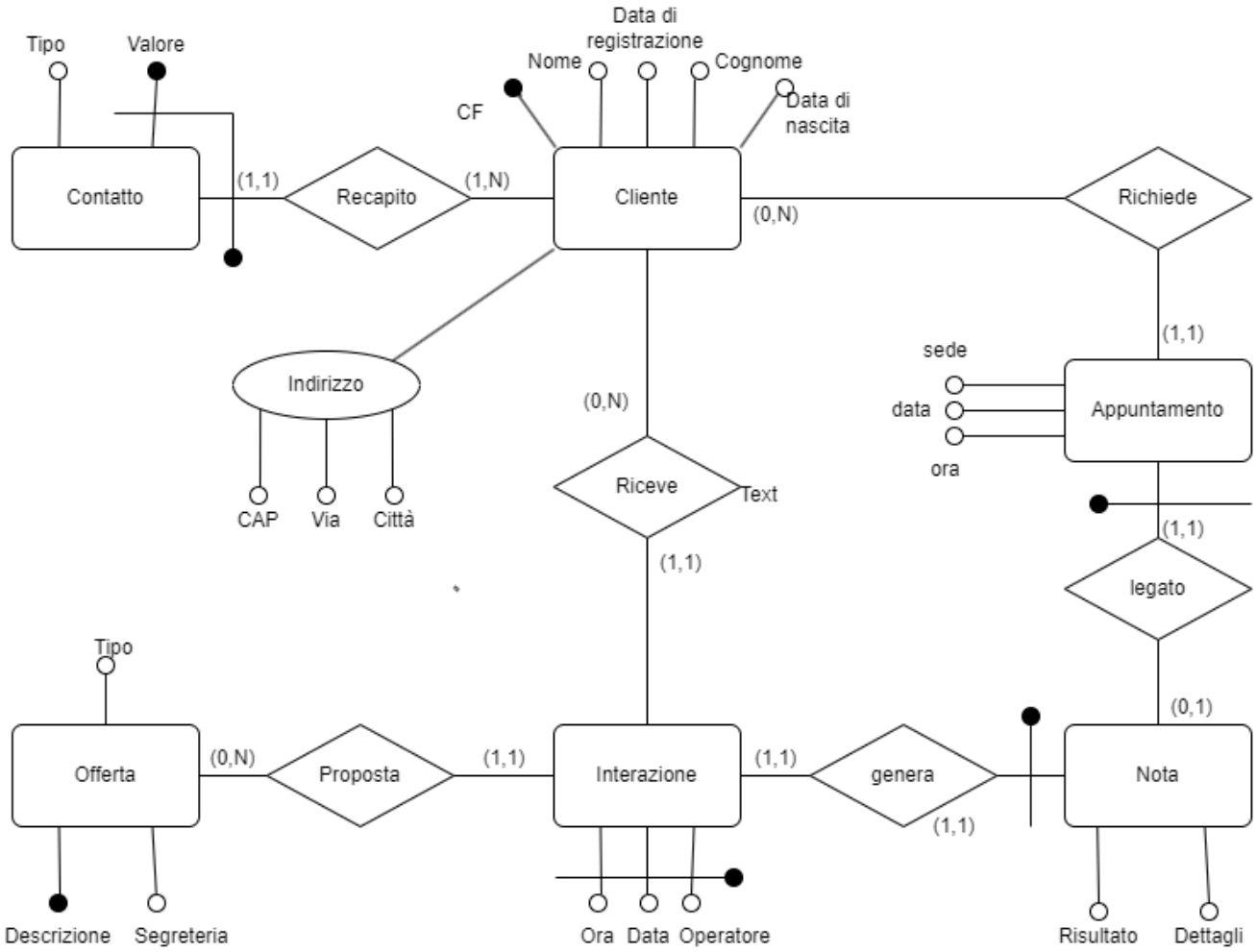
- Cliente: rappresenta i soggetti che l'azienda contatta per proporre offerte commerciali.
- Interazione: rappresenta un evento in cui l'azienda contatta un cliente attraverso uno dei suoi recapiti.
- Offerta: una proposta commerciale che l'azienda presenta.
- Nota: può contenere dettagli sull'esito dell'interazione, come l'accettazione o il rifiuto di un'offerta.
- Appuntamento: evento richiesto o meno dal cliente.

Ho voluto rappresentare contatto come entità e non attributo multivalore, è una ristrutturazione tipica della progettazione logica, ma che per comodità è stata adottata fin da subito. In seguito sono state aggiunte le relazioni, così creando uno schema scheletrico. (bottom-up).



È stato aggiunto l'attributo composto ***indirizzo***, essendo stato dichiarato nella specifica che l'azienda operi in più città e l'indirizzo è di residenza, cosicché posso evitare ambiguità con altri clienti.
Da qui ho proceduto con una strategia inside out, ottenuti alcuni concetti importanti, si continua aggiungendo attributi, identificatori e cardinalità. Non sono stati riscontrati particolari conflitti.

Integrazione finale



Regole aziendali

1. Un cliente può avere più contatti e indirizzi.
2. Un cliente può avere più interazioni e appuntamenti.
3. Un'offerta può essere proposta durante più interazioni.
4. Le date di registrazione non possono essere precedenti alla data di nascita del cliente.
5. Le date e gli orari degli appuntamenti devono essere coerenti con l'orario di lavoro.
6. L’azienda ha la sede in una sola città ma telefonicamente può operare in più citta.
7. Un’offerta non può essere proposta allo stesso cliente se già accettata.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Contatto	Certo numero di contatti telefonici e email.	Tipo, valore	Valore, cliente

Cliente	Le informazioni anagrafiche del cliente a cui è proposta l'offerta.	Data registrazione, data nascita, nome, cognome, indirizzo, codice fiscale	Codice fiscale
Appuntamento	Incontro richiesto dal cliente.	Sede, data, ora.	Interazione, nota
Offerta	Proposta dall'azienda al cliente, che può essere di varia natura.	Descrizione.	Descrizione
Interazione	Operazione effettiva che viene svolta	Ora, data, operatore	Operatore, ora, data
Nota	Risultato dell'operazione, con risultato	Risultato, dettagli	Interazione

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Contatti	E	20.000 se ogni cliente ha in media 2 recapiti
Cliente	E	10.000
Appuntamento	E	1.000
Nota	E	1.000 al giorno
Interazione	E	1.000 al giorno
Offerta	E	5.000
Recapito	R	20.000
Richiede	R	1.000 se il 10% richiede un appuntamento
Genera	R	1.000 al giorno
Legato	R	1.000 al giorno
Riceve	R	1.666 Ogni cliente riceve mensilmente 5 interazioni,
Proposta	R	15.000 ogni offerta viene proposta tramite 3 interazioni in media

Tavola delle operazioni

Sono state considerate le operazioni con frequenza giornaliera. Con un numero di operatori uguale a 50 e impiegati di segreteria 10.

Cod.	Descrizione	Frequenza attesa
S1	Inserisci i dati del cliente	50
S2	Inserisci Offerte	20
S3	Report clienti	10
S4	Mostra clienti e contatti	200
S5	Aggiorna recapiti e indirizzi	30
S6	Mostra appuntamenti	20
S7	Mostra offerte	150
OP2	Scrivi note	500
OP3	Report note	5
OP4	Aggiungi appuntamenti	50
OP5	Mostra clienti	100
OP6	Mostra offerte	100
OP7	Mostra appuntamenti	100
L1	Login	200
L2	Aggiungi utente	4 mensilmente

[1] Indicare con E le entità, con R le relazioni

Costo delle operazioni

- S1 inserisci i dati del cliente

Entità/Relazione	Volume totale	Accessi(giorno)	Costo	Totale costo
Cliente	10.000	2500	2	5.000
Recapito	20.000	5000	2	10.000

Totale accessi: 7.500

Costo totale S1: 10.000

- S2 Inserisci Offerte

Entità/Relazione	Volume totale	Accessi(giorno)	Costo	Totale costo
Offerte	5.000	20	2	20 ×2=40

Totale accessi: 20

Costo totale: 40

- S3 Report client

Entità/Relazione	Volume totale	Accessi(giorno)	Costo	Totale costo
Offerte	5.000	20	1	20
Interazione	1.000	1.000	1	1000
Note	1.000	1.000	1	1000
Cliente	10.000	2500	1	2500
Riceve	50.000	1.666	1	1666
Proposta	15.000	1.500	1	1.500
Genera	1.000	1.000	1	1.000

Totale accessi: 8686

Costo totale: 8686

- S4 Mostra clienti e contatti

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Cliente	10.000	2500	1	2500
Recapito	20.000	5000	1	5000
Contatto	20.000	5000	1	5000

Totale accessi: 12.500

Costo totale: 12.500

- S5 Aggiorna contatti

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Contatti	20.000	20	2	20 ×2=40

Totale accessi: 20

Costo totale: 40

- S6 Mostra appuntamenti

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Appuntamenti	1.000	20	1	20
Cliente	10.000	100	1	100
Richiede	1.000	1.000	1	1.000

Totale accessi: 1.120

Costo totale: 1.120

- S7 Mostra offerte

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Offerte	5.000	20	1	20

Totale accessi: 20

Costo totale: 20

- OP2 Scrivi note

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Interazione	1.000	500	2	1.000
Genera	1.000	500	2	1.000
Note	1.000	500	2	1.000

Totale accessi: 1.500

Costo totale: 3.000

- OP3 Report note

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Note	1.000	250	1	250
Interazione	1.000	250	1	250
Cliente	10.000	100	1	100
Riceve	50.000	1.666	1	1.666
Genera	1.000	250	1	250

Totale accessi: 2.516

Costo totale: 2.516

- OP4 Aggiungi appuntamenti

Entità/Relazione	Volume totale	Accessi(giorno)	Costo(W)	Totale costo
Appuntamenti	1.000	20	2	20 ×2=40
Nota	1.000	200	1	200
Interazione	1.000	200	1	200
Genera	1.000	200	1	200
Legato	1.000	200	1	200

Totale accessi: 820

Costo totale: 840

- OP5 Mostra clienti e contatti

Come per S4

- OP6 Mostra offerte

Come per S7

– OP7 Mostra appuntamenti
  Come per S6

Ristrutturazione dello schema E-R

Per la ristrutturazione dello schema concettuale, è stato eliminato l'attributo composto indirizzi e sostituito con una entità che ha una cardinalità uno a uno con cliente, essendo riferito alla residenza. Gli identificatori saranno via, c.a.p. e città con rispettivo cliente. Non sono presenti generalizzazioni.
Per gli identificatori sono stati introdotti:

- idinterazione
- idofferta
- idappuntamento

Si può notare dal costo delle operazioni, che l'entità *nota* può portare eventuali ridondanze. Può essere sovrapposta in molti casi all'entità interazione e se eliminata porta una riduzione notevole nel costo delle operazioni. Vogliamo quindi risparmiare gli accessi necessari per risalire ai dati dell'*interazione* e della *nota,* attraverso la relazione che li lega. L'accorpamento di entità può essere effettuato perché è una associazione uno a uno, se fosse stato uno a molti avremmo creato la possibilità di valori nulli.
Inoltre ho aggiunto l'attributo segreteria nell'entità *cliente* per tenere traccia degli impiegati della segreteria che inseriscono i clienti.
APPUNTAMENTO (idappuntamento, sede, data, ora, interazione, cliente)
INTERAZIONI (idinterazione, cliente, offerta, operatore, data, ora, risultato, dettagli)
OFFERTA (idofferta, descrizione, tipo)
INDIRIZZI (via, c.a.p., città, cliente)
CONTATTI (valore, cliente, tipo)
CLIENTE (codicefiscale, nome, cognome, dataDiNascita, dataDiRegistrazione, segreteria, indirizzi)
Vincoli:
APPUNTAMENTO (cliente)$\subseteq$ CLIENTE (codicefiscale)
APPUNTAMENTO (interazione)$\subseteq$ INTERAZIONE (idinterazione)
INTERAZIONE (cliente) $\subseteq$ CLIENTE (codicefiscale)
INDIRIZZI (cliente) $\subseteq$ CLIENTE (codicefiscale)
CONTATTI (cliente) $\subseteq$ CLIENTE (codicefiscale)

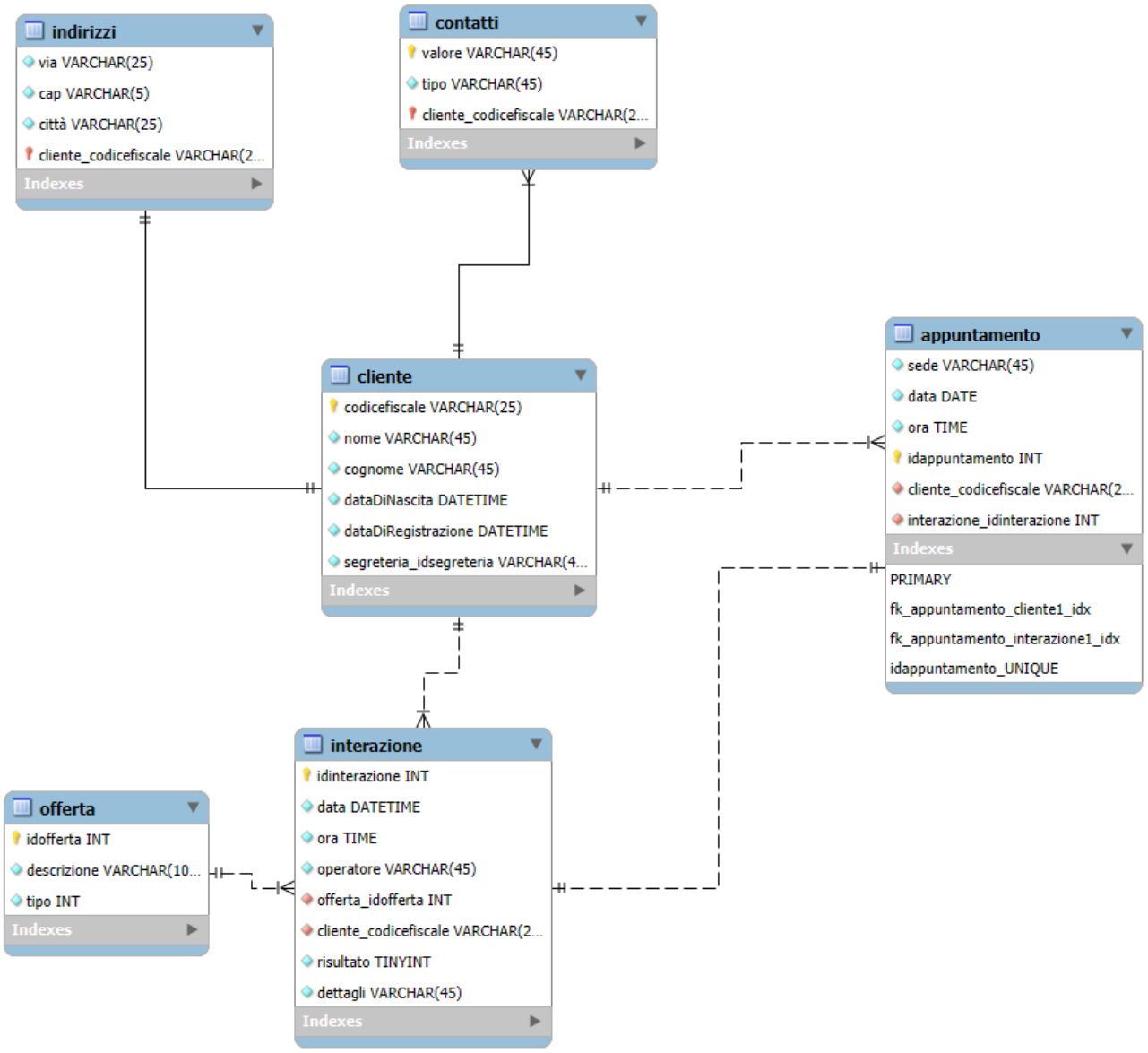
Trasformazione di attributi e identificatori

Come parte della progettazione, sono stati introdotti identificatori univoci per le entità *interazione* e *appuntamento*. Questo consente di evitare che siano legati esclusivamente alla chiave primaria del cliente, riducendo la ripetizione dell'attributo *cliente* nelle entità correlate, come *appuntamento*. Questa trasformazione segue i principi della normalizzazione, che mirano a eliminare ridondanze e dipendenze funzionali inappropriate.

Gli attributi *data* e *ora*, pur presenti sia in *interazione* che in *appuntamento*, non rappresentano una ripetizione problematica. Essi sono specifici per ciascuna entità e svolgono ruoli distinti (ad esempio, la data di un'interazione è diversa dalla data di un appuntamento). Pertanto, vengono mantenuti nel formato attuale.

Inoltre, eventuali identificatori esterni (chiavi esterne) sono stati aggiunti per rappresentare correttamente le relazioni tra entità nello schema relazionale, garantendo l'integrità referenziale e una traduzione fedele dal modello concettuale.

Traduzione di entità e associazioni



Normalizzazione del modello relazionale

Il modello relazionale è sia in 3NF che BCNF, in quanto non ci sono dipendenze parziali, transitive o violazioni dei requisiti di superchiave. Ogni tupla ha una chiave che la identifica univocamente.

La combinazione idofferta e descrizione non è necessaria, per questo motivo viene usata la prima come chiave primaria (intero univoco auto incrementale), e per la seconda viene impostato un vincolo di unicità per garantire che ogni descrizione sia unica all'interno della tabella.

Stessa cosa vale per indirizzi, via, cap e città vengono impostati unici

Per quanto riguarda valore e codicefiscale per contatti viene mantenuto per motivi prestazionali.

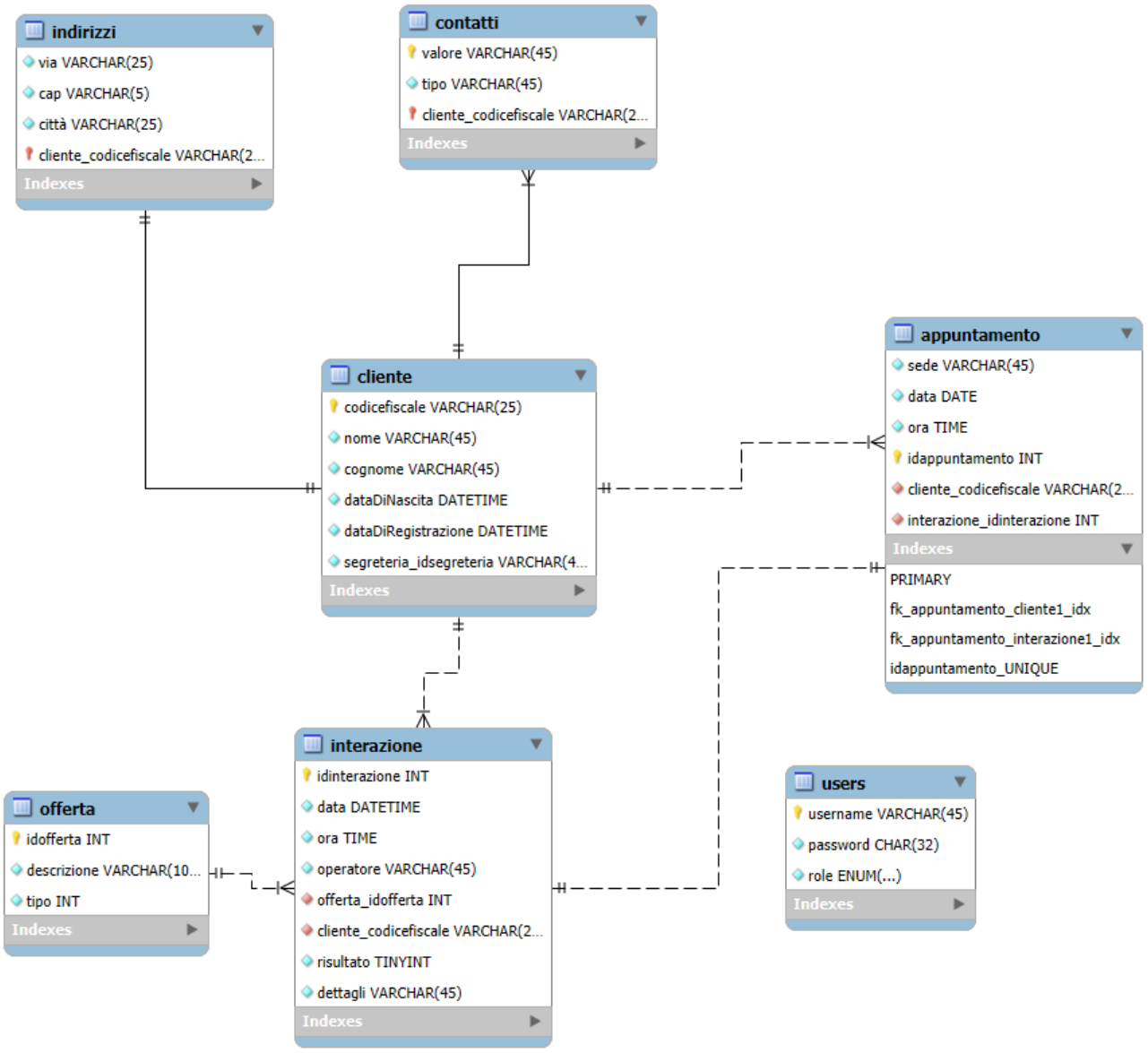
5. Progettazione fisica

Utenti e privilegi

Sono stati identificati tre ruoli all'interno della base di dati, per implementare il Principle of Least Privilege.

- Login: grant in esecuzione sull'operazione L1 e L2;
- Operatore: grant in esecuzione sulle operazioni OP1, OP2, OP3, OP4, OP5, OP6, OP7;
- Segreteria: grant in esecuzione sulle operazioni S1, S2, S3, S4, S5, S6;

è stata introdotta una tabella users per mantenere le credenziali.



Grant per login:

GRANT EXECUTE ON PROCEDURE crm_basedidati.login TO 'login'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.register TO 'login'@'localhost';

GRANT INSERT, DELETE ON users TO 'login'@'localhost';

Username=login password=login

Grant per segreteria:

GRANT insert,update,select,delete on cliente TO 'segreteria'@'localhost';

GRANT insert,update,select,delete on indirizzi TO 'segreteria'@'localhost';

GRANT insert,update,select,delete on contatti TO 'segreteria'@'localhost';

GRANT insert,update,select,delete on offerta TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.insertCustomer TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.insertPhone TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.insertEmail TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.insert_offer TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.reportCustomers TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.updateIndirizzi TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.updateContacts TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.searchContact TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.customer TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.getOffers TO 'segreteria'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.getAppointment TO 'segreteria'@'localhost';

Username=segreteria password=segreteria

Grant per operatore:

GRANT insert,update,select,delete on appuntamento TO 'operatore'@'localhost';

GRANT insert,update,select,delete on interazione TO 'operatore'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.writeNote TO 'operatore'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.insertAppointment TO 'operatore'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.reportNotes TO 'operatore'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.customer TO 'operatore'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.getOffers TO 'operatore'@'localhost';

GRANT EXECUTE ON PROCEDURE crm_basedidati.getAppointment TO 'operatore'@'localhost';

Strutture di memorizzazione

Tabella <users>		
Colonna	**Tipo di dato**	**Attributi[2]**
username	VARCHAR(45)	PK, NN
password	CHAR(32)	NN
role	ENUM('SEGRETERIA','OPERATORE')	NN

Tabella <contatti>		
Colonna	**Tipo di dato**	**Attributi[3]**
valore	VARCHAR(45)	PK,NN,UQ
Cliente_codicefiscale	VARCHAR(25)	PK,NN
tipo	VARCHAR(45)	NN

Tabella <indirizzi>		
Colonna	**Tipo di dato**	**Attributi[4]**
cliente_codicefiscale	VARCHAR(25)	PK,NN
via	VARCHAR(25)	NN, UQ
cap	VARCHAR(5)	NN, UQ
città	VARCHAR(25)	NN,UQ

Tabella <cliente>		
Colonna	**Tipo di dato**	**Attributi[5]**
codicefiscale	VARCHAR(25)	PK,NN,UQ
nome	VARCHAR(45)	NN
cognome	VARCHAR(45)	NN
dataDiNascita	DATETIME	NN
dataDiRegistrazione	DATETIME	NN
segreteria	VARCHAR(45)	NN

Tabella <interazione>		
Colonna	**Tipo di dato**	**Attributi[6]**

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[3] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[4] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[5] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

idinterazione	INT	PK, NN, UQ, AI
data	DATETIME	NN
ora	TIME	NN
operatore	VARCHAR(45)	NN
risultato	TINYINT	NN
dettagli	VARCHAR(45)	NN
offerta_idofferta	INT	NN
cliente_codicefiscale	VARCHAR(25)	NN

Tabella <offerta>		
Colonna	**Tipo di dato**	**Attributi[7]**
idofferta	INT	PK,NN,AI,UQ
descrizione	VARCHAR(100)	NN
tipo	INT	NN

Tabella <appuntamento>		
Colonna	**Tipo di dato**	**Attributi[8]**
idappuntamento	INT	PK,AI,NN
cliente_codicefiscale	VARCHAR(25)	NN
interazione_idinterazione	INT	NN
sede	VARCHAR(45)	NN
data	DATE	NN
ora	TIME	NN

Indici

Ad esclusione degli indici autogenerati per le chiavi primarie e per le foreign key, sono stati introdotti degli indici unique all'interno delle tabelle contatti e indirizzi . Usati perché non ci siano duplicati all'interno delle tabelle, cosicché i dati riferiti al cliente siano sempre associati ad uno ed un solo interessato.

[6] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[7] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[8] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

La creazione dell'indice idx_interazione_cliente_offerta sulla tabella interazione è motivata dal desiderio di ottimizzare le prestazioni del trigger e procedure che utilizzano frequentemente i campi cliente_codicefiscale, offerta_idofferta e risultato.

Tabella <contatti>	
Indice <valore>	**Tipo[9]:**
valore_UNIQUE	UNIQUE

Tabella <indirizzi>	
Indice <via>	**Tipo[10]:**
via_UNIQUE	UNIQUE
Indice <città>	**Tipo[11]:**
città_UNIQUE	UNIQUE
Indice <cap>	**Tipo[12]:**
cap_UNIQUE	UNIQUE

Tabella <interazione>	
Indice <cliente_offerta>	**Tipo[13]:**
idx_interazione_cliente_offerta	IDX

Trigger

1. Il trigger seguente serve per specificare la data di registrazione del cliente prima dell'inserimento ed impostarla alla data odierna.

```
DELIMITER $$
CREATE TRIGGER ImpostaDataDiRegistrazione
BEFORE INSERT ON cliente
FOR EACH ROW
BEGIN
    -- Imposta la data corrente se non specificata
```

[9] IDX = index, UQ = unique, FT = full text, PR = primary.

[10] IDX = index, UQ = unique, FT = full text, PR = primary.

[11] IDX = index, UQ = unique, FT = full text, PR = primary.

[12] IDX = index, UQ = unique, FT = full text, PR = primary.

[13] IDX = index, UQ = unique, FT = full text, PR = primary.

```
    IF NEW.dataDiRegistrazione IS NULL THEN
        SET NEW.dataDiRegistrazione = CURRENT_DATE;
    END IF;
END$$
DELIMITER ;
```

2. Controllo dell' inserimento per la data dell'interazione, che non sia antecedente alla registrazione del cliente e che non possa essere futura.

```
DELIMITER $$
CREATE TRIGGER VerificaDateInterazione
BEFORE INSERT ON interazione
FOR EACH ROW
BEGIN
    DECLARE dataRegistrazione DATE;

    -- Recupera la data di registrazione del cliente
    SELECT dataDiRegistrazione
    INTO dataRegistrazione
    FROM cliente
    WHERE codicefiscale = NEW.cliente_codicefiscale;

    -- Verifica che la data dell'interazione non sia futura
    IF NEW.data > CURRENT_DATE THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La data di interazione non può essere futura.';
    END IF;

    -- Verifica che la data dell'interazione non sia precedente alla registrazione del cliente
    IF NEW.data < dataRegistrazione THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La data di interazione non può essere antecedente alla registrazione del cliente.';
    END IF;
END$$
DELIMITER ;
```

3. Imposto l’ora dell’interazione se nulla.

```
DELIMITER $$
CREATE TRIGGER ImpostaOraInterazione
BEFORE INSERT ON interazione
FOR EACH ROW
BEGIN
   IF NEW.ora IS NULL THEN
      SET NEW.ora = HOUR(CURRENT_TIME());
   END IF;
END$$
DELIMITER ;
-- Se l'ora di un'interazione non viene specificata
```

4. Introdotto perché, un appuntamento non possa essere allegato se non è avvenuta/inserita un’interazione.

```
DELIMITER $$

CREATE TRIGGER checkAppuntamento
BEFORE INSERT ON appuntamento
FOR EACH ROW
BEGIN
   -- Verifica che esista un'interazione con l'ID fornito
   IF NOT EXISTS (SELECT 1 FROM interazione WHERE idinterazione = NEW.interazione_idinterazione) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Impossibile inserire un appuntamento senza un interazione';
   END IF;
END$$

DELIMITER ;
```

5. Questo trigger è stato inserito per evitare che ci siano appuntamenti nella stessa data, ora e sede con lo stesso cliente.

```
DELIMITER $$
CREATE TRIGGER VerificaConflittiAppuntamenti
```

```
BEFORE INSERT ON appuntamento
FOR EACH ROW
BEGIN
  IF EXISTS (
    SELECT 1
    FROM appuntamento
    WHERE cliente_codicefiscale = NEW.cliente_codicefiscale
    AND data = NEW.data
    AND ora = NEW.ora
    AND sede = NEW.sede
  ) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Conflitto di appuntamento: un cliente non può avere più appuntamenti nella stessa sede, data e ora.';
  END IF;
END$$
DELIMITER ;
-- impedire che un cliente abbia più appuntamenti nello stesso orario e nella stessa sede.
```

6. Ho introdotto questo trigger, per evitare che lo stesso operatore abbia un appuntamento già fissato con il cliente in futuro.

```
DELIMITER $$

CREATE TRIGGER check_duplicate_appointment
BEFORE INSERT ON appuntamento
FOR EACH ROW
BEGIN
  DECLARE var_conta_appuntamenti INT;

  -- Controlla se esiste già un appuntamento non scaduto per lo stesso cliente e operatore
  SELECT COUNT(*) INTO var_conta_appuntamenti
  FROM appuntamento
  WHERE cliente_codicefiscale = NEW.cliente_codicefiscale
    AND interazione_idinterazione = NEW.interazione_idinterazione
    AND data >= CURDATE();  -- Assicurati che l'appuntamento non sia scaduto
```

```
    -- Se esiste già un appuntamento non scaduto, impedisce l'inserimento
    IF var_conta_appuntamenti > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Errore: L\'operatore ha già un appuntamento con il cliente che non è scaduto.';
    END IF;
END$$
```

7. Verifico prima di inserire un'offerta che la sua descrizione non sia già presente quindi evito duplicati.

```
DELIMITER $$
CREATE TRIGGER prevent_duplicate_offer
BEFORE INSERT ON offerta
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1
        FROM offerta
        WHERE descrizione=NEW.descrizione
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Non si posso inserire offerte con la stessa descrizione.';
    END IF;
END$$
DELIMITER ;
```

8. Verifico prima di inserire gli utenti riferiti a segreteria e operatore che username e password abbiano un formato corretto.

```
DELIMITER $$

CREATE TRIGGER VerificaUsernamePassword
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    -- Verifica che lo username sia univoco
    IF EXISTS (
```

```
        SELECT 1
        FROM users
        WHERE username = NEW.username
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Lo username deve essere univoco.';
    END IF;

    -- Verifica che lo username rispetti il formato
    IF NEW.username NOT REGEXP '^[a-zA-Z0-9._]{3,45}$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Lo username deve avere tra 3 e 45 caratteri e può contenere
solo lettere, numeri, punti e underscore.';
    END IF;

    -- Verifica la lunghezza della password
    IF CHAR_LENGTH(NEW.password) < 8 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La password deve avere almeno 8 caratteri.';
    END IF;



END$$

DELIMITER ;
```

9. Questi due trigger compiono lo stesso compito, cioè controllare la correttezza dei dati prima dell'inserimento nella tabella indirizzi, solamente che uno si occupa dell'update e l'altro dell'inserimento vero e proprio.

```
DELIMITER $$

CREATE TRIGGER VerificaIndirizziUpdate
BEFORE UPDATE ON indirizzi
FOR EACH ROW
BEGIN
```

```
    -- Verifica che la via sia valida
    IF NEW.via NOT REGEXP '^[a-zA-Z0-9 ]+$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato via non valido. Può contenere solo lettere, numeri e
spazi.';
    END IF;

    -- Verifica che la città sia valida
    IF NEW.città NOT REGEXP '^[a-zA-Z ]+$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato città non valido. Può contenere solo lettere e spazi.';
    END IF;

    -- Verifica che il CAP sia valido (solo numeri, 5 cifre)
    IF NEW.cap NOT REGEXP '^[0-9]{5}$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato CAP non valido. Deve essere composto da 5 cifre.';
    END IF;
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE TRIGGER VerificaIndirizzi
BEFORE INSERT ON indirizzi
FOR EACH ROW
BEGIN
    -- Verifica che la via sia valida
    IF NEW.via NOT REGEXP '^[a-zA-Z0-9 ]+$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato via non valido. Può contenere solo lettere, numeri e
spazi.';
    END IF;
```

```
    -- Verifica che la città sia valida
    IF NEW.città NOT REGEXP '^[a-zA-Z ]+$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato città non valido. Può contenere solo lettere e spazi.';
    END IF;

    -- Verifica che il CAP sia valido (solo numeri, 5 cifre)
    IF NEW.cap NOT REGEXP '^[0-9]{5}$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato CAP non valido. Deve essere composto da 5 cifre.';
    END IF;
END$$

DELIMITER ;
```

10. Come i precedent trigger, questi verificano la correttezza dell'inserimento per il formato email e numero di telefono per la tabella contatti, nel caso update e insert.

```
DELIMITER $$

CREATE TRIGGER VerificaEmailTelefonoUpdate
BEFORE UPDATE ON contatti
FOR EACH ROW
BEGIN
    -- Verifica l'email
    IF NEW.tipo = 'email' AND NEW.valore NOT REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Formato email non valido. Assicurati di usare un formato corretto.';
    END IF;

    -- Verifica il telefono
    IF NEW.tipo = 'telefono' AND NEW.valore NOT REGEXP '^\\+?[0-9]{8,15}$' THEN
        SIGNAL SQLSTATE '45000'
```

```
    SET MESSAGE_TEXT = 'Formato telefono non valido. Deve contenere solo numeri (opzionalmente un + all'inizio) e deve essere lungo tra 8 e 15 cifre.';
  END IF;
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE TRIGGER VerificaEmailTelefono
BEFORE INSERT ON contatti
FOR EACH ROW
BEGIN
  -- Verifica l'email
  IF NEW.tipo = 'email' AND NEW.valore NOT REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Formato email non valido. Assicurati di usare un formato corretto.';
  END IF;

  -- Verifica il telefono
  IF NEW.tipo = 'telefono' AND NEW.valore NOT REGEXP '^\\+?[0-9]{8,15}$' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Formato telefono non valido. Deve contenere solo numeri (opzionalmente un + all'inizio) e deve essere lungo tra 8 e 15 cifre.';
  END IF;
END$$

DELIMITER ;
```

11. Questo trigger è stato creato per non permettere l'inserimento nell'interazione di un'offerta già proposta e accettata da un cliente.

```
DELIMITER $$
```

```
CREATE TRIGGER offerta_proposta_accettata
BEFORE INSERT ON interazione
FOR EACH ROW
BEGIN
   -- Controlla se l'offerta è già stata proposta al cliente
   IF EXISTS (
      SELECT 1
      FROM interazione
      WHERE cliente_codicefiscale = NEW.cliente_codicefiscale
        AND offerta_idofferta = NEW.offerta_idofferta AND risultato=1
   ) THEN
      -- Segnala un errore
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Errore: L\'offerta è già stata proposta a questo cliente ed è stata accettata.';
   END IF;
END $$

DELIMITER ;
```

12. Questi due trigger sono stati inseriti per rispettare le regole aziendali, gli orari per interazioni e appuntamenti devono essere coerenti rispetto agli orari e i giorni lavorativi.

```
DELIMITER $$
CREATE TRIGGER check_orari_interazione
BEFORE INSERT ON interazione
FOR EACH ROW
BEGIN
   -- Controlla che la data e l'ora siano coerenti con gli orari lavorativi
   IF WEEKDAY(NEW.data) > 4 OR
     TIME(NEW.ora) < '09:00:00' OR
     TIME(NEW.ora) > '18:00:00' THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Errore: giorno e ora non sono conformi agli orari e ai giorni di lavoro.' ;
   END IF;
```

```
END$$

DELIMITER ;
```

```
DELIMITER $$
CREATE TRIGGER check_orari_appuntamento
BEFORE INSERT ON appuntamento
FOR EACH ROW
BEGIN
   -- Controlla che la data e l'ora siano coerenti con gli orari lavorativi
   IF WEEKDAY(NEW.data) > 4 OR
      TIME(NEW.ora) < '09:00:00' OR
      TIME(NEW.ora) > '18:00:00' THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Errore: La data e l\'ora devono rientrare negli orari lavorativi
(lun-ven, 9:00-18:00).';
   END IF;
END$$

DELIMITER $$
```

Eventi

1. Questo evento elimina gli appuntamenti più vecchi di 7 giorni dalla data corrente

   ```
   CREATE EVENT appuntamentiScaduti
   ON SCHEDULE EVERY 1 DAY
   STARTS CURRENT_TIMESTAMP
   DO
   DELETE FROM appuntamento
   WHERE data < CURDATE() - INTERVAL 7 DAY;
   ```

2. Elimina le interazioni più vecchie di due anni

   ```
   DELIMITER $$
   CREATE EVENT PulisciInterazioniVecchie
   ON SCHEDULE EVERY 1 MONTH
   DO
   BEGIN
   ```

```
    DELETE FROM interazione WHERE data < DATE_SUB(CURRENT_DATE, INTERVAL 2 YEAR);
    END$$
    DELIMITER ;
    -- Se le interazioni sono più vecchie di due anni vengono eliminate
```

3. Notifico l'imminenza di un appuntamento

```
    DELIMITER $$
    CREATE EVENT NotificaAppuntamenti
    ON SCHEDULE EVERY 1 DAY
    DO
    BEGIN
      -- Aggiorna lo stato degli appuntamenti imminenti
      UPDATE appuntamento
      SET dettagli = CONCAT('Promemoria: appuntamento imminente il ', data, ' alle ', ora)
      WHERE data = CURRENT_DATE + INTERVAL 1 DAY;
    END$$
    DELIMITER ;
    -- Notifca l'imminenza di un appuntamento
```

4.Elimino I client che non hanno avuto interazioni negli ultimi 3 anni.

```
    DELIMITER $$
    CREATE EVENT CleanOldCustomers
    ON SCHEDULE EVERY 1 DAY
    STARTS '2025-01-01 02:00:00'
    DO
    BEGIN
      -- Elimina i clienti che non hanno avuto interazioni negli ultimi 3 anni
      DELETE FROM cliente
      WHERE cliente_codicefiscale NOT IN (
        SELECT DISTINCT cliente_codicefiscale
        FROM interazione
        WHERE data_interazione >= CURDATE() - INTERVAL 3 YEAR
      );
    END$$
```

```
DELIMITER ;
```

Viste

Non sono state introdotte viste.

Stored Procedures e transazioni

1. Procedura di login per utenti

```
DELIMITER $$
CREATE PROCEDURE login(
   IN var_username VARCHAR(45),
   IN var_pass CHAR(32),
   OUT var_role INT
)
BEGIN
   DECLARE var_user_role ENUM('SEGRETERIA','OPERATORE');

   -- Seleziona il ruolo dell'utente dalla tabella users
   SELECT role
   INTO var_user_role
   FROM users
   WHERE username = var_username AND `password` = md5(var_pass);
```

```
                    -- Assegna il valore del ruolo all'output var_role
                    if var_user_role='SEGRETERIA' then
                            set var_role=1;
        elseif var_user_role='OPERATORE' then
                            set var_role=2;
        else
```

```
                    set var_role=3; -- Se il ruolo non è riconosciuto
                end if;

END$$
DELIMITER ;
```

2. Procedura di registrazione da parte dell'utente login per gli utenti

```
DELIMITER $$

CREATE PROCEDURE register(
    IN var_username VARCHAR(50),
    IN var_password VARCHAR(255),
    IN var_role INT,
    OUT result INT
)
BEGIN
    -- Variabili per il risultato e l'errore
    DECLARE ruolo VARCHAR(50);
    DECLARE error_message VARCHAR(255);

    -- Gestione degli errori
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Log dettagliato dell'errore
        GET DIAGNOSTICS CONDITION 1
            error_message = MESSAGE_TEXT;

        -- Se l'errore non contiene un messaggio, usa un messaggio di fallback
        IF error_message IS NULL THEN
            SET error_message = 'Errore sconosciuto durante la registrazione.';
        END IF;

        -- Rollback e invio del messaggio d'errore
        ROLLBACK;
```

```
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END;

    -- Verifica se il ruolo è valido
    IF var_role NOT IN (1, 2) THEN
        SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Errore: Ruolo non valido';
    END IF;

    -- Verifica se l'username è già presente
    IF EXISTS (
        SELECT 1
        FROM users
        WHERE username = var_username
    ) THEN
        SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'Errore: Username già in uso';
    END IF;

    -- Impostazione della password (MD5)
    SET var_password = MD5(var_password);

    -- Inizio della transazione
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  -- Usato un livello più performante rispetto a SERIALIZABLE
    START TRANSACTION;

    -- Determina il ruolo in base al parametro var_role
    IF var_role = 1 THEN
        SET ruolo = 'SEGRETERIA';
    ELSEIF var_role = 2 THEN
        SET ruolo = 'OPERATORE';
    ELSE
        SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'Ruolo errato';
    END IF;
```

```
    -- Inserimento del nuovo utente
    INSERT INTO users (username, password, role)
    VALUES (var_username, var_password, ruolo);

    -- Verifica dell'inserimento
    IF ROW_COUNT() = 1 THEN
        SET result = 1;  -- Registrazione riuscita
    ELSE
        ROLLBACK;
        SIGNAL SQLSTATE '45006' SET MESSAGE_TEXT = 'Errore: Impossibile registrare l\'utente';
    END IF;

    -- Conferma della transazione
    COMMIT;

    -- Restituzione del risultato
    SELECT result;

END $$

DELIMITER ;
```

3. Inserimento dei dati del cliente, nella tabella cliente e indirizzi. Ho usato un livello di isolamento per la transazione reapatable read, con un controllo se il cliente è già presente nella tabella.

```
DELIMITER $$

CREATE PROCEDURE insertCustomer(
    IN var_codiceFiscale VARCHAR(25),
    IN var_nome VARCHAR(45),
    IN var_cognome VARCHAR(45),
    IN var_dataDiNascita DATETIME,
    IN var_cap VARCHAR(5),
    IN var_address VARCHAR(255),
```

```
    IN var_city VARCHAR(45),
    IN var_segreteria VARCHAR(45)
)
BEGIN
    -- Dichiarazione di variabili locali
    DECLARE segreteria_count INT;
    DECLARE cliente_count INT;

    -- Gestione degli errori
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK; -- Annulla la transazione
        RESIGNAL; -- Propaga ulteriormente l'errore
    END;

    -- Setto il livello di isolamento
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  -- Livello di isolamento migliorato
    -- Inizio della transazione
    START TRANSACTION;



    -- Verifica se il cliente esiste già
    SELECT COUNT(*)
    INTO cliente_count
    FROM cliente
    WHERE codicefiscale = var_codiceFiscale;

    -- Se il cliente esiste già, lancia un errore
    IF cliente_count > 0 THEN
        SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Errore: Il cliente esiste già.';
    END IF;

    -- Inserimento del cliente
```

```
    INSERT INTO cliente (codicefiscale, nome, cognome, dataDiNascita, segreteria_idsegreteria)
    VALUES (var_codiceFiscale, var_nome, var_cognome, var_dataDiNascita, var_segreteria);

    -- Inserimento dell'indirizzo
    INSERT INTO indirizzi (via, cap, città, cliente_codicefiscale)
    VALUES (var_address, var_cap, var_city, var_codiceFiscale);

    -- Completamento della transazione
    COMMIT;
END$$

DELIMITER ;
```

4. Ho usato due procedure per inserire le email e i telefoni per la tabella contatti per distinguerli al meglio. Anche qui ho sempre usato un livello di isolamento repeatable read.

```
DELIMITER $$

CREATE PROCEDURE insertEmail(
    IN var_codiceFiscale VARCHAR(25),
    IN var_email VARCHAR(45)
)
BEGIN
        DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK; -- Annulla la transazione
        RESIGNAL; -- Propaga ulteriormente l'errore
    END;
        SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;
    IF var_email IS NOT NULL THEN
                INSERT INTO contatti (valore, tipo,cliente_codicefiscale)
                VALUES (var_email, 'email', var_codiceFiscale);
        ELSE
```

```
                SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = 'Errore: Email non trovata non valida';
            END IF;
    COMMIT;
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE PROCEDURE insertPhone(
    IN var_codiceFiscale VARCHAR(25),
    IN var_phone VARCHAR(45)
)
BEGIN
            DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK; -- Annulla la transazione
        RESIGNAL; -- Propaga ulteriormente l'errore
    END;
            SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;
    IF var_phone IS NOT NULL THEN
                INSERT INTO contatti (valore, tipo,cliente_codicefiscale)
                VALUES (var_phone, 'phone', var_codiceFiscale);
            ELSE
                SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = 'Errore: Numero di telefono non valido o non trovato';
            END IF;
    COMMIT;
END$$

DELIMITER ;
```

5. Procedure e transazione per inserire offerte. Una validazione preventiva dei parametri di input garantisce che l'operazione venga eseguita solo quando i dati sono validi, evitando di inserire valori inconsistenti nella tabella. Ho usato un livello di isolamento repeatable read, per garantire la consistenza dei dati durante l'esecuzione della transazione, evitando letture non ripetibili.

```
DELIMITER $$

CREATE PROCEDURE insert_offer(
   IN var_type_offer INT,
   IN var_description_offer VARCHAR(225)
)
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
      RESIGNAL;
   END;

   -- Imposta il livello di isolamento della transazione
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

   -- Inizia la transazione
   START TRANSACTION;

   -- Controlla che i parametri siano validi
   IF var_description_offer IS NULL OR var_description_offer = '' THEN
      SIGNAL SQLSTATE '45001'
      SET MESSAGE_TEXT = 'Errore: Descrizione dell\'offerta non valida.';
   ELSEIF var_type_offer IS NULL OR var_type_offer = '' THEN
      SIGNAL SQLSTATE '45002'
      SET MESSAGE_TEXT = 'Errore: Tipo offerta non valido.';
   ELSE
      -- Inserisci i dati nell'offerta
      INSERT INTO offerta (descrizione, tipo)
```

```
        VALUES (var_description_offer, var_type_offer);
    END IF;

    -- Commit della transazione
    COMMIT;
END $$

DELIMITER ;
```

6. Ultimo inserimento tra le procedure, quello di appuntamneti, con livello repeatable read, con ulteriore controllo della presenza di un'interazione. Ricordiamo che questi due concetti sono legati.

```
DELIMITER $$

CREATE PROCEDURE insertAppointment(
    IN var_data DATE,
    IN var_time TIME,
    IN var_cliente VARCHAR(25),
    IN var_branch VARCHAR(45),
    IN var_operatore VARCHAR(45)
)
BEGIN
    DECLARE var_id_inter INT;

    -- Gestione degli errori
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    -- Imposta il livello di isolamento e avvia la transazione
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  -- Livello di isolamento migliorato
    START TRANSACTION;
```

```
    -- Recupera l'ID dell'interazione per l'operatore e il cliente
    SELECT i.idinterazione
    INTO var_id_inter
    FROM interazione as i
    WHERE i.operatore = var_operatore
    AND i.cliente_codicefiscale = var_cliente
    AND var_data > i.data  -- Controllo della data
    ORDER BY i.data DESC, i.ora DESC
    LIMIT 1;

    -- Controlla se l'ID è valido
    IF var_id_inter IS NOT NULL THEN
        -- Inserisce il nuovo appuntamento (il controllo sugli appuntamenti esistenti è gestito dal trigger)
        INSERT INTO appuntamento (sede, data, ora, cliente_codicefiscale, interazione_idinterazione)
        VALUES (var_branch, var_data, var_time, var_cliente, var_id_inter);
    ELSE
        SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = 'Errore: Nessuna interazione valida trovata per il cliente.';
    END IF;

    -- Conferma la transazione
    COMMIT;
END$$

DELIMITER ;
```

7. Le procedure fornite di seguito saranno tre select per mostrare all'utente la presenza rispettivamente gli appuntamenti, dati del cliente e le offerte.
   Ho usato un livello di isolamento repeatable read perché protegge da modifiche concorrenti durante la lettura e garantisce che tutte le righe lette rimangano consistenti.
   Viene usato un gestore degli errori in caso di mancata connessione o errore nel recupero dei dati.

```
DELIMITER $$

CREATE PROCEDURE getAppointment()
BEGIN
        -- Handler per le eccezioni SQL
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
                ROLLBACK;
    RESIGNAL;
  END;

  -- Imposta il livello di isolamento per una transazione
  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

  -- Avvia la transazione
  START TRANSACTION;

  -- Seleziona le informazioni degli appuntamenti
  SELECT distinct
    a.sede AS Branch,
    a.ora AS Time,
    a.data AS Date,
    a.cliente_codicefiscale AS Customer,
    i.operatore AS Operator
  FROM appuntamento AS a
  JOIN interazione AS i
    ON a.cliente_codicefiscale = i.cliente_codicefiscale;


  COMMIT;
END$$
```

```
DELIMITER ;
```

```
DELIMITER $$

CREATE PROCEDURE getOffers()
BEGIN
        -- Handler per le eccezioni SQL
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
                ROLLBACK;
      RESIGNAL;
   END;
   -- Imposta il livello di isolamento
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

   -- Inizia la transazione
   START TRANSACTION;

   -- Recupera le offerte
   SELECT
      descrizione AS Descrizione,
      tipo AS Tipo
   FROM
      offerta;

   -- Conferma la transazione
   COMMIT;
END$$
DELIMITER ;
```

```
DELIMITER $$
CREATE PROCEDURE customer()
BEGIN
        -- Handler per le eccezioni SQL
```

```
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
                    ROLLBACK;
       RESIGNAL;
    END;
    -- Imposta il livello di isolamento della transazione
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
          -- Inizia una transazione
    START TRANSACTION;

         -- Seleziona i dati specifici
    SELECT
       c.codicefiscale AS cliente_codicefiscale,
       c.nome AS cliente_nome,
       c.cognome AS cliente_cognome,
       c.dataDiNascita AS cliente_data_nascita,
       c.dataDiRegistrazione AS cliente_data_di_registrazione,
       cont.valore AS contatto_valore,
       cont.tipo AS contatto_tipo,
       addr.via AS indirizzo_via,
       addr.città AS indirizzo_citta,
       addr.cap AS indirizzo_cap
    FROM
       cliente AS c
    JOIN
       contatti AS cont ON c.codicefiscale = cont.cliente_codicefiscale
          JOIN
       indirizzi AS addr ON c.codicefiscale = addr.cliente_codicefiscale
    ORDER BY c.cognome; -- Ordina per cognome
     -- Completa la transazione
    COMMIT;
END $$
DELIMITER ;
```

8. Ho creato questa procedura, per la ricerca dei contatti di un cliente. Pensata, con l'unico scopo pratico, dell'essere utilizza da parte degli operatori, prima di una interazione. È un'operazione di sola lettura, non modifica i dati e si limita a recuperare informazioni dalla tabella, quindi ho usato come livello di isolamento read committed.

```
DELIMITER $$

CREATE PROCEDURE searchContact(IN var_cliente VARCHAR(25))
BEGIN
  -- Validazione dell'input
  IF var_cliente IS NULL OR var_cliente = '' THEN
    SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Errore: Cliente non valido.';
  END IF;

  -- Livello di isolamento per l'operazione di lettura
  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

  -- Inizia la transazione
  START TRANSACTION;

  -- Query per recuperare i contatti
  IF EXISTS (
    SELECT 1
    FROM contatti
    WHERE cliente_codicefiscale = var_cliente
  ) THEN
    SELECT
      valore AS ContactValue,
      tipo AS Contact
    FROM
      contatti
    WHERE
      cliente_codicefiscale = var_cliente;
  ELSE
```

```
        SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Nessun contatto trovato per il cliente specificato.';
    END IF;

    -- Convalida della transazione
    COMMIT;
END $$
DELIMITER ;
```

9. Le due procedure sottostanti sono state create per l'aggiornamento di indirizzi e contatti. Nei casi in cui un cliente cambi indirizzo di residenza o recapiti. Un livello di isolamento meno restrittivo (come **REPEATABLE READ**) offre un buon equilibrio tra consistenza e velocità.

```
DELIMITER $$

CREATE PROCEDURE updateContacts(
    IN var_valore VARCHAR(45),
    IN var_tipo VARCHAR(45),
    IN var_cliente VARCHAR(25)
)
BEGIN
    -- Gestione degli errori
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    -- Validazione dei parametri
    IF var_valore IS NULL OR var_valore = '' THEN
        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Errore: Valore non valido.';
    END IF;

    IF var_tipo NOT IN ('email', 'phone') THEN
        SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Errore: Tipo non valido.';
    END IF;
```

```
    IF var_cliente IS NULL OR var_cliente = '' THEN
        SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Errore: Cliente non valido.';
    END IF;

    -- Imposta il livello di isolamento
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

    -- Inizia la transazione
    START TRANSACTION;

    -- Esegue l'update sulla tabella contatti
    UPDATE contatti
    SET valore = var_valore
    WHERE cliente_codicefiscale = var_cliente AND tipo = var_tipo;

    -- Verifica se l'update ha modificato righe
    IF ROW_COUNT() = 0 THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Errore: Nessun contatto trovato per aggiornamento.';
    ELSE
        -- Conferma la transazione
        COMMIT;
    END IF;
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE PROCEDURE updateIndirizzi(
    IN var_via VARCHAR(25),
```

```
    IN var_cap VARCHAR(5),
    IN var_città VARCHAR(25),
    IN var_cliente VARCHAR(25)
)
BEGIN
    -- Gestione degli errori
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Rollback in caso di errore
        ROLLBACK;
        RESIGNAL;
    END;

    -- Validazione dei parametri
    IF var_via IS NULL OR var_via = '' THEN
        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Errore: Via non valida.';
    END IF;

    IF var_cap IS NULL OR var_cap = '' OR CHAR_LENGTH(var_cap) != 5 THEN
        SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Errore: CAP non valido.';
    END IF;

    IF var_città IS NULL OR var_città = '' THEN
        SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Errore: Città non valida.';
    END IF;

    IF var_cliente IS NULL OR var_cliente = '' THEN
        SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Errore: Cliente non valido.';
    END IF;

    -- Imposta il livello di isolamento
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

    -- Inizia la transazione
```

```
START TRANSACTION;

-- Esegue l'update e verifica il risultato
UPDATE indirizzi
SET via = var_via, cap = var_cap, città = var_città
WHERE cliente_codicefiscale = var_cliente;

-- Verifica se l'aggiornamento ha avuto successo
IF ROW_COUNT() = 0 THEN
   ROLLBACK;
   SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cliente non trovato.';
ELSE
   -- Conferma la transazione
   COMMIT;
END IF;
END$$

DELIMITER ;
```

10. Ho introdotto questa procedura, per permettere all'operatore di scrivere la propria nota, scaturita dall'interazione. Come è già stato detto in precedenza i due concetti sono estremamente sovrapponibili. Quindi ho preferito usare nota come atto pratico e interazione come avvenimento. Ho introdotto la possibilità di allegare l'appuntamento.
    Il livello di isolamento scelto è meno restrittivo migliora la concorrenza senza sacrificare la consistenza dei dati.

```
DELIMITER $$

CREATE PROCEDURE writeNote(
   IN var_risultato BOOLEAN,
   IN var_dettagli  VARCHAR(100),
   IN var_cliente VARCHAR(25),
   IN var_ora  TIME,
   IN var_data DATETIME,
   IN var_operatore VARCHAR(45),
```

```
    IN var_dofferta VARCHAR(225),
    IN var_branch_app VARCHAR(45),
    IN var_app_date DATE,
    IN var_app_time TIME
)
BEGIN
    DECLARE var_id_off INT;
    DECLARE var_id_inter INT;

    -- Gestione degli errori
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    -- Validazione dei parametri principali
    IF var_cliente IS NULL OR var_cliente = '' THEN
        SIGNAL SQLSTATE '45002'
        SET MESSAGE_TEXT = 'Errore: Cliente non valido.';
    END IF;

    IF var_dofferta IS NULL OR var_dofferta = '' THEN
        SIGNAL SQLSTATE '45003'
        SET MESSAGE_TEXT = 'Errore: Descrizione offerta non valida.';
    END IF;

    -- Controllo validità data appuntamento
    IF var_app_date IS NOT NULL AND var_app_date < CURRENT_DATE THEN
        SIGNAL SQLSTATE '45004'
        SET MESSAGE_TEXT = 'Errore: Data appuntamento non valida.';
    END IF;

    -- Imposta il livello di isolamento e avvia la transazione
```

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

-- Recupera l'ID dell'offerta corrispondente
SELECT idofferta
INTO var_id_off
FROM offerta
WHERE var_dofferta = offerta.descrizione;

-- Controllo esistenza offerta
IF var_id_off IS NULL THEN
   SIGNAL SQLSTATE '45005'
   SET MESSAGE_TEXT = 'Errore: Offerta non trovata.';
END IF;

-- Inserisce la nuova interazione
INSERT INTO interazione(data, ora, operatore, offerta_idofferta, cliente_codicefiscale, risultato, dettagli)
VALUES (var_data, var_ora, var_operatore, var_id_off, var_cliente, var_risultato, var_dettagli);

-- Recupera l'ID dell'interazione appena creata
SET var_id_inter = LAST_INSERT_ID();

-- Inserisce l'appuntamento solo se i parametri sono validi
IF var_branch_app IS NOT NULL AND var_app_date IS NOT NULL AND var_app_time IS NOT NULL THEN
   INSERT INTO appuntamento (sede, data, ora, cliente_codicefiscale, interazione_idinterazione)
   VALUES (var_branch_app, var_app_date, var_app_time, var_cliente, var_id_inter);
END IF;

-- Conferma la transazione
COMMIT;
```

```
END $$

DELIMITER ;
```

11. La procedura sottostante è utilizzata dagli operatori per richiamare l'elenco delle note di un cliente che stanno contattando. Come risultati avrà tutti i dati del cliente, le interazioni che ha ricevuto con gli eventuali risultati, e le offerte accettate. Ho usato il livello serializable, questo è più restrittivo e garantisce che nessuna transazione concorrente possa leggere o scrivere sui dati coinvolti, garantendo la massima consistenza. Il report riguarda un singolo cliente e non coinvolge transazioni concorrenti su altri clienti. La procedura utilizza tabelle temporanee solo per l'elaborazione di dati specifici al report.

```
DELIMITER $$

CREATE PROCEDURE reportNotes(IN var_cliente VARCHAR(25))
BEGIN
  -- Gestore di errore: rollback e propagazione dell'errore
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    RESIGNAL;
  END;

  -- Verifica se il cliente esiste
  IF NOT EXISTS (
    SELECT 1
    FROM cliente
    WHERE codicefiscale = var_cliente
  ) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Errore: il codice fiscale fornito non corrisponde a un cliente esistente.';
  END IF;

  -- Imposta il livello di isolamento e inizia la transazione
  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
START TRANSACTION;

-- Creazione della tabella temporanea per le interazioni del cliente
CREATE TEMPORARY TABLE temp_interazioni AS
SELECT
   i.idinterazione,
   i.cliente_codicefiscale,
   i.data AS data_interazione,
   i.ora AS ora_interazione,
   i.risultato AS nota_risultato,
   i.dettagli AS nota_dettagli,
   i.operatore AS operatore_interazione,
   i.offerta_idofferta
FROM interazione AS i
WHERE i.cliente_codicefiscale = var_cliente;

-- Creazione della tabella temporanea per le offerte
CREATE TEMPORARY TABLE temp_offerte AS
SELECT
   o.idofferta,
   o.descrizione AS offerta_scelta
FROM offerta AS o;

-- Creazione della tabella temporanea per gli appuntamenti
CREATE TEMPORARY TABLE temp_appuntamenti AS
SELECT
   a.interazione_idinterazione,
   a.sede AS appuntamento_sede,
   a.data AS appuntamento_data,
   a.ora AS appuntamento_ora
FROM appuntamento AS a;

-- Generazione del report combinando le tabelle temporanee
SELECT
```

```
        c.nome AS cliente_nome,
        c.cognome AS cliente_cognome,
        ti.data_interazione,
        ti.ora_interazione,
        ti.nota_risultato,
        ti.nota_dettagli,
        ti.operatore_interazione,
        toff.offerta_scelta,
        tapp.appuntamento_sede,
        tapp.appuntamento_data,
        tapp.appuntamento_ora
    FROM cliente AS c
    INNER JOIN temp_interazioni AS ti
        ON c.codicefiscale = ti.cliente_codicefiscale
    LEFT JOIN temp_offerte AS toff
        ON ti.offerta_idofferta = toff.idofferta
    LEFT JOIN temp_appuntamenti AS tapp
        ON ti.idinterazione = tapp.interazione_idinterazione;
ORDER BY ti.data_interazione DESC, ti.ora_interazione DESC;

    -- Rimozione delle tabelle temporanee
    DROP TEMPORARY TABLE IF EXISTS temp_interazioni;
    DROP TEMPORARY TABLE IF EXISTS temp_offerte;
    DROP TEMPORARY TABLE IF EXISTS temp_appuntamenti;

    COMMIT;
END $$

DELIMITER ;
```

12. L’ultima procedura che espongo, è la più complessa, cioè il report chiamato da parte della segreteria, che mostra in un intervallo temporale specificato, per tutti i clienti, quanti sono contattati e quante volte, così come quante offerte sono state accettate da ciascuno. Ho usato un livello di isolamento repeatable read anche in questo caso, per garantire consistenza senza bloccare tutte le transazioni concorrenti, e altre transazioni stanno inserendo o modificando

dati nelle tabelle di interesse mentre il report è in esecuzione, serializable impedisce che vengano create letture inconsistente, ma il report coinvolge un numero elevato di righe e tabelle con un traffico elevato, potrebbero verificarsi rallentamenti o deadlock.

```
DELIMITER $$

CREATE PROCEDURE reportCustomers(
   IN start_date DATE,
   IN end_date DATE
)
BEGIN
   -- Dichiarazione del gestore di errore
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
      RESIGNAL;
   END;

   -- Verifica che la data di inizio non sia successiva alla data di fine
   IF start_date > end_date THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Start date non può essere maggiore di end date';
   END IF;

   -- Imposta il livello di isolamento
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

   -- Avvia la transazione
   START TRANSACTION;

   -- Creazione di una tabella temporanea consolidata per le informazioni necessarie
   CREATE TEMPORARY TABLE temp_report AS
   SELECT
      c.codicefiscale AS cliente_codicefiscale,
      c.nome AS cliente_nome,
```

```
        c.cognome AS cliente_cognome,
        COUNT(i.cliente_codicefiscale) AS interazioni_cliente,
        COUNT(CASE WHEN i.risultato = 1 THEN 1 END) AS numero_offerte_accettate,
        GROUP_CONCAT(DISTINCT CASE WHEN i.risultato=1 THEN  o.descrizione END
ORDER BY o.descrizione ASC SEPARATOR ', ') AS tipi_offerte_accettate
    FROM cliente AS c
    LEFT JOIN interazione AS i
        ON c.codicefiscale = i.cliente_codicefiscale
        AND i.data BETWEEN start_date AND end_date
    LEFT JOIN offerta AS o
        ON i.offerta_idofferta = o.idofferta
    GROUP BY c.codicefiscale, c.nome, c.cognome;

    -- Selezione finale del report
    SELECT
        cliente_codicefiscale,
        cliente_nome,
        cliente_cognome,
        interazioni_cliente,
        numero_offerte_accettate,
        tipi_offerte_accettate
    FROM temp_report
    ORDER BY cliente_cognome;

    -- Rimozione della tabella temporanea
    DROP TEMPORARY TABLE IF EXISTS temp_report;

    -- Completa la transazione
    COMMIT;
END $$

DELIMITER ;
```