



SEN

Servizio di Emergenza Nazionale

Università degli Studi di Napoli Federico II

Progetto di Basi di Dati

Docente: Chianese Angelo

A.A. 2021-22

Catello Leonardo N46004862

Cipollaro Daiana N46004941

Di Serio Francesco N46004817

Gallucci Ciro N46004791

Indice

1	Introduzione	3
1.1	Specifiche sui dati	3
1.2	Specifiche sulle operazioni	4
1.3	Vincoli tecnologici e politiche di sicurezza	4
2	Progettazione della base di dati	4
2.1	Analisi delle specifiche e ristrutturazione dei requisiti	5
2.1.1	Informazioni generali	5
2.1.2	Informazioni sugli utenti	5
2.1.3	Informazioni sui dipartimenti	5
2.1.4	Informazioni sulle segnalazioni	5
2.1.5	Informazioni sugli operatori	5
2.2	Progettazione concettuale	5
2.2.1	Schema ER portante	5
2.2.2	Raffinamento dello schema: ER semplificato	7
2.2.3	Raffinamento dello schema: ER avanzato	7
2.3	Progettazione logica	8
2.3.1	Fase di ristrutturazione	8
2.3.2	Fase di traduzione	9
2.3.3	Schema logico	11
2.4	Progettazione fisica	11
2.4.1	Dimensionamento fisico della base di dati	11
2.4.2	Creazione della connessione	15
2.4.3	Creazione degli utenti/ruoli e assegnazione dei privilegi	15
2.4.4	Creazione degli oggetti della base di dati e definizione dei vincoli	15
2.4.5	Popolamento della base di dati	15
3	Cenni alla progettazione delle applicazioni	16
3.1	Livello Dati	16
4	Implementazione	16
4.1	Gestione delle eccezioni	16
4.2	<code>system.sql</code>	17
4.3	<code>sen.sql</code>	17
4.4	Package <code>PACK_SEN</code>	24
4.4.1	Specification	25
4.4.2	Body	27
4.5	Trigger	37
4.5.1	<code>_triggerSEN_per_verifica_insert_utente</code>	37
4.5.2	<code>_triggerSEN_per_verifica_insert_titolo</code>	38
4.5.3	<code>_triggerSEN_per_verifica_insert_operatore</code>	38
4.5.4	<code>_triggerSEN_per_verifica_insert_dipartimento</code>	39
4.5.5	<code>_triggerSEN_per_verifica_data</code>	39
4.5.6	<code>_triggerSEN_per_verifica_titolodistudi</code>	40
4.5.7	<code>_triggerSEN_per_insert_operatore</code>	40
4.5.8	<code>_triggerSEN_per_insert_coinvolgimento</code>	41
4.5.9	<code>_triggerSEN_per_update_operatore</code>	41
4.5.10	<code>_triggerSEN_per_delete_operatore</code>	42
4.5.11	<code>_triggerSEN_per_storico_segnalazioni</code>	42

1 Introduzione

Si vuole realizzare una base di dati per gestire un Servizio di Emergenza Nazionale (S.E.N.) costituito da tre principali Dipartimenti: Vigili del Fuoco, Pronto Soccorso e Forze dell'Ordine (queste ultime a loro volta suddivise in Polizia di Stato, Arma dei Carabinieri, Guardia di Finanza ed Esercito). L'obiettivo è consentire la cooperazione tra i Dipartimenti al fine di risolvere una data urgenza in maniera efficiente e rapida.

In seguito al verificarsi di un'emergenza, si stabilisce una relazione tra UTENTE e SEGNALAZIONE sotto forma di RICHIESTA, nella quale sono presenti tutti i dettagli dell'urgenza. A tal punto, tramite un COINVOLGIMENTO, DIPARTIMENTO è informato dell'emergenza e si appresta a risolverla.

1.1 Specifiche sui dati

La base dati da progettare contiene informazioni relative agli UTENTI, alla SEGNALAZIONE e ai DIPARTIMENTI con i relativi OPERATORI.

Degli UTENTI, identificati dal *Numero di Telefono*, devono essere registrate le usuali informazioni anagrafiche e geografiche. La base dei dati deve tener traccia di tutte le comunicazioni tra UTENTI e SEGNALAZIONI, le quali avvengono tramite una RICHIESTA. Ciascun UTENTE può effettuare, in generale, più RICHIESTE diverse.

Della SEGNALAZIONE, identificata dal *Codice* del servizio, deve essere registrata la tipologia dell'emergenza.

Del DIPARTIMENTO, identificato dall'*ID* corrispettivo, devono essere conservate le usuali informazioni geografiche e nominative, nonché del personale e dei mezzi che lo costituiscono. La base dati deve inoltre tener traccia di ciascun COINVOLGIMENTO di uno o più DIPARTIMENTI che concorrono ad una data emergenza.

Degli OPERATORI, ciascuno con proprio *ID*, si tiene traccia delle solite informazioni anagrafiche e del relativo *Titolo di Studi*.

Tale base dati ha l'obiettivo di essere estesa su scala nazionale, che prevede:

- Un numero di UTENTI compreso nell'intervallo [1; 59,55 milioni], ove l'estremo superiore è la popolazione italiana;
- Un numero di SEGNALAZIONI che può essere maggiore o uguale al numero di UTENTI (ciascun UTENTE può effettuare più di una RICHIESTA);
- Circa 34 000 DIPARTIMENTI (circa 28 000 pronto soccorso¹, 447 stazioni dei vigili del fuoco², 580 commissariati di polizia³, circa 4500 comandi stazione dei carabinieri⁴, circa 170 reparti della guardia di finanza⁵ e circa 210 caserme⁶);
- Circa 420 500 OPERATORI (circa 99 000 medici che svolgono attività di pronto soccorso⁷, circa 30 000 pompieri⁸, circa 100 000 poliziotti⁹, circa 118 000 carabinieri¹⁰, circa 63 500 agenti nel settore della guardia di finanza¹¹ e circa 100 000 militari¹²).

Considerando che non si hanno a disposizione né server né supercomputer, tali valori sono ridimensionati per la base dati di implementazione in tale progetto universitario. Si prevedono inizialmente i seguenti valori.

- 10 000 UTENTI;

¹Fonte Ministero della Salute: www.salute.com

²Fonte Pagine Gialle: www.paginegialle.it

³Fonte Pagine Gialle: www.paginegialle.it

⁴Fonte Wikipedia: it.wikipedia.org/wiki/Organizzazione_territoriale_dell%27Arma_dei_Carabinieri

⁵Fonte Wikipedia: it.wikipedia.org/wiki/Guardia_di_Finanza

⁶Fonte ForumFree: militari.forumfree.it/?t=9992290

⁷Fonte ISTAT: dati.istat.it

⁸Fonte Wikipedia: it.wikipedia.org/wiki/Corpo_nazionale_dei_vigili_del_fuoco

⁹Fonte Wikipedia: it.wikipedia.org/wiki/Polizia_di_Stato

¹⁰Fonte Wikipedia: it.wikipedia.org/wiki/Forze_armate_italiane

¹¹Fonte Wikipedia: it.wikipedia.org/wiki/Forze_armate_italiane

¹²Fonte Wikipedia: it.wikipedia.org/wiki/Forze_armate_italiane

- 10 000 SEGNALAZIONI;
- 120 DIPARTIMENTI (una singola tipologia per regione);
- 3000 OPERATORI.

1.2 Specifiche sulle operazioni

Le principali operazioni previste sulla base di dati sono:

- Determinare le richieste inviate dall'utente;
- Determinare luogo, data e ora della richiesta di emergenza;
- Determinare quali operatori appartengono ad ogni dipartimento;
- Determinare il ruolo degli operatori all'interno del proprio dipartimento.
- Inserire e rimuovere gli utenti, nonché stampare la lista degli stessi presenti nella base dati;
- Inserire e rimuovere i dipartimenti, nonché stampare la lista degli stessi presenti nella base dati;
- Inserire e rimuovere gli operatori, nonché stampare la lista degli stessi presenti nella base dati;
- Inserire più titoli di studio per uno stesso operatore;
- Inserire le segnalazioni e ricavare informazioni da esse;
- Inserire un nuovo coinvolgimento.

È inoltre richiesto che (regola di business):

- All'atto della cancellazione di un utente, le informazioni sulle segnalazioni da egli generate devono essere eliminate dalla base di dati e conservate in un'apposita tabella contenente solo dati storici.

1.3 Vincoli tecnologici e politiche di sicurezza

La strumentazione tecnologica adoperata per tale progetto universitario non è in grado di gestire una mole di dati come quella nazionale, per cui si è adoperato il ridimensionamento proposto nella sezione [1.1]. Il DBMS è installato, infatti, su Windows 10 con un laptop *HP Pavilion 15* con 8 GB di RAM e 512 GB di SSD.

Vi sono quattro categorie di utenti che dovranno interagire col sistema di base di dati:

- Gli autori e progettisti della documentazione, che svolgeranno il compito di DBA (*Data Base Administrator*), possedendo, quindi, tutti i privilegi sullo schema;
- Gli utenti, che richiedono il servizio tramite un'applicazione web o cellulare, compilando la richiesta inserendo i propri dati anagrafici, l'indirizzo dell'emergenza e la sua tipologia, scelta tramite un elenco a discesa;
- I dipartimenti, i quali possono inserire, aggiornare e cancellare le informazioni relative ai propri operatori;
- Gli operatori, che tramite un'applicazione su palmare, possono visualizzare le emergenze in corso.

2 Progettazione della base di dati

Terminata la raccolta dei requisiti, è necessario farne un'analisi per avviare la progettazione concettuale, logica e fisica.



Figura 1: Schema a scheletro.

2.1 Analisi delle specifiche e ristrutturazione dei requisiti

2.1.1 Informazioni generali

Si vuole progettare una base dati per creare un collegamento tra le varie entità di soccorso e aumentarne la tempestività di intervento coordinato. Tale base dati dovrà contenere informazioni relative alle emergenze, agli utenti che chiedono assistenza e agli operatori che intervengono.

2.1.2 Informazioni sugli utenti

Degli utenti devono essere conservate le usuali informazioni anagrafiche e geografiche, nonché il recapito telefonico.

2.1.3 Informazioni sui dipartimenti

Per ciascun dipartimento devono essere memorizzati il nome, l'indirizzo della centrale, l'ID corrispondente che lo identifica, nonché il numero degli operatori e dei veicoli.

2.1.4 Informazioni sulle segnalazioni

La base dati deve tener traccia di tutte le richieste fatte dagli utenti, memorizzandone luogo, data, ora e la corrispettiva tipologia.

2.1.5 Informazioni sugli operatori

Per quanto riguarda gli operatori, interessano i dati anagrafici e la specializzazione professionale, nonché un codice identificativo.

Nel **glossario dei termini**, [Tabella 1], si riportano i termini utilizzati, la relativa descrizione, i sinonimi ed i termini collegati.

2.2 Progettazione concettuale

2.2.1 Schema ER portante

Partendo dalle specifiche precedentemente espone, si estrae lo schema ER (*Entity-Relationship*) portante con i concetti fondamentali. Le entità principali in esame sono UTENTE, SEGNALAZIONE, DIPARTIMENTO, rappresentanti rispettivamente l'insieme degli utenti che possono effettuare richieste presso la base dati, l'insieme delle segnalazioni e l'insieme delle parti che compongono la struttura di intervento.

Le entità UTENTE e SEGNALAZIONE sono collegate tramite l'associazione RICHIESTA, la cui cardinalità è $(1, N)$ dal lato UTENTE e $(1, 1)$ dal lato SEGNALAZIONE, in quanto un utente può effettuare una o più richieste, le quali saranno assunte nella SEGNALAZIONE e le stesse sono tutte diverse tra loro, dipendendo da quanto inserito dallo specifico UTENTE.

Le entità SEGNALAZIONE e DIPARTIMENTO sono collegate tramite l'associazione COINVOLGIMENTO, con cardinalità $(1, 6)$ dal lato SEGNALAZIONE, la quale può richiedere l'intervento di uno o più dipartimenti, e $(1, N)$ dal lato DIPARTIMENTO, in quanto è possibile soddisfare almeno un'emergenza.

Alla luce di tali osservazioni, è possibile definire lo schema a scheletro/ER portante del progetto in esame, riportato in [Figura 1].

Termine	Descrizione	Sinonimi	Collegamenti
UTENTE	Individuo che effettua una richiesta di segnalazione di un'emergenza.	Chiamante, Richiedente	RICHIESTA, SEGNALAZIONE
RICHIESTA	Fase di registrazione di un evento che richiede assistenza, corredata di informazioni quali luogo, data ed ora dell'emergenza.	Domanda	UTENTE, SEGNA-LAZIONE
SEGNALAZIONE	Richiesta contenente un codice identificativo e la tipologia dell'emergenza.	Servizio di necessità	UTENTE, RI-CHiesta, COIN-VOLGIMENTO, DIPARTIMENTO
COINVOLGIMENTO	Selezione dei dipartimenti che devono intervenire.	Interessamento	SEGNALAZIONE, DIPARTIMENTO
DIPARTIMENTO	Sezione del servizio di emergenza nazionale adibita al trattamento e alla risoluzione dell'emergenza.	Compartimento, Divisione, Sezione	OPERATORE, VI-GILI DEL FUOCO, PRONTO SOC-CORSO, POLIZIA DI STATO, ARMA DEI CARABINIE-RI, GUARDIA DI FINANZA, ESERCITO.
VIGILI DEL FUOCO	Sezione specifica del Dipartimen-to adibita a emergenze di soccor-so e salvataggio di persone e di contenimento incendi.	Pompieri	DIPARTIMENTO, OPERATORE
PRONTO SOC-CORSO	Sezione specifica del Dipartimen-to adibita alle emergenze di dia-gnosi e alla cura delle persone.	Personale medico, Assistente sanitario	DIPARTIMENTO, OPERATORE
POLIZIA DI STATO	Sezione specifica del Dipartimen-to costituita dal corpo di polizia e volta all'ordinamento civile.	Agente, Pattuglia	DIPARTIMENTO, OPERATORE
ARMA DEI CA-RABINIERI	Sezione specifica del Dipartimen-to con competenza generale e in servizio permanente di pubblica sicurezza.	Corpo armato, Vigilanza	DIPARTIMENTO, OPERATORE
GUARDIA DI FINANZA	Sezione specifica del Dipartimen-to con competenza generale in materia economica e finanziaria.	Finanziere, Guardia	DIPARTIMENTO, OPERATORE
ESERCITO	Sezione specifica del Dipartimen-to che svolge compiti militari, principalmente terrestri.	Soldato, Mi-litare, Mili-zia	DIPARTIMENTO, OPERATORE
OPERATORE	Individuo facente parte dello staff di un Dipartimento con in-carichi dipendenti dalle proprie competenze.	Addetto	DIPARTIMENTO

Tabella 1: Glossario dei termini.

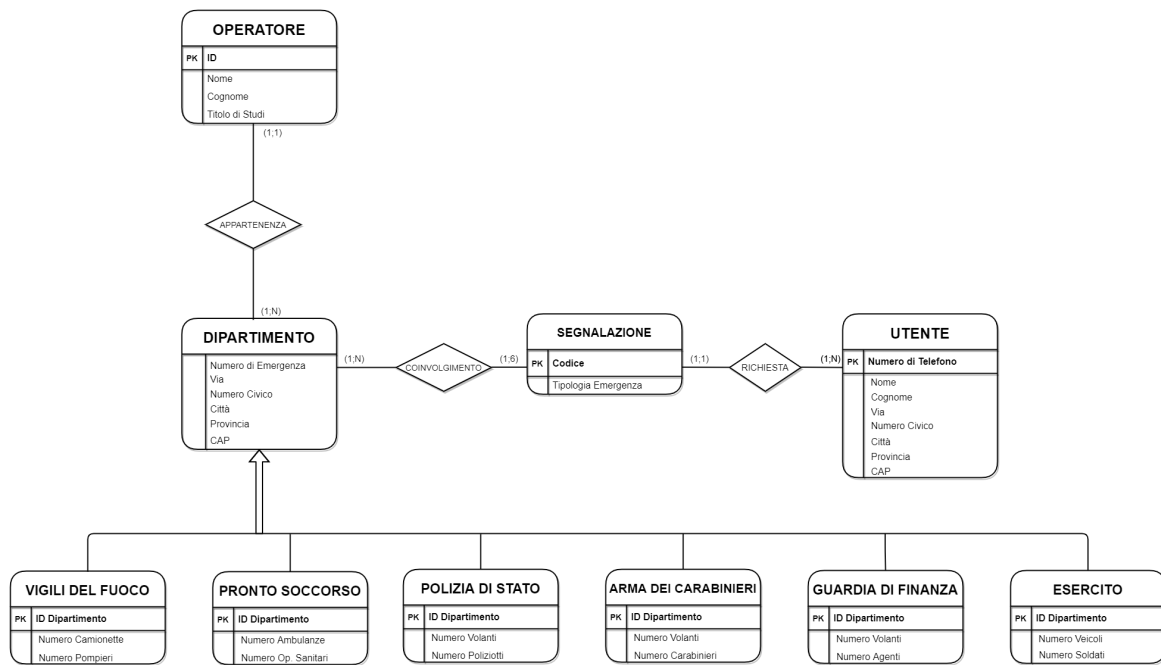


Figura 2: Schema ER semplificato.

2.2.2 Raffinamento dello schema: ER semplificato

Se si esaminano le specifiche, ci si accorge che esistono ulteriori entità ed associazioni da prendere in considerazione, oltre che a un numero considerevole di attributi.

Innanzitutto, vi è una gerarchia ove all'entità padre DIPARTIMENTO sono collegate sei entità figlie VIGILI DEL FUOCO, PRONTO SOCCORSO e le varie forze dell'ordine, ossia POLIZIA DI STATO, ARMA DEI CARABINIERI, GUARDIA DI FINANZA ed ESERCITO. Inoltre, dato che i vari dipartimenti godono di un personale con ruoli specifici, si aggiunge l'entità OPERATORE legata a DIPARTIMENTO al tramite l'associazione APPARTENENZA.

Per ciò che riguarda la cardinalità delle associazioni non vi sono modifiche a quanto analizzato nello schema ER portante. Vale però la seguente considerazione aggiuntiva:

- Le entità OPERATORE e DIPARTIMENTO sono collegate tramite l'associazione APPARTENENZA, la cui cardinalità è $(1, N)$ dal lato DIPARTIMENTO e $(1, 1)$ dal lato OPERATORE, in quanto un dipartimento può avere molteplici operatori, mentre un operatore può appartenere ad uno ed un solo dipartimento; dunque, il rapporto di cardinalità è uno a molti.

Si ottiene dunque quanto in [Figura 2], definito *schema ER semplificato*.

2.2.3 Raffinamento dello schema: ER avanzato

Lo schema ER di [Figura 2] consente di avere una chiara visione della base dati, eppure non completa la descrizione informativa di tutte le entità in esame. In un raffinamento successivo è importante far notare, oltre alla cospicua mole di attributi già presente, l'aggiunta di:

- Attributi composti quali *Indirizzo di Residenza* per l'entità UTENTE, *Luogo Emergenza* per la relazione RICHIESTA, *Indirizzo della Centrale* per l'entità DIPARTIMENTO;
- Attributo *Ruolo* per l'associazione APPARTENENZA, che permette di specificare la mansione svolta dall'OPERATORE nel DIPARTIMENTO, nonché l'attributo di tipo **timestamp**, *Data_Ora*, per l'associazione RICHIESTA;
- Attributo multivalore *Titolo di Studi* per l'entità OPERATORE.

Tali aggiunte permettono di ottenere uno schema finale ricco di informazioni per il cliente, riportato in [Figura 3].

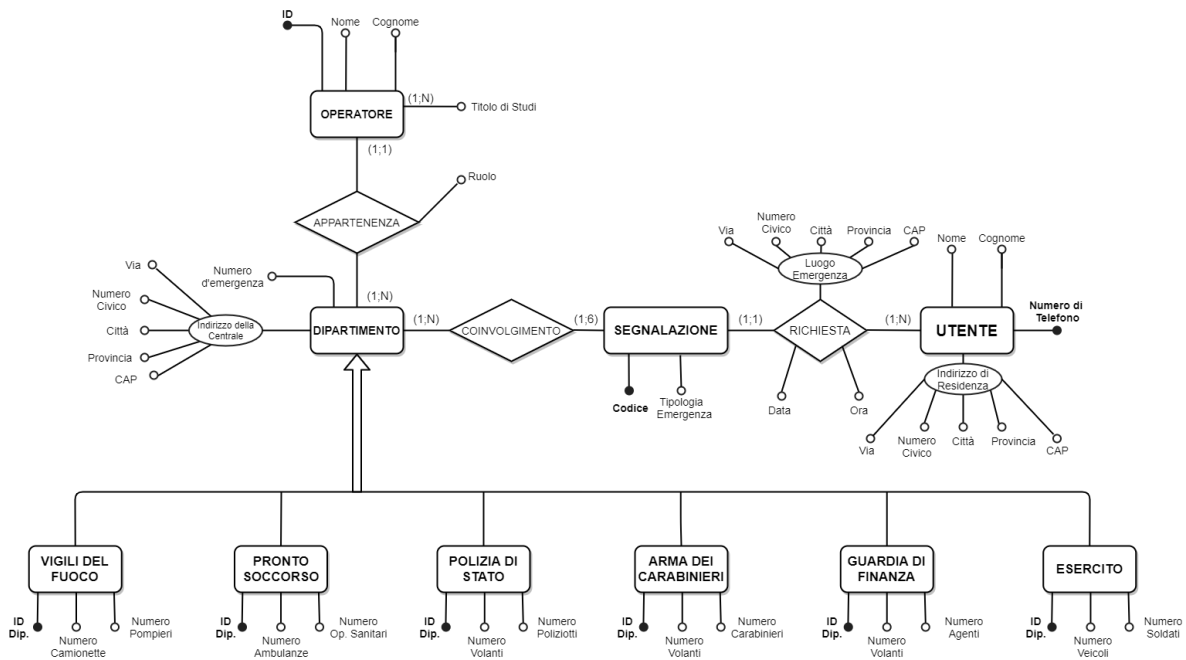


Figura 3: Schema ER avanzato.

2.3 Progettazione logica

La fase successiva è la progettazione logica, la quale prevede la definizione dello schema relazionale, in termini di insieme di relazioni e vincoli, della base di dati.

Essa si divide in due fasi:

- **Ristrutturazione:** trasformazione dello schema concettuale al fine di poter essere sottoposto alla fase di traduzione e alla relativa semplificazione;
- **Traduzione** dello schema semplificato nello schema logico.

2.3.1 Fase di ristrutturazione

Prendendo in considerazione lo schema ER avanzato di [Figura 3], si attui la ristrutturazione. Quest'ultima si rende necessaria, in quanto non tutti gli schemi ER possono essere tradotti nel modello logico. Nella fattispecie, è impossibile tradurre attributi composti e multivalore, così come le gerarchie.

Tale fase, che produce lo schema ER ristrutturato, consta delle seguenti fasi:

- Eliminazione attributi composti e multivalore;
- Scelta degli identificatori principali;
- Eliminazione delle gerarchie;
- Partizione/Accorpamento delle entità;
- Analisi delle ridondanze.

Per quanto riguarda il primo punto, onde violare la I forma normale, ciascun attributo multivalore (*Titolo di Studi* di OPERATORE) è sostituito da una nuova entità (TITOLO DI STUDI per l'appunto) ed una nuova relazione (COMPETENZA), mentre ciascun attributo composto (*Indirizzo di Residenza* dell'UTENTE, *Indirizzo della Centrale* di DIPARTIMENTO e *Luogo Emergenza* di RICHIESTA) è scisso in tanti attributi quanti ne costituiscono il composto.

Successivamente, per quanto concerne secondo e quarto punto, non è necessario intervento alcuno: infatti, ciascuna entità presenta una sola chiave primaria e non risulta necessario operare partizioni od accorpamenti di entità.

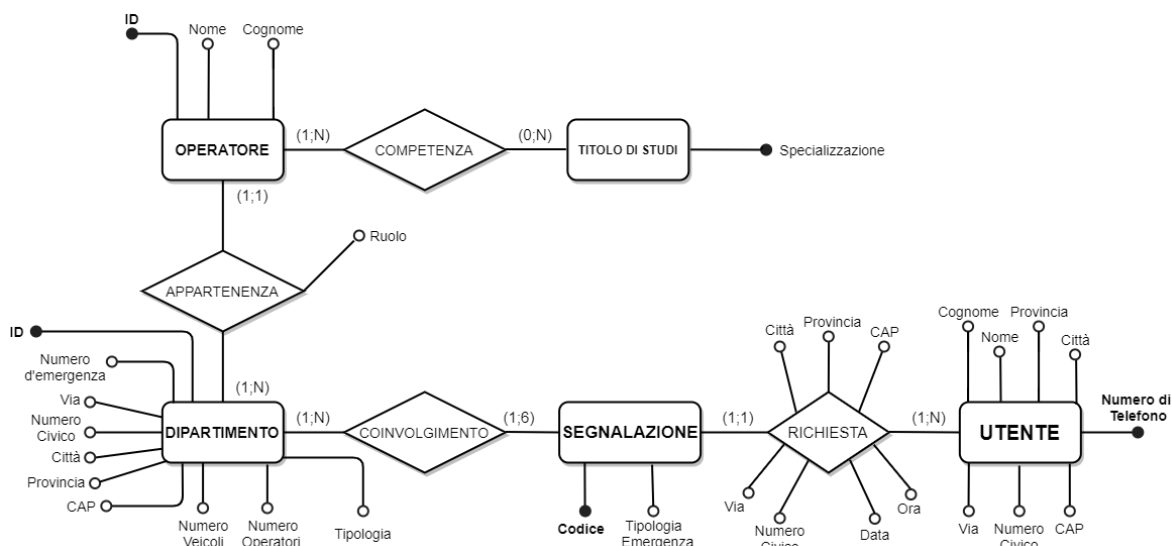


Figura 4: Schema ER equivalente/ristrutturato.

Segue l'eliminazione delle gerarchie, le quali, sebbene siano costrutti potenti usati nella progettazione concettuale per evidenziare le dipendenze della realtà, non trovano, nei modelli logici delle basi dati, un immediato riscontro. Difatti, tali gerarchie impongono una trasformazione del costrutto adoperato per rappresentarle in combinazioni di costrutti base quali entità e relazioni. La generalizzazione nel progetto in esame di tipo **Totale/Disgiunta** (*totale* poiché ogni occorrenza della classe padre DIPARTIMENTO, denominata *superclasse*, è occorrenza di almeno un'entità figlia, detta *sottoclasse*; *disgiunta* – od *esclusiva* – in quanto ogni occorrenza di DIPARTIMENTO è al più di una entità figlia). Onde evitare la presenza di ridondanze tra i vari dipartimenti, si è scelto di accoppiare le sottoclassi nella superclasse e non viceversa. A quest'ultima è inoltre aggiunto un attributo per distinguere il tipo di dipartimento di cui è richiesto l'intervento, denominato *Tipologia*.

L'unica ridondanza presente nel progetto, come sarà indicato dalla [Figura 6], è *Numero operatori* di DIPARTIMENTO (osservabile già in [Figura 4]).

2.3.2 Fase di traduzione

Al termine della ristrutturazione, si è ottenuto lo schema ER equivalente, [Figura 4], ristrutturato sulla base delle esigenze del modello logico. Si può quindi procedere alla traduzione nel modello relazionale e generare lo schema della base dati. Tale fase avrà come risultato le tabelle, che saranno indicate secondo il loro schema. Sia le entità sia le relazioni saranno trasformate in tabelle.

La traduzione di un'entità produce una tabella che ha per nome quello dell'entità, come attributi – ossia gli identificativi delle colonne – quelli presenti in quest'ultima, nonché per chiave l'identificatore principale dell'entità stessa.

La traduzione delle relazioni presenta approcci differenti in base alla cardinalità di relazione. In particolare, si tratteranno solo le *MaM* (molti a molti) e le *UaM* (uno a molti); le *UaU* (uno ad uno) non sono presenti nel progetto soggetto della trattazione.

- La traduzione di una relazione **MaM** produce una tabella che ha per nome quello della relazione, come attributi quelli della relazione e gli identificatori primari di tutte le entità coinvolte; questi ultimi formano la chiave della relazione.

È il caso di COINVOLGIMENTO (che lega DIPARTIMENTO e SEGNALAZIONE) e di COMPETENZA (che lega OPERATORE e TITOLODISTUDI). In particolare, si è scelto di rinominare gli attributi esterni al fine di aumentare la leggibilità degli schemi stessi.

- La traduzione di una relazione **UaM** consente la fusione della relazione nell'entità che fornisce la **U** (e quindi l'*uno*) della cardinalità. Si ottiene che la tabella dell'entità contiene, in aggiunta a tutte le informazioni su di sé, anche gli eventuali attributi della relazione e la chiave primaria

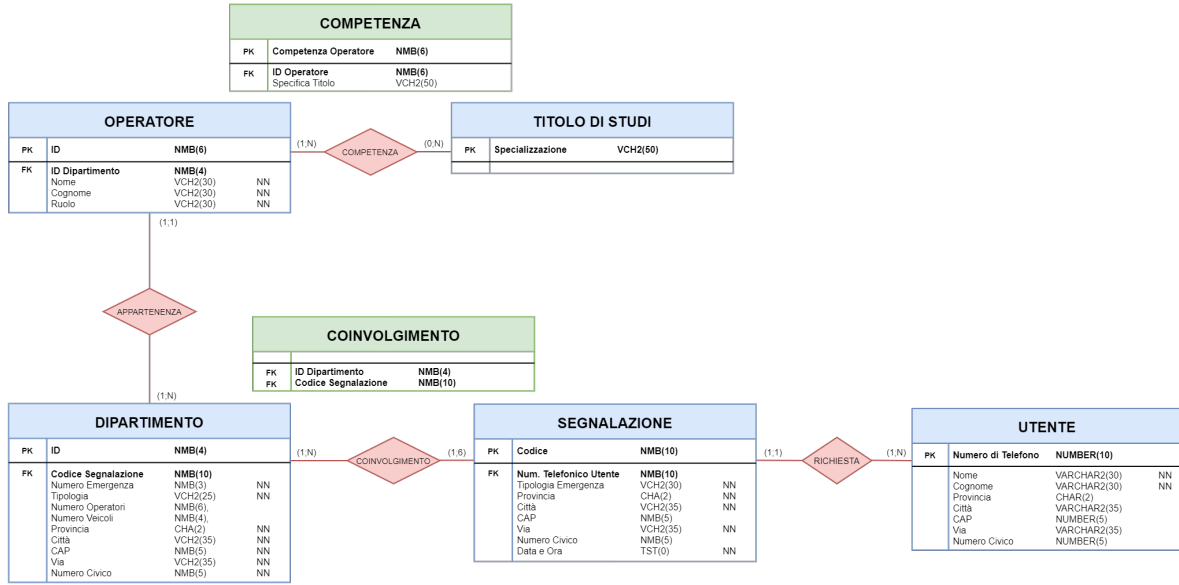


Figura 5: Schema ER semplificato finale.

dell'entità che fornisce la **M** (e quindi il *molti*) della cardinalità. Per semplicità, si rinomina quest'ultimo attributo nel nome dell'entità da cui proviene.

È il caso di **APPARTENENZA**, che confluisce in **OPERATORE**, e di **RICHIESTA**, che confluisce in **SEGNALAZIONE**.

Si riporta dunque lo schema ER semplificato (adesso *finale*, [Figura 5]) sotto una nuova veste, che riporta, rispetto alla precedente [Figura 2], tutte le modifiche apportate durante la ristrutturazione e la traduzione.

Inoltre, si riporta lo schema relazionale completo per la base dati in oggetto. In aggiunta a quanto finora definito, si inserisce **STORICOSEGNALAZIONI** al fine di soddisfare la regola di business definita nella sezione [1.2].

UTENTI(NumeroDiTelefono, Nome, Cognome, Via, NumeroCivico, Città, Provincia, CAP)
SEGNALAZIONI(Codice, TipologiaEmergenza, NumTelefonicoUtente:UTENTE, Via, NumeroCivico, Città, Provincia, CAP, Data_Ora)
DIPARTIMENTI(ID, NumeroEmergenza, Tipologia, Via, NumeroCivico, Città, Provincia, CAP, NumeroVeicoli, NumeroOperatori)
OPERATORI(ID, Nome, Cognome, IDDip:DIPARTIMENTO, Ruolo)
COMPETENZE(IDOpe:OPERATORI, SpecTitolo:TITOLODI STUDI)
TITOLODI STUDI(Specializzazione)
COINVOLGIMENTI(IDDip:DIPARTIMENTO, CodSegnalazione:SEGNALAZIONE)

STORICOSEGNALAZIONI(Codice, NumeroDiTelefono, Nome, Cognome, TipologiaEmergenza, Via, NumeroCivico, Città, Provincia, CAP, Data_Ora)

Si può facilmente verificare che tutte le relazioni dello schema ricavato sono in **BCNF** (Forma Normale di Boyce e Codd).

Si ricordi infatti che uno schema di relazione $R(X)$ è in BCNF se è in 1NF e per ogni dipendenza funzionale non banale $Y \rightarrow Z$ – con Y e Z sottoinsiemi non vuoti di X , insieme degli attributi –, Y è superchiave di $R(X)$.

Che tutti gli schemi di relazioni presenti nello schema relazionale siano in 1NF è presto detto: tutti i domini sono atomici. In seguito, si osserva come gli attributi non primi dipendano tutti da quelli primi e non viceversa.



Figura 6: Schema logico.

2.3.3 Schema logico

Si conclude la progettazione logica mostrando lo schema omonimo. Relativamente a quest'ultimo, [Figura 6], sono indicate in giallo le chiavi primarie, in viola quelle esterne, in rosso le ridondanze.

2.4 Progettazione fisica

La progettazione fisica di una base di dati è un processo complesso, il quale può essere suddiviso nelle fasi seguenti:

1. Dimensionamento fisico della base dati, sia globale sia a livello dei singoli oggetti, in termini di calcolo dello *storage* – ossia lo spazio di memorizzazione – richiesto su disco;
2. Creazione della connessione;
3. Creazione degli utenti/ruoli e assegnazione dei privilegi;
4. Creazione degli oggetti della base di dati e definizione dei vincoli;
5. Creazione di un'istanza (popolamento della base dati).

La progettazione fisica del database *SEN - Servizio di Emergenza Nazionale* si basa sulle seguenti condizioni operative:

- Installazione della licenza di *Oracle 11g*;
- Server con quattro CPU ed otto processori logici, 8 GB RAM e 512 GB SSD;
- Sistema operativo Windows 10.

2.4.1 Dimensionamento fisico della base di dati

A partire dalle informazioni sul volume dei dati ed una volta scelti i tipi degli attributi da utilizzare nell'implementazione di ogni tabella, si effettua una stima dei costi in termini di occupazione di memoria della base dati per la successiva fase di implementazione.

Seguono le occupazioni – in termini di byte – per il DBMS Oracle dei tipi di dati adoperati.

- **NUMBER(x)**: $(\lceil x/2 \rceil + 2)$ byte;

Attributo	Tipo	Byte	Initial 10 000 occ.	Next 5 000 occ.
NumeroDiTelefono	NUMBER(10)	7	70.0 kB	35.0 kB
Nome	VARCHAR2(30)	30	300 kB	150 kB
Cognome	VARCHAR2(30)	30	300 kB	150 kB
Provincia	CHAR(2)	2	20.0 kB	10.0 kB
Città	VARCHAR2(35)	35	350 kB	175 kB
CAP	NUMBER(5)	5	50.0 kB	25.0 kB
Via	VARCHAR2(35)	35	350 kB	175 kB
NumeroCivico	NUMBER(5)	5	50.0 kB	25.0 kB
Totale (dati)			1.49 MB	745 kB

Tabella 2: Dimensionamento tabella UTENTI.

Attributo	Tipo	Byte	Initial 10 000 occ.	Next 5 000 occ.
Codice	NUMBER(10)	7	70.0 kB	35.0 kB
TipologiaEmergenza	VARCHAR2(30)	30	300 kB	150 kB
NumTelefonicoUtente	NUMBER(10)	7	70.0 kB	35.0 kB
Provincia	CHAR(2)	2	20.0 kB	10.0 kB
Città	VARCHAR2(35)	35	350 kB	175 kB
CAP	NUMBER(5)	5	50.0 kB	25.0 kB
Via	VARCHAR2(35)	35	350 kB	175 kB
NumeroCivico	NUMBER(5)	5	50.0 kB	25.0 kB
Data_Ora	TIMESTAMP(0)	7	70.0 kB	35.0 kB
Totale (dati)			1.33 MB	665 kB

Tabella 3: Dimensionamento tabella SEGNALAZIONI.

- CHAR(x): x byte;
- VARCHAR2(x): $(0 \div x)$ byte;
- TIMESTAMP(x): $(7 \div 13)$ byte, in particolare TIMESTAMP(0) occupa 7 byte.

Si riporta nelle [Tabelle 2 a 9] una stima dell'occupazione di memoria a regime per ciascuna tabella considerata. Tale stima è effettuata su scala del progetto corrente (campo *initial*) e su una previsione dopo sei mesi (campo *next*).

Per ogni tabella viene calcolata l'occupazione di memoria nel *worst case scenario*, andando a considerare il numero di byte richiesti da ogni singolo campo.

A scopo riepilogativo, si riporta la [Tabella 10], riportante l'occupazione di memoria dei singoli oggetti adoperati.

Per tutte le tabelle appena citate, è stata effettuata la semplificazione $1 \text{ MB} = 1000 \text{ kB}$, anziché 1024 kB , e $1 \text{ kB} = 1000 \text{ B}$, anziché 1024 B .

Attributo	Tipo	Byte	Initial 120 occ.	Next 30 occ.
ID	NUMBER(4)	4	0.48 kB	0.12 kB
NumeroEmergenza	NUMBER(3)	4	0.48 kB	0.12 kB
Tipologia	VARCHAR2(25)	25	3.00 kB	0.75 kB
NumeroOperatori	NUMBER(6)	5	0.60 kB	0.15 kB
NumeroVeicoli	NUMBER(4)	4	0.48 kB	0.12 kB
Provincia	CHAR(2)	2	0.24 kB	0.06 kB
Città	VARCHAR2(35)	35	4.20 kB	1.05 kB
CAP	NUMBER(5)	5	0.60 kB	0.15 kB
Via	VARCHAR2(35)	35	4.20 kB	1.05 kB
NumeroCivico	NUMBER(5)	5	0.60 kB	0.15 kB
Totale (dati)			14.9 kB	3.72 kB

Tabella 4: Dimensionamento tabella DIPARTIMENTI.

Attributo	Tipo	Byte	Initial 3 000 occ.	Next 600 occ.
ID	NUMBER(6)	5	15.0 kB	3.00 kB
Nome	VARCHAR2(30)	30	90.0 kB	18.0 kB
Cognome	VARCHAR2(30)	30	90.0 kB	18.0 kB
SpecTitolo	VARCHAR2(50)	50	150 kB	30.0 kB
Provincia	NUMBER(4)	4	12.0 kB	2.40 kB
IDDip	NUMBER(4)	4	12.0 kB	2.40 kB
Ruolo	VARCHAR2(30)	30	90.0 kB	18.0 kB
Totale (dati)			459 kB	91.8 kB

Tabella 5: Dimensionamento tabella OPERATORI.

Attributo	Tipo	Byte	Initial 1 000 occ.	Next -
Specializzazione	VARCHAR2(50)	50	50.0 kB	
Totale (dati)			50.0 kB	-

Tabella 6: Dimensionamento tabella TITOLIDISTUDI.

Attributo	Tipo	Byte	Initial 10 000 occ.	Next 5 000 occ.
IDDip	NUMBER(4)	4	40.0 kB	20.0 kB
CodSegnalazione	NUMBER(10)	7	70.0 kB	35.0 kB
Totale (dati)			110 kB	55.0 kB

Tabella 7: Dimensionamento tabella COINVOLGIMENTI.

Attributo	Tipo	Byte	Initial 3750 occ.	Next 900 occ.
IDOpe	NUMBER(6)	5	18.8 kB	4.50 kB
SpecTitolo	VARCHAR2(50)	50	188 kB	45.0 kB
Totale (dati)			207 kB	49.5 kB

Tabella 8: Dimensionamento tabella COMPETENZE.

Attributo	Tipo	Byte	Initial 10 000 occ.	Next 20 000 occ.
NumeroDiTelefono	NUMBER(10)	7	70.0 kB	140 kB
Nome	VARCHAR2(30)	30	300 kB	600 kB
Cognome	VARCHAR2(30)	30	300 kB	600 kB
Provincia	CHAR(2)	2	20.0 kB	40.0 kB
Città	VARCHAR2(35)	35	350 kB	700 kB
CAP	NUMBER(5)	5	50.0 kB	100 kB
Via	VARCHAR2(35)	35	350 kB	700 kB
NumeroCivico	NUMBER(5)	5	50.0 kB	100 kB
Data_Ora	TIMESTAMP(0)	7	70.0 kB	140 kB
Totale (dati)			1.56 MB	3.12 MB

Tabella 9: Dimensionamento tabella STORICOSEGNALAZIONI.

Oggetto	Nome	Initial	Next
Tabella	UTENTI	1.49 MB	745 kB
Tabella	SEGNALAZIONI	1.33 MB	665 kB
Tabella	DIPARTIMENTI	14.9 kB	3.72 kB
Tabella	OPERATORI	459 kB	91.8 kB
Tabella	TITOLI Di STUDI	50.0 kB	-
Tabella	COINVOLGIMENTI	110 kB	55.0 kB
Tabella	COMPETENZE	207 kB	49.5 kB
Tabella	STORICOSEGNALAZIONI	1.56 MB	3.12 MB
Sequenza	codSeg_seq	16.0 kB	-
Sequenza	idDip_seq	16.0 kB	-
Sequenza	idOpe_seq	16.0 kB	-

Tabella 10: Dimensionamento finale.

2.4.2 Creazione della connessione

Inerentemente ad Oracle SQL Developer, inizialmente non vi sono connessioni preesistenti al di fuori degli utenti permanenti del DBMS, **sys** e **system**.

Dunque, bisogna adoperare **system** affinché sia possibile effettuare la connessione di cui si necessita: infatti, dopo aver fatto click su *Crea una connessione manualmente* – o eventualmente sul *+* verde nella finestra *Connessioni* – si inserisce **system** presso le voci *Nome* e *Nome utente*, nonché la password, definita in fase di installazione di Oracle, nel campo *Password*.

In seguito, in *Dettagli*, si popola il *Nome host* con *local host*, la *Porta* con *1521* ed il *S_ID* si modifica da **xe** (*Oracle Expression*) a **orcl**.

Se nel *Test* non vengono riscontrati problemi, si può effettuare la connessione con un click su *Connetti*.

2.4.3 Creazione degli utenti/ruoli e assegnazione dei privilegi

Si è deciso di creare un utente DBA – *DataBase Administrator* –, **sen**, proprietario della base di dati, a cui sono stati assegnati tutti i privilegi. Inoltre, la password di accesso a quest'ultimo risulta **progetto_2021**.

Lo script per la creazione del DBA del progetto in esame è il seguente.

```
CREATE USER sen IDENTIFIED BY progetto_2021;  
GRANT ALL PRIVILEGES TO sen;
```

Di seguito si riportano gli script relativi alla generazione dei ruoli con la definizione dei relativi privilegi.

```
CREATE ROLE utente;  
GRANT CONNECT TO utente;  
GRANT INSERT, UPDATE, DELETE ON sen.UTENTI TO utente;
```

```
CREATE ROLE amm_dipartimento;  
GRANT CONNECT TO amm_dipartimento;  
GRANT INSERT, UPDATE, DELETE ON sen.DIPARTIMENTI TO amm_dipartimento;  
GRANT INSERT, UPDATE, DELETE ON sen.OPERATORI TO amm_dipartimento;  
GRANT INSERT, UPDATE, DELETE ON sen.TITOLIDISTUDI TO amm_dipartimento;  
GRANT INSERT, UPDATE, DELETE ON sen.COMPETENZE TO amm_dipartimento;
```

```
CREATE ROLE centralinista;  
GRANT CONNECT TO centralinista;  
GRANT SELECT ON sen.SEGNALAZIONI TO centralinista;  
GRANT INSERT, UPDATE, DELETE ON sen.COINVOLGIMENTI TO centralinista;
```

2.4.4 Creazione degli oggetti della base di dati e definizione dei vincoli

In tale fase, risulta fondamentale adoperare due comandi del DDL (*Data Definition Language*).

- **CREATE**, al fine di creare gli oggetti della base dati, nello specifico tabelle e sequenze;
- **ALTER**, al fine di alterare il catalogo della base dati, introducendo i vincoli di integrità referenziale.

Per il codice di riferimento, si rimanda alla sezione [4.3]

2.4.5 Popolamento della base di dati

Il popolamento della base di dati avviene attraverso gli statement SQL di **insert** oppure attraverso opportune procedure di *import* che automatizzano la creazione dell'istanza.

Per il codice di riferimento, si rimanda alla sezione [4.3].

3 Cenni alla progettazione delle applicazioni

L'architettura software dell'applicazione deve svilupparsi su tre livelli logico-funzionali:

1. Il **livello di presentazione**, anche noto come *presentation layer* o *front-end*;
2. Il **livello applicativo**, anche noto come *application layer* o *business layer*;
3. Il **livello dati**, anche noto come *access data layer* o *back-end*.

La definizione dei primi due livelli esula dall'obiettivo del progetto in esame.

Quest'ultimo potrà, in tempi futuri, essere un punto di partenza per eventuali applicazioni o interfacce web, come suggerito già nella sezione [1.3].

Di conseguenza, si focalizzerà l'attenzione sul solo livello dati.

3.1 Livello Dati

Tale livello ha il compito di fornire i servizi per la gestione efficiente dei dati.

È interamente costituito dal DBMS Oracle 11g, dalla base di dati finora progettata e da applicazioni memorizzate nel DBS (*DataBase System*), ossia *stored procedure* e *trigger*.

L'obiettivo è rendere l'informazione quanto più possibile indipendente dalle applicazioni di livello superiore, non proposte in tale sede.

Dunque, al fine di visionare il codice PL/SQL relativo alle procedure e ai trigger, si rimanda alle sezioni [4.4] e [4.5] rispettivamente.

4 Implementazione

In tale sezione, viene mostrato il codice PL/SQL del progetto *SEN - Servizio di Emergenza Nazionale*, adoperando una suddivisione in sottoparagrafi concorde alla suddivisione in file del codice.

Si comincia con una panoramica sulle eccezioni adoperate in *stored procedure* e *trigger*.

4.1 Gestione delle eccezioni

In virtù di tutte le accortezze avute nella fase di progettazione della base dati, si è in grado di fornire una gestione delle eccezioni, tramite l'ausilio di **RAISE** in package e trigger.

- `PROCEDURE delete_utente;`
- `PROCEDURE delete_dipartimento;`
- `PROCEDURE delete_operatore;`
- `PROCEDURE delete_titolo;`
- `TRIGGER VINCOLI_COINVOLGIMENTI_INSERT;`
- `TRIGGER VERIFICA_DATA;`
- `TRIGGER VERIFICA_INSERT_DIPARTIMENTI;`
- `TRIGGER VERIFICA_INSERT_OPERATORI;`
- `TRIGGER VERIFICA_INSERT_TITOLI;`
- `TRIGGER VERIFICA_INSERT_UTENTI;`

Si proceda nell'esaminare ciascuna eccezione presente nel package:

- `delete_utente`: dà in output un messaggio di errore qualora l'utente da rimuovere non sia presente nella base dati (codice: **-20600**);
- `delete_dipartimento`: dà in output un messaggio di errore qualora il dipartimento da rimuovere non sia presente nella base dati (codice: **-20601**);

- `delete_operatore`: dà in output un messaggio di errore qualora l'operatore da rimuovere non sia presente nella base dati (codice: **-20602**);
- `delete_titolo`: dà in output un messaggio di errore qualora il titolo da rimuovere non sia presente nella base dati (codice: **-20603**);

Per quanto concerne i trigger, invece:

- `VINCOLI_COINVOLGIMENTI_INSERT`: l'eccezione si manifesta all'inserimento di un numero di dipartimenti superiore a quello previsto dalla base dati (codice: **-20000**);
- `VERIFICA_DATA`: l'eccezione si manifesta all'inserimento di data e ora non valide (codice: **-20900**);
- `VERIFICA_INSERT_DIPARTIMENTI`: l'eccezione si manifesta all'inserimento di un dipartimento già esistente nella base dati (codice: **-20501**);
- `VERIFICA_INSERT_OPERATORI`: l'eccezione si manifesta all'inserimento di un operatore già esistente nella base dati (codice: **-20502**);
- `VERIFICA_INSERT_TITOLI`: l'eccezione si manifesta all'inserimento di un titolo già esistente nella base dati (codice: **-20503**);
- `VERIFICA_INSERT_UTENTI`: l'eccezione si manifesta all'inserimento di un utente già esistente nella base dati (codice: **-20500**);

4.2 system.sql

Come descritto nella sezione [2.4.2], in Oracle SQL Developer non vi sono connessioni preesistenti al di fuori degli utenti permanenti del DBMS.

Per tale motivazione, una volta connessi come `system`, il codice da apporre nel file `system.sql` è il seguente.

```
-- Creazione utente "sen" con password "progetto_2021"
CREATE USER sen IDENTIFIED BY progetto_2021;

-- Assegnazione dei privilegi
GRANT ALL PRIVILEGES TO sen;
```

4.3 sen.sql

Il file `sen.sql` riporta l'essenza del progetto, tra cui la creazione degli oggetti e la loro alterazione e modifica. Inoltre, contiene anche la sequenza di comandi adoperata per testare la base dati, in particolare le procedure ed i trigger.

```
-- 1) Tabella utenti
CREATE TABLE UTENTI(
    NumeroDiTelefono NUMBER(10),
    Nome VARCHAR2(30) NOT NULL,
    Cognome VARCHAR2(30) NOT NULL,
    Provincia CHAR(2),
    Città VARCHAR2(35), -- La città con nome più lungo in Italia presenta ben 34 lettere
    CAP NUMBER(5),
    Via VARCHAR2(35),
    NumeroCivico NUMBER(5), -- Il numero civico più alto in Italia è circa 14500

    CONSTRAINT PK_UTENTI PRIMARY KEY(NumeroDiTelefono)
)
STORAGE(INITIAL 1490k NEXT 745k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);
-- Eseguendo ciascuna CREATE TABLE, nell'output screen appare "Creato table <nome_table>."
```

```
-- 2) Tabella segnalazioni
/* LISTA DELLE SEGNALAZIONI ACCETTATE DAL DATABASE
*
* ----- INTERVENTO SINGOLO -----
*
* ***** VIGILI DEL FUOCO *****
* INCENDI, FUGHE DI GAS, CROLLI, CALAMITÀ, ATTACCHI TERRORISTICI, SOCCORSO IN RICERCA,
* ALLUVIONI, INCIDENTI
*
* ***** PRONTO SOCCORSO *****
* INCIDENTI, CALAMITÀ, ATTACCHI TERRORISTICI, INCENDI, CROLLI, VIOLENZE
*
* ***** POLIZIA DI STATO *****
* FURTI, SMARRIMENTI, VIOLENZE, INFLAZIONI, TRAFFICO URBANO, MANIFESTAZIONI, INCIDENTI,
* ASSASINI, VIOLAZIONI VARIE, TERRORISMO
*
* ***** ARMA DEI CARABINIERI *****
* FURTI, SMARRIMENTI, ATTACCHI TERRORISTICI, VIOLENZE, ESPLOSIONI, MANIFESTAZIONI,
* INCIDENTI, ASSASINI, VIOLAZIONI VARIE
*
* ***** GUARDIA DI FINANZA *****
* TRUFFA, EVASIONE FISCALE, CONTRABBANDO, RICICLAGGIO, CONTRAFFAZIONE
*
* ***** ESERCITO *****
* ATTACCHI TERRORISTICI, VIOLENZE, POSSESSO DI DROGA, INCIDENTI, MANIFESTAZIONI
*
* ----- INTERVENTO MULTIPOLO -----
*
* INCENDI (VIGILI DEL FUOCO, PRONTO SOCCORSO, ARMA DEI CARABINIERI)
*
* ATTACCHI TERRORISTICI (VIGILI DEL FUOCO, PRONTO SOCCORSO, ARMA DEI CARABINIERI)
*
* CALAMITÀ (VIGILI DEL FUOCO, PRONTO SOCCORSO, ARMA DEI CARABINIERI, ESERCITO)
*
* VIOLENZE (PRONTO SOCCORSO, POLIZIA DI STATO, ARMA DEI CARABINIERI, ESERCITO)
*
* INCIDENTI (VIGILI DEL FUOCO, PRONTO SOCCORSO, POLIZIA DI STATO, ARMA DEI CARABINIERI, ESERCITO)
*
* CROLLI (VIGILI DEL FUOCO, PRONTO SOCCORSO, POLIZIA DI STATO, ARMA DEI CARABINIERI, ESERCITO)
*
* MANIFESTAZIONI (POLIZIA DI STATO, ARMA DEI CARABINIERI, ESERCITO)
*
* ASSASSINI (POLIZIA DI STATO, ARMA DEI CARABINIERI, ESERCITO)
*
*/
CREATE TABLE SEGNALAZIONI(
    Codice NUMBER(10),
    TipologiaEmergenza VARCHAR2(30) NOT NULL,
    NumTelefonicoUtente NUMBER(10),
    Provincia CHAR(2) NOT NULL,
    Città VARCHAR2(35) NOT NULL,
    CAP NUMBER(5),
    Via VARCHAR2(35) NOT NULL,
    NumeroCivico NUMBER(5),
    Data_Ora TIMESTAMP(0) NOT NULL,
```

```

        CONSTRAINT PK_SEGNALAZIONI PRIMARY KEY(Codice)
    )
    STORAGE(INITIAL 1330k NEXT 665k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- 3) Tabella dipartimenti
CREATE TABLE DIPARTIMENTI(
    ID NUMBER(4),
    NumeroEmergenza NUMBER(3) NOT NULL,
    Tipologia VARCHAR2(25) NOT NULL,
    NumeroOperatori NUMBER(6),
    NumeroVeicoli NUMBER(4),
    Provincia CHAR(2) NOT NULL,
    Città VARCHAR2(35) NOT NULL,
    CAP NUMBER(5) NOT NULL,
    Via VARCHAR2(35) NOT NULL,
    NumeroCivico NUMBER(5) NOT NULL,

    CONSTRAINT PK_DIPARTIMENTI PRIMARY KEY(ID),
    CONSTRAINT CK_DIPARTIMENTI CHECK(Tipologia IN
                                     ('Vigili del Fuoco', 'Pronto Soccorso',
                                      'Polizia di Stato', 'Arma dei Carabinieri',
                                      'Guardia di Finanza', 'Esercito'))
)
    STORAGE(INITIAL 15k NEXT 4k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- 4) Tabella operatori
CREATE TABLE OPERATORI(
    ID NUMBER(6),
    Nome VARCHAR2(30) NOT NULL,
    Cognome VARCHAR2(30) NOT NULL,
    IDDip NUMBER(4) NOT NULL,
    Ruolo VARCHAR2(30) NOT NULL,

    CONSTRAINT PK_OPERATORI PRIMARY KEY(ID)
)
    STORAGE(INITIAL 459k NEXT 92k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- 5) Tabella titoli_di_studi
CREATE TABLE TITOLIDISTUDI(
    Specializzazione VARCHAR2(50),

    CONSTRAINT PK_TITOLIDISTUDI PRIMARY KEY(Specializzazione)
)
    STORAGE(INITIAL 50k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- 6) Tabella competenze
CREATE TABLE COMPETENZE(
    IDOpe NUMBER(6),
    SpecTitolo VARCHAR2(50),

    CONSTRAINT PK_COMPETENZE PRIMARY KEY(IDOpe, SpecTitolo)
)
    STORAGE(INITIAL 207k NEXT 50k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- 7) Tabella coinvolgimenti
CREATE TABLE COINVOLGIMENTI(

```

```

        IDDip NUMBER(4),
        CodSegnalazione NUMBER(10),

        CONSTRAINT PK_COINVOLGIMENTI PRIMARY KEY(IDDip, CodSegnalazione)
    )
    STORAGE(INITIAL 110k NEXT 55k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- 8) Tabella storico_segnalazioni
CREATE TABLE STORICO_SEGNALAZIONI (
    Codice NUMBER(10),
    NumeroDiTelefono NUMBER(10),
    Nome VARCHAR2(30),
    Cognome VARCHAR2(30),
    TipologiaEmergenza VARCHAR2(30),
    Provincia CHAR(2),
    Città VARCHAR2(35),
    CAP NUMBER(5),
    Via VARCHAR2(35),
    NumeroCivico NUMBER(5),
    Data_Ora TIMESTAMP(0),

    CONSTRAINT PK_STORICO_SEGNALAZIONI PRIMARY KEY(Codice)
)
    STORAGE(INITIAL 1560k NEXT 3120k MINEXTENTS 1 MAXEXTENTS 5 PCTINCREASE 5);

-- Definizione dei ruoli
-- Ruolo utente e relativi privilegi
CREATE ROLE utente; -- creazione del ruolo
GRANT CONNECT TO utente; -- attribuzione dei privilegi
GRANT INSERT, UPDATE, DELETE ON sen.UTENTI TO utente; -- attribuzione dei privilegi
-- Dopo ogni singolo GRANT, apparirà nell'output screen "Grant riuscito/a."

-- Ruolo amm_dipartimento e relativi privilegi
CREATE ROLE amm_dipartimento;
GRANT CONNECT TO amm_dipartimento;
GRANT INSERT, UPDATE, DELETE ON sen.DIPARTIMENTI TO amm_dipartimento;
GRANT INSERT, UPDATE, DELETE ON sen.OPERATORI TO amm_dipartimento;
GRANT INSERT, UPDATE, DELETE ON sen.TITOLIDISTUDI TO amm_dipartimento;
GRANT INSERT, UPDATE, DELETE ON sen.COMPETENZE TO amm_dipartimento;

-- Ruolo centralinista e relativi privilegi
CREATE ROLE centralinista;
GRANT CONNECT TO centralinista;
GRANT SELECT ON sen.SEGNALAZIONI TO centralinista;
GRANT INSERT, UPDATE, DELETE ON sen.COINVOLGIMENTI TO centralinista;

-- Creazione delle sequenze
/* Si creano le seguenti sequenze:
* 1- codSeg_seq per Codice di SEGNALAZIONI;
* 2- idDip_seq per ID di DIPARTIMENTI;
* 3- idOpe_seq per ID di OPERATORI.
*/
-- Dopo ogni singola esecuzione, apparirà nell'output screen "Creato sequence <nome_sequence>."
-- Sequenza per il codice della segnalazione

```

```

CREATE SEQUENCE codSeg_seq
    INCREMENT BY 1
    START WITH 1000000
    MINVALUE 1000000
    MAXVALUE 9999999999
    NOCYCLE;

-- Sequenza per l'ID del dipartimento
CREATE SEQUENCE idDip_seq
    INCREMENT BY 1
    START WITH 0
    MINVALUE 0
    MAXVALUE 9999
    NOCYCLE;

-- Sequenza per l'ID dell'operatore
CREATE SEQUENCE idOpe_seq
    INCREMENT BY 1
    START WITH 10000
    MINVALUE 10000
    MAXVALUE 999999
    NOCYCLE;

-- Aggiunta delle chiavi esterne
/* Si aggiungono le seguenti chiavi esterne:
* 1- IDDip:DIPARTIMENTI per OPERATORI;
* 2- NumTelefonicoUtente:UTENTI per SEGNALAZIONI;
* 3- IDDip:DIPARTIMENTI per COINVOLGIMENTI;
* 4- CodSegnalazione:SEGNALAZIONI per COINVOLGIMENTI;
* 5- IDOpe:OPERATORI per COMPETENZE;
* 6- SpecTitolo:TITOLIDISTUDI per COMPETENZE.
*/

-- Dopo ogni singola esecuzione, apparirà nell'output screen "Table <nome_table> modificato."
ALTER TABLE OPERATORI
    ADD CONSTRAINT FK_OPE_DIM FOREIGN KEY (IDDip)
    REFERENCES DIPARTIMENTI(ID)
    ON DELETE CASCADE;
    -- ON DELETE CASCADE: quando si cancella un Dipartimenti, si eliminano anche i suoi Operatori.

ALTER TABLE SEGNALAZIONI
    ADD CONSTRAINT FK_SEG_UTE FOREIGN KEY (NumTelefonicoUtente)
    REFERENCES UTENTI(NumeroDiTelefono);
    -- Nessuna operazione ON DELETE.

ALTER TABLE COINVOLGIMENTI
    ADD CONSTRAINT FK_COI_DIP FOREIGN KEY (IDDip)
    REFERENCES DIPARTIMENTI(ID)
    ON DELETE CASCADE;
    -- ON DELETE CASCADE: quando si cancella un Dipartimento, si eliminano anche i
    -- Coinvolgimenti relativi.

ALTER TABLE COINVOLGIMENTI
    ADD CONSTRAINT FK_COI_SEG FOREIGN KEY (CodSegnalazione)
    REFERENCES SEGNALAZIONI(Codice)
    ON DELETE CASCADE;

```

```

-- ON DELETE CASCADE: quando si cancella una Segnalazione, si eliminano anche i
-- Coinvolgimenti relativi.

ALTER TABLE COMPETENZE
  ADD CONSTRAINT FK_COM_OPE FOREIGN KEY (IDOpe)
  REFERENCES OPERATORI(ID)
  ON DELETE CASCADE;
-- ON DELETE CASCADE: cancello da Competenze ogni volta che si cancella un Operatore.

ALTER TABLE COMPETENZE
  ADD CONSTRAINT FK_COM_TIT FOREIGN KEY (SpecTitolo)
  REFERENCES TITOLIDISTUDI(Specializzazione)
  ON DELETE CASCADE;
-- ON DELETE CASCADE: cancello da Competenze ogni volta che si cancella un Titolo di Studi.

-----

-- SEQUENZA DI COMANDI
-- Dopo ogni singola esecuzione degli EXEC, apparirà nell'output screen
-- "Procedura PL/SQL completata correttamente."

-- Abilito la scrittura nell'output screen
SET SERVEROUT ON;

----- UTENTI
EXEC PACK_SEN.valori_iniziali_utenti;
EXEC PACK_SEN.stampa_utenti;

EXEC PACK_SEN.insert_utente(3230001111, 'Pippo', 'Baudo', 'NA', 'Napoli', 80137,
  'Via Verdi', 13);
EXEC PACK_SEN.insert_utente(3234441111, 'Mike', 'Bongiorno', 'PA', 'Palermo', 90121,
  'Piazza Mozart', 112);
EXEC PACK_SEN.stampa_utenti;

INSERT INTO UTENTI VALUES(3990001234, 'Carlo', 'Conti', 'FI', 'Firenze', 50122, 'Viale Bach', 2);
-- Cambiamone il civico
UPDATE UTENTI SET UTENTI.NumeroCivico=4 WHERE (NumeroDiTelefono=3990001234);
EXEC PACK_SEN.stampa_utenti;

-- Rimuovo Mike Bongiorno
EXEC PACK_SEN.delete_utente(3234441111);
EXEC PACK_SEN.stampa_utenti;

----- DIPARTIMENTI
EXEC PACK_SEN.valori_iniziali_dipartimenti;
EXEC PACK_SEN.stampa_dipartimenti;

EXEC PACK_SEN.insert_dipartimento(113, 'Polizia di Stato', 40, 'BO', 'Bologna', 40121,
  'Corso Navona', 1);
EXEC PACK_SEN.stampa_dipartimenti;

-- Rimuovo Polizia di Stato di Bologna, ID=7
EXEC PACK_SEN.delete_dipartimento(7);
EXEC PACK_SEN.stampa_dipartimenti;

```

```

----- OPERATORI
EXEC PACK_SEN.valori_iniziali_operatori;
EXEC PACK_SEN.stampa_operatori;
EXEC PACK_SEN.stampa_dipartimenti;

EXEC PACK_SEN.insert_operatore('Carla', 'Fracci', 'Ragioneria', 5, 'Ragioniere');
INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Scienze Politiche');
EXEC PACK_SEN.stampa_operatori;
EXEC PACK_SEN.stampa_dipartimenti;

EXEC PACK_SEN.delete_operatore(idOpe_seq.CURRVAL);
EXEC PACK_SEN.stampa_operatori;
EXEC PACK_SEN.stampa_dipartimenti;

-- Tutti gli operatori del dipartimento con ID=1 passano in quello con ID=2
UPDATE OPERATORI SET OPERATORI.IDDip=2 WHERE (IDDip=1);
EXEC PACK_SEN.stampa_operatori;
EXEC PACK_SEN.stampa_dipartimenti;

----- STAMPA TITOLI
EXEC PACK_SEN.stampa_titoli;
-- Verifichiamo il titolo di studi. Inseriamo un altro laureato in Lettere:
-- non sarà nuovamente aggiunto
EXEC PACK_SEN.insert_operatore('Roberto', 'Bolle', 'Lettere', 3, 'Sovrintendente');
EXEC PACK_SEN.stampa_titoli;

----- STAMPA COMPETENZE
EXEC PACK_SEN.stampa_competenze;

----- SEGNALAZIONI
-- Inseriamo una segnalazione: furto a casa di Pippo Baudo
EXEC PACK_SEN.insert_segna1azione(3230001111, 'Furto', 'NA', 'Napoli', 80137, 'Via Verdi',
13, CURRENT_TIMESTAMP);
EXEC PACK_SEN.stampa_segna1azioni;
-- Inseriamo una seconda segnalazione: incendio segnalato da Carlo Conti
EXEC PACK_SEN.insert_segna1azione(3990001234, 'Incendio', 'FI', 'Firenze', 50122, 'Via Escher',
212, CURRENT_TIMESTAMP);
EXEC PACK_SEN.stampa_segna1azioni;
-- Inseriamo una segnalazione in data futura
EXEC PACK_SEN.insert_segna1azione(3990001234, 'Incidente', 'TA', 'Taranto', 74121,
'Largo Caravaggio', 7, '11-DIC-22 21:09:09');
-- Verrà giustamente dato un errore!

----- STORICO SEGNALAZIONI
-- Inizialmente vuoto
EXEC PACK_SEN.stampa_storico;
-- Rimuoviamo l'utente Carlo Conti
EXEC PACK_SEN.delete_utente(3990001234);
EXEC PACK_SEN.stampa_storico;
-- Mostriamo gli utenti rimanenti sottoforma di tabella
SELECT * FROM UTENTI;
-----

-- Eliminazione delle sequenze

```

```

-- Dopo ogni singola esecuzione, apparirà nell'output screen "Sequence <nome_sequence> eliminato."
DROP SEQUENCE codSeg_seq;
DROP SEQUENCE idDip_seq;
DROP SEQUENCE idOpe_seq;

-- Revocazione dei privilegi
-- Dopo ogni singola esecuzione, apparirà nell'output screen "Revoke riuscito/a."
REVOKE INSERT, UPDATE, DELETE ON sen.DIPARTIMENTI FROM amm_dipartimento;
REVOKE INSERT, UPDATE, DELETE ON sen.OPERATORI FROM amm_dipartimento;
REVOKE SELECT ON sen.SEGNALAZIONI FROM centralinista;
REVOKE INSERT, UPDATE, DELETE ON sen.UTENTI FROM utente;
REVOKE INSERT, UPDATE, DELETE ON sen.COINVOLGIMENTI FROM centralinista;
REVOKE INSERT, UPDATE, DELETE ON sen.TITOLIDISTUDI FROM amm_dipartimento;
REVOKE INSERT, UPDATE, DELETE ON sen.COMPETENZE FROM amm_dipartimento;

-- Rimozione dei ruoli
-- Dopo ogni singola esecuzione, apparirà nell'output screen "Role <nome_role> eliminato."
DROP ROLE utente;
DROP ROLE amm_dipartimento;
DROP ROLE centralinista;

-- Eliminazione dei vincoli di integrità referenziale
-- Dopo ogni singola esecuzione, apparirà nell'output screen "Table <nome_tabella> modificato."
ALTER TABLE OPERATORI DROP CONSTRAINT FK_OPE_DIM;
ALTER TABLE SEGNALAZIONI DROP CONSTRAINT FK_SEG_UTE;
ALTER TABLE COINVOLGIMENTI DROP CONSTRAINT FK_COI_DIP;
ALTER TABLE COINVOLGIMENTI DROP CONSTRAINT FK_COI_SEG;
ALTER TABLE COMPETENZE DROP CONSTRAINT FK_COM_OPE;
ALTER TABLE COMPETENZE DROP CONSTRAINT FK_COM_TIT;

-- Eliminazione delle tabelle
-- Dopo ogni singola esecuzione, apparirà nell'output screen "Table <nome_tabella> eliminato."
DROP TABLE OPERATORI;
DROP TABLE COINVOLGIMENTI;
DROP TABLE UTENTI;
DROP TABLE DIPARTIMENTI;
DROP TABLE TITOLIDISTUDI;
DROP TABLE COMPETENZE;
DROP TABLE SEGNALAZIONI;
DROP TABLE STORICO_SEGNALAZIONI;

-- Eliminazione del package
-- Dopo l'esecuzione, apparirà nell'output screen "Package PACK_SEN eliminato."
--DROP PACKAGE PACK_SEN;

```

4.4 Package PACK_SEN

Nel progetto SEN è stato previsto il package PACK_SEN, contenente tutte le *stored procedure* che si è scelto di utilizzare.

Si noti che i vantaggi introdotti dal package sono molteplici:

- Modularità dell'architettura applicativa;
- Riutilizzabilità in progetti diversi delle stesse funzionalità;

- Occultamento delle informazioni.

Inoltre, la separazione dei file di *specification* e di *body* promuove la modularità e risolve il problema delle dichiarazioni di procedure nel rispetto della regola di visibilità.

4.4.1 Specification

Si riporta di seguito la *specification* del package PACK_SEN.

```
CREATE OR REPLACE PACKAGE PACK_SEN
IS

    PROCEDURE valori_iniziali_utenti;

-----

    PROCEDURE insert_utente(
        InNumeroDiTelefono IN NUMBER,
        InNome IN VARCHAR2,
        InCognome IN VARCHAR2,
        InProvincia IN CHAR,
        InCittà IN VARCHAR2,
        InCAP IN NUMBER,
        InVia IN VARCHAR2,
        InNumeroCivico IN NUMBER
    );

-----

    PROCEDURE stampa_utenti;

-----

    PROCEDURE delete_utente(
        InNumeroDiTelefono IN NUMBER
    );

-----

    PROCEDURE valori_iniziali_dipartimenti;

-----

    PROCEDURE insert_dipartimento(
        InNumeroEmergenza IN NUMBER,
        InTipologia IN VARCHAR2,
        InNumeroVeicoli IN NUMBER,
        InProvincia IN CHAR,
        InCittà IN VARCHAR2,
        InCAP IN NUMBER,
        InVia IN VARCHAR2,
        InNumeroCivico IN NUMBER
    );

-----

    PROCEDURE stampa_dipartimenti;
```

```

-----
PROCEDURE delete_dipartimento(
    InID IN NUMBER
);

```

```

-----
PROCEDURE valori_iniziali_operatori;

```

```

-----
PROCEDURE insert_operatore(
    InNome IN VARCHAR2,
    InCognome IN VARCHAR2,
    InSpecTitolo VARCHAR2,
    InIDDip NUMBER,
    InRuolo IN VARCHAR2
);

```

```

-----
PROCEDURE insert_titolo(
    InSpecializzazione IN TITOLIDISTUDI.Specializzazione%TYPE
);

```

```

-----
PROCEDURE insert_nuovo_titolo_operatore(
    InIDOpe IN NUMBER,
    InSpecTitolo IN VARCHAR2
);

```

```

-----
PROCEDURE stampa_operatori;

```

```

-----
PROCEDURE delete_operatore(
    InID IN OPERATORI.ID%TYPE
);

```

```

-----
PROCEDURE delete_titolo(
    InSpecializzazione IN TITOLIDISTUDI.Specializzazione%TYPE
);

```

```

-----
PROCEDURE insert_segna1azione(
    InNumTelefonicoUtente IN SEGNALAZIONI.NumTelefonicoUtente%TYPE,
    InTipologiaEmergenza IN SEGNALAZIONI.TipologiaEmergenza%TYPE,
    InProvincia IN SEGNALAZIONI.Provincia%TYPE,
    InCittà IN SEGNALAZIONI.Città%TYPE,

```

```

        InCAP IN SEGNALAZIONI.CAP%TYPE,
        InVia IN SEGNALAZIONI.Via%TYPE,
        InNumeroCivico IN SEGNALAZIONI.NumeroCivico%TYPE,
        InData_Ora IN SEGNALAZIONI.Data_Ora%TYPE
    );

-----

    PROCEDURE informazioni_segna1azione(
        InCodiceSegnalazione IN SEGNALAZIONI.Codice%TYPE,
        OutTipologiaEmergenza OUT SEGNALAZIONI.TipologiaEmergenza%TYPE,
        OutProvincia OUT SEGNALAZIONI.Provincia%TYPE,
        OutCittà OUT SEGNALAZIONI.Città%TYPE,
        OutCAP OUT SEGNALAZIONI.CAP%TYPE,
        OutVia OUT SEGNALAZIONI.Via%TYPE,
        OutNumeroCivico OUT SEGNALAZIONI.NumeroCivico%TYPE,
        OutData_Ora OUT SEGNALAZIONI.Data_Ora%TYPE
    );

-----

    PROCEDURE insert_coinvolgimento(
        InIDDip IN COINVOLGIMENTI.IDDip%TYPE,
        InCodSegnalazione IN COINVOLGIMENTI.CodSegnalazione%TYPE
    );

-----

    PROCEDURE stampa_titoli;

-----

    PROCEDURE stampa_segna1azioni;

-----

    PROCEDURE stampa_storico;

-----

    PROCEDURE stampa_competenze;

-----

END PACK_SEN;

```

4.4.2 Body

Si riporta di seguito il *body* del package PACK_SEN.

CREATE OR REPLACE PACKAGE BODY PACK_SEN IS

```

    PROCEDURE valori_iniziali_utenti
    IS
    BEGIN
        INSERT INTO UTENTI VALUES(3331231231, 'Leonardo', 'Catello',

```

```

        'NA', 'San Giorgio a Cremano', 80046,
        'Via Roma', 23);
INSERT INTO UTENTI VALUES(3334564564, 'Daiana', 'Cipollaro',
        'MI', 'Milano', 20019,
        'Via Torino', 1);
INSERT INTO UTENTI VALUES(3662342342, 'Francesco', 'Di Serio',
        'CN', 'Cuneo', 12100,
        'Via Milano', 40);
INSERT INTO UTENTI VALUES(3667897897, 'Ciro', 'Gallucci',
        'RM', 'Tivoli', 00019,
        'Via Napoli', 7);
END;

```

```

-----

PROCEDURE insert_utente(
    InNumeroDiTelefono IN NUMBER,
    InNome IN VARCHAR2,
    InCognome IN VARCHAR2,
    InProvincia IN CHAR,
    InCittà IN VARCHAR2,
    InCAP IN NUMBER,
    InVia IN VARCHAR2,
    InNumeroCivico IN NUMBER
)
IS
BEGIN
    INSERT INTO UTENTI VALUES(InNumeroDiTelefono, InNome, InCognome,
        InProvincia, InCittà, InCAP,
        InVia, InNumeroCivico);
END;

```

```

-----

PROCEDURE stampa_utenti
IS
    CURSOR cursore IS SELECT * FROM UTENTI;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella UTENTI');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' Numero di telefono: ' || record.NumeroDiTelefono);
        DBMS_OUTPUT.PUT_LINE(' Nome: ' || record.Nome);
        DBMS_OUTPUT.PUT_LINE(' Cognome: ' || record.Cognome);
        DBMS_OUTPUT.PUT_LINE(' Provincia: ' || record.Provincia);
        DBMS_OUTPUT.PUT_LINE(' Città: ' || record.Città);
        DBMS_OUTPUT.PUT_LINE(' CAP: ' || record.CAP);
        DBMS_OUTPUT.PUT_LINE(' Via: ' || record.Via);
        DBMS_OUTPUT.PUT_LINE(' Numero civico: ' || record.NumeroCivico);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;
END;

```

```

        END LOOP;
    CLOSE cursore;
END;

```

```

-----

PROCEDURE delete_utente(
    InNumeroDiTelefono IN NUMBER
)
IS
    -- Variabile per la verifica dell'esistenza dell'utente
    utente_presente NUMBER := 0;
    -- Dichiarazione dell'eccezione
    UTENTE_NON_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO utente_presente FROM UTENTI U
        WHERE U.NumeroDiTelefono = InNumeroDiTelefono;
    IF (utente_presente = 0)
    THEN
        RAISE UTENTE_NON_ESISTENTE;
    END IF;
    -- Utente trovato correttamente
    DELETE FROM UTENTI WHERE (NumeroDiTelefono = InNumeroDiTelefono);
EXCEPTION
    WHEN UTENTE_NON_ESISTENTE
    THEN
        DBMS_OUTPUT.PUT_LINE('Cancellamento rifiutato: utente non esistente.');
```

```

        RAISE_APPLICATION_ERROR(-20600,'Errore cancellamento di un utente non esistente.');
```

```

END;

```

```

-----

PROCEDURE valori_iniziali_dipartimenti
IS
BEGIN
    INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, 115, 'Vigili del Fuoco',
        0, 50,
        'NA', 'Napoli', 80100,
        'Largo Tarantini', 1);
    INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, 118, 'Pronto Soccorso',
        0, 70,
        'TO', 'Torino', 10126,
        'Corso Bramante', 88);
    INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, 113, 'Polizia di Stato',
        0, 50,
        'MI', 'Milano', 20121,
        'Via Fatebenefratelli', 11);
    INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, 112, 'Arma dei Carabinieri',
        0, 40,
        'BA', 'Bari', 70121,
        'Largo Tarantini', '1');
    INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, 117, 'Guardia di Finanza',
        0, 40,
        'RM', 'Roma', 00162,
        'Viale XXI Aprile', 51);

```

```

INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, 112, 'Esercito',
                                0, 30,
                                'PA', 'Palermo', 90134,
                                'Piazza del Parlamento', 5);

END;

```

```

-----

PROCEDURE insert_dipartimento(
    InNumeroEmergenza IN NUMBER,
    InTipologia IN VARCHAR2,
    InNumeroVeicoli IN NUMBER,
    InProvincia IN CHAR,
    InCittà IN VARCHAR2,
    InCAP IN NUMBER,
    InVia IN VARCHAR2,
    InNumeroCivico IN NUMBER
)
IS
BEGIN
    INSERT INTO DIPARTIMENTI VALUES(idDip_seq.NEXTVAL, InNumeroEmergenza, InTipologia,
                                     0, InNumeroVeicoli,
                                     InProvincia, InCittà, InCAP,
                                     InVia, InNumeroCivico);

END;

```

```

-----

PROCEDURE stampa_dipartimenti
IS
    CURSOR cursore IS SELECT * FROM DIPARTIMENTI;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella DIPARTIMENTI');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' ID: ' || record.ID);
        DBMS_OUTPUT.PUT_LINE(' Numero emergenza: ' || record.NumeroEmergenza);
        DBMS_OUTPUT.PUT_LINE(' Tipologia: ' || record.Tipologia );
        DBMS_OUTPUT.PUT_LINE(' Numero veicoli: ' || record.NumeroVeicoli);
        DBMS_OUTPUT.PUT_LINE(' Numero operatori: ' || record.NumeroOperatori);
        DBMS_OUTPUT.PUT_LINE(' Provincia: ' || record.Provincia);
        DBMS_OUTPUT.PUT_LINE(' Città: ' || record.Città);
        DBMS_OUTPUT.PUT_LINE(' CAP: ' || record.CAP);
        DBMS_OUTPUT.PUT_LINE(' Via: ' || record.Via);
        DBMS_OUTPUT.PUT_LINE(' Numero civico: ' || record.NumeroCivico );
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;
    CLOSE cursore;
END;

```

```

PROCEDURE delete_dipartimento(
    InID IN NUMBER
)
IS
    -- Variabile per la verifica dell'esistenza del dipartimento
    dipartimento_presente NUMBER := 0;
    -- Dichiarazione dell'eccezione
    DIPARTIMENTO_NON_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO dipartimento_presente FROM DIPARTIMENTI D
        WHERE D.ID = InID;
    IF (dipartimento_presente = 0)
    THEN
        RAISE DIPARTIMENTO_NON_ESISTENTE;
    END IF;
    -- Dipartimento trovato correttamente
    DELETE FROM DIPARTIMENTI WHERE (ID = InID);
EXCEPTION
    WHEN DIPARTIMENTO_NON_ESISTENTE
    THEN
        DBMS_OUTPUT.PUT_LINE('Cancellamento rifiutato: dipartimento non esistente.');
```

```

        RAISE_APPLICATION_ERROR(-20601,'Errore cancellamento di dipartimento non esistente.');
```

```

END;
```

```

PROCEDURE valori_iniziali_operatori
IS
BEGIN
    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, 'Pietro', 'Tornindietro',
        1, 'Pompieri');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Scienze Motorie');

    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, 'Maria', 'Salvarezza',
        2, 'Infermiere');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Infermieristica');

    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, 'Federica', 'Varlese',
        3, 'Poliziotto');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Diploma scuola secondaria di II grado');

    -- Operatore con due titoli di studio
    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, 'Fabio', 'Cristalli',
        4, 'Carabiniere');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Perito Agrario');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Lettere');

    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, 'Umberto', 'Pirozzi',
        5, 'Maresciallo di Finanza');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Scienze Politiche');

    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, 'Vittorio', 'Solombrino',
        6, 'Artificiere');
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, 'Ingegneria meccanica');
END;
```

```

-----

PROCEDURE insert_operatore(
    InNome IN VARCHAR2,
    InCognome IN VARCHAR2,
    InSpecTitolo VARCHAR2,
    InIDDip NUMBER,
    InRuolo IN VARCHAR2
)
IS
BEGIN
    INSERT INTO OPERATORI VALUES(idOpe_seq.NEXTVAL, InNome, InCognome,
                                   InIDDip, InRuolo);
    INSERT INTO COMPETENZE VALUES(idOpe_seq.CURRVAL, InSpecTitolo);
END;

```

```

-----

PROCEDURE insert_titolo(
    InSpecializzazione IN TITOLIDISTUDI.Specializzazione%TYPE
)
IS
BEGIN
    INSERT INTO TITOLIDISTUDI VALUES(InSpecializzazione);
END;

```

```

-----

PROCEDURE insert_nuovo_titolo_operatore(
    InIDOpe IN NUMBER,
    InSpecTitolo IN VARCHAR2
)
IS
BEGIN
    INSERT INTO COMPETENZE VALUES(InIDOpe, InSpecTitolo);
END;

```

```

-----

PROCEDURE stampa_operatori
IS
    CURSOR cursore IS SELECT * FROM OPERATORI;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella OPERATORI');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' ID: ' || record.ID);
        DBMS_OUTPUT.PUT_LINE(' Nome: ' || record.Nome);
        DBMS_OUTPUT.PUT_LINE(' Cognome: ' || record.Cognome);
    END LOOP;
END;

```



```

        DBMS_OUTPUT.PUT_LINE(' ID dipartimento: ' || record.IDDip);
        DBMS_OUTPUT.PUT_LINE(' Ruolo: ' || record.Ruolo);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;
    CLOSE cursore;
END;

```

```

PROCEDURE delete_operatore(
    InID IN OPERATORI.ID%TYPE
)
IS
    -- Variabile per la verifica dell'esistenza dell'operatore
    operatore_presente NUMBER := 0;
    -- Dichiarazione dell'eccezione
    OPERATORE_NON_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO operatore_presente FROM OPERATORI O
        WHERE O.ID = InID;
    IF (operatore_presente = 0)
    THEN
        RAISE OPERATORE_NON_ESISTENTE;
    END IF;
    -- Operatore trovato correttamente
    DELETE FROM OPERATORI WHERE (ID = InID);
EXCEPTION
    WHEN OPERATORE_NON_ESISTENTE
    THEN
        DBMS_OUTPUT.PUT_LINE('Cancellamento rifiutato: operatore non esistente.');
```

```

PROCEDURE delete_titolo(
    InSpecializzazione IN TITOLIDISTUDI.Specializzazione%TYPE
)
IS
    -- Variabile per la verifica dell'esistenza del titolo di studi
    titolo_presente NUMBER := 0;
    -- Dichiarazione dell'eccezione
    TITOLO_NON_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO titolo_presente FROM TITOLIDISTUDI T
        WHERE T.Specializzazione = InSpecializzazione;
    IF (titolo_presente = 0)
    THEN
        RAISE TITOLO_NON_ESISTENTE;
    END IF;
    -- titolo di studi trovato correttamente
    DELETE FROM TITOLIDISTUDI WHERE (Specializzazione = InSpecializzazione);
EXCEPTION
    WHEN TITOLO_NON_ESISTENTE
    THEN
        DBMS_OUTPUT.PUT_LINE('Cancellamento rifiutato: titolo non esistente.');
```

```

        RAISE_APPLICATION_ERROR(-20603,'Errore cancellamento di un titolo non esistente.');
```

END;

```

-----

PROCEDURE insert_segnalazione(
    InNumTelefonicoUtente IN SEGNALAZIONI.NumTelefonicoUtente%TYPE,
    InTipologiaEmergenza IN SEGNALAZIONI.TipologiaEmergenza%TYPE,
    InProvincia IN SEGNALAZIONI.Provincia%TYPE,
    InCittà IN SEGNALAZIONI.Città%TYPE,
    InCAP IN SEGNALAZIONI.CAP%TYPE,
    InVia IN SEGNALAZIONI.Via%TYPE,
    InNumeroCivico IN SEGNALAZIONI.NumeroCivico%TYPE,
    InData_Ora IN SEGNALAZIONI.Data_Ora%TYPE
)
IS
BEGIN
    INSERT INTO SEGNALAZIONI VALUES(codSeg_seq.NEXTVAL, InTipologiaEmergenza,
                                     InNumTelefonicoUtente, InProvincia,
                                     InCittà, InCAP, InVia, InNumeroCivico,
                                     InData_Ora);

END;
```

```

-----

PROCEDURE informazioni_segnalazione(
    InCodiceSegnalazione IN SEGNALAZIONI.Codice%TYPE,
    OutTipologiaEmergenza OUT SEGNALAZIONI.TipologiaEmergenza%TYPE,
    OutProvincia OUT SEGNALAZIONI.Provincia%TYPE,
    OutCittà OUT SEGNALAZIONI.Città%TYPE,
    OutCAP OUT SEGNALAZIONI.CAP%TYPE,
    OutVia OUT SEGNALAZIONI.Via%TYPE,
    OutNumeroCivico OUT SEGNALAZIONI.NumeroCivico%TYPE,
    OutData_Ora OUT SEGNALAZIONI.Data_Ora%TYPE
)
IS
BEGIN
    SELECT S.TipologiaEmergenza, S.Provincia, S.Città, S.CAP, S.Via,
           S.NumeroCivico, S.Data_Ora
    INTO OutTipologiaEmergenza, OutProvincia, OutCittà, OutCAP, OutVia,
         OutNumeroCivico, OutData_Ora
    FROM SEGNALAZIONI S
    WHERE S.Codice = InCodiceSegnalazione;

    -- Stampa risultato del join
    DBMS_OUTPUT.PUT_LINE('INFORMAZIONI RELATIVE ALLA SEGNALAZIONE NUMERO '
                          || InCodiceSegnalazione || ':'');
    DBMS_OUTPUT.PUT_LINE('  Tipologia emergenza: ' || OutTipologiaEmergenza);
    DBMS_OUTPUT.PUT_LINE('  Provincia: ' || OutProvincia);
    DBMS_OUTPUT.PUT_LINE('  Città: ' || OutCittà);
    DBMS_OUTPUT.PUT_LINE('  CAP: ' || OutCAP);
    DBMS_OUTPUT.PUT_LINE('  Via: ' || OutVia);
    DBMS_OUTPUT.PUT_LINE('  Numero civico: ' || OutNumeroCivico);
    DBMS_OUTPUT.PUT_LINE('  Data e ora: ' || OutData_Ora);

END;
```

```

-----

PROCEDURE insert_coinvolgimento(
    InIDDip IN COINVOLGIMENTI.IDDip%TYPE,
    InCodSegnalazione IN COINVOLGIMENTI.CodSegnalazione%TYPE
)
IS
BEGIN
    INSERT INTO COINVOLGIMENTI VALUES (InIDDip, InCodSegnalazione);
END;

```

```

-----

PROCEDURE stampa_titoli
IS
    CURSOR cursore IS SELECT * FROM TITOLIDISTUDI;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella TITOLIDISTUDI');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' Specializzazione: ' || record.Specializzazione);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;
    CLOSE cursore;
END;

```

```

-----

PROCEDURE stampa_segnalazioni
IS
    CURSOR cursore IS SELECT * FROM SEGNALAZIONI;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella SEGNALAZIONI');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' Codice: ' || record.Codice);
        DBMS_OUTPUT.PUT_LINE(' Tipologia emergenza: ' || record.TipologiaEmergenza);
        DBMS_OUTPUT.PUT_LINE(' Numero telefonico utente: ' || record.NumTelefonicoUtente);
        DBMS_OUTPUT.PUT_LINE(' Provincia: ' || record.Provincia);
        DBMS_OUTPUT.PUT_LINE(' Città: ' || record.Città);
        DBMS_OUTPUT.PUT_LINE(' CAP: ' || record.CAP);
        DBMS_OUTPUT.PUT_LINE(' Via: ' || record.Via);
        DBMS_OUTPUT.PUT_LINE(' Numero civico: ' || record.NumeroCivico);
    END LOOP;
END;

```

```

        DBMS_OUTPUT.PUT_LINE(' Data e ora: ' || record.Data_Ora);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;
    CLOSE cursore;
END;

```

```

PROCEDURE stampa_storico
IS
    CURSOR cursore IS SELECT * FROM STORICO_SEGNALAZIONI;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella STORICO_SEGNALAZIONI');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' Codice: ' || record.Codice);
        DBMS_OUTPUT.PUT_LINE(' Numero telefonico utente: ' || record.NumeroDiTelefono);
        DBMS_OUTPUT.PUT_LINE(' Nome: ' || record.Nome);
        DBMS_OUTPUT.PUT_LINE(' Cognome: ' || record.Cognome);
        DBMS_OUTPUT.PUT_LINE(' Tipologia emergenza: ' || record.TipologiaEmergenza);
        DBMS_OUTPUT.PUT_LINE(' Provincia emergenza: ' || record.Provincia);
        DBMS_OUTPUT.PUT_LINE(' Città emergenza: ' || record.Città);
        DBMS_OUTPUT.PUT_LINE(' CAP emergenza: ' || record.CAP);
        DBMS_OUTPUT.PUT_LINE(' Via emergenza: ' || record.Via);
        DBMS_OUTPUT.PUT_LINE(' Numero civico emergenza: ' || record.NumeroCivico);
        DBMS_OUTPUT.PUT_LINE(' Data e ora emergenza: ' || record.Data_Ora);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;
    CLOSE cursore;
END;

```

```

PROCEDURE stampa_competenze
IS
    CURSOR cursore IS SELECT * FROM COMPETENZE;
    record cursore%ROWTYPE;
    counter number:=0;
BEGIN
    OPEN cursore;
    DBMS_OUTPUT.PUT_LINE('Tabella COMPETENZE');
    LOOP
        counter := counter + 1;
        FETCH cursore INTO record;
        EXIT WHEN cursore%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Riga n# ' || counter);
        DBMS_OUTPUT.PUT_LINE(' ID operatore: ' || record.IDOpe);
        DBMS_OUTPUT.PUT_LINE(' Titolo di studi: ' || record.SpecTitolo);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;

```

```

        CLOSE cursore;
    END;

```

```

END PACK_SEN;

```

4.5 Trigger

Il progetto SEN è ricco di trigger, specifiche per le quali una data azione o funzione deve attivarsi in maniera automatica.

Questi ultimi sono infatti applicazioni memorizzate nel DBMS, eseguite automaticamente quando accade un evento, a patto che la condizione sulla quale si basano – qualora sia presente – venga rispettata.

Si riporta, a conclusione di tale trattazione, ciascun trigger separatamente, corredato da:

- Tabella monitorata;
- Modo di esecuzione (BEFORE o AFTER);
- Evento monitorato (INSERT, UPDATE o AFTER);
- Granularità (*statement-level* o *row-level*).

Si partirà dai trigger di verifica, per poi proseguire con quelli di inserimento, modifica e rimozione. Si terminerà con il trigger che consente di rispettare la regola di business, definita nella sezione [1.2].

4.5.1 _triggerSEN_per_verifica_insert_utente

- Nome trigger: VERIFICA_INSERT_UTENTI;
- Tabella monitorata: UTENTI;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER VERIFICA_INSERT_UTENTI
BEFORE INSERT ON UTENTI
FOR EACH ROW
DECLARE
    -- Dichiarazione variabile per verifica dell'esistenza dell'utente
    numero_utenti NUMBER := 0;
    -- Dichiarazione dell'eccezione
    UTENTE_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO numero_utenti FROM UTENTI U
        WHERE U.NumeroDiTelefono = :NEW.NumeroDiTelefono;
    IF (numero_utenti > 0)
    THEN
        RAISE UTENTE_ESISTENTE;
    END IF;
EXCEPTION
    WHEN UTENTE_ESISTENTE
    THEN
        DBMS_OUTPUT.PUT_LINE('Inserimento rifiutato: utente già esistente.');
```

```

        RAISE_APPLICATION_ERROR(-20500,'Errore inserimento di utente già esistente.');
```

```

END;
```

4.5.2 _triggerSEN_per_verifica_insert_titolo

- Nome trigger: VERIFICA_INSERT_TITOLI;
- Tabella monitorata: TITOLIDISTUDI;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```
CREATE OR REPLACE TRIGGER VERIFICA_INSERT_TITOLI
BEFORE INSERT ON TITOLIDISTUDI
FOR EACH ROW
DECLARE
    -- Dichiarazione variabile per verifica dell'esistenza del titolo di studi
    numero_titoli NUMBER := 0;
    -- Dichiarazione dell'eccezione
    TITOLO_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO numero_titoli FROM TITOLIDISTUDI T
        WHERE T.Specializzazione = :NEW.Specializzazione;
    IF (numero_titoli > 0)
    THEN
        RAISE TITOLO_ESISTENTE;
    END IF;
EXCEPTION
WHEN TITOLO_ESISTENTE
THEN
    DBMS_OUTPUT.PUT_LINE('Inserimento rifiutato: titolo già esistente.');
```

```
    RAISE_APPLICATION_ERROR(-20503,'Errore inserimento di titolo già esistente.');
```

```
END;
```

4.5.3 _triggerSEN_per_verifica_insert_operatore

- Nome trigger: VERIFICA_INSERT_OPERATORI;
- Tabella monitorata: OPERATORI;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```
CREATE OR REPLACE TRIGGER VERIFICA_INSERT_OPERATORI
BEFORE INSERT ON OPERATORI
FOR EACH ROW
DECLARE
    -- Dichiarazione variabile per verifica dell'esistenza dell'operatore
    numero_operatori NUMBER := 0;
    -- Dichiarazione dell'eccezione
    OPERATORE_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO numero_operatori FROM OPERATORI O WHERE O.ID = :NEW.ID;
    IF (numero_operatori > 0)
    THEN
        RAISE OPERATORE_ESISTENTE;
```

```

        END IF;
    EXCEPTION
        WHEN OPERATORE_ESISTENTE
        THEN
            DBMS_OUTPUT.PUT_LINE('Inserimento rifiutato: operatore già esistente.');
```

```

            RAISE_APPLICATION_ERROR(-20502,'Errore inserimento di operatore già esistente.');
```

```

    END;
```

4.5.4 _triggerSEN_per_verifica_insert_dipartimento

- Nome trigger: VERIFICA_INSERT_DIPARTIMENTI;
- Tabella monitorata: DIPARTIMENTI;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER VERIFICA_INSERT_DIPARTIMENTI
BEFORE INSERT ON DIPARTIMENTI
FOR EACH ROW
DECLARE
    -- Dichiarazione variabile per verifica dell'esistenza del dipartimento
    numero_dipartimenti NUMBER := 0;
    -- Dichiarazione dell'eccezione
    DIPARTIMENTO_ESISTENTE EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO numero_dipartimenti FROM DIPARTIMENTI D WHERE D.ID = :NEW.ID;
    IF (numero_dipartimenti > 0)
    THEN
        RAISE DIPARTIMENTO_ESISTENTE;
    END IF;
EXCEPTION
    WHEN DIPARTIMENTO_ESISTENTE
    THEN
        DBMS_OUTPUT.PUT_LINE('Inserimento rifiutato: dipartimento già esistente.');
```

```

        RAISE_APPLICATION_ERROR(-20501,'Errore inserimento di dipartimento già esistente.');
```

```

    END;
```

4.5.5 _triggerSEN_per_verifica_data

- Nome trigger: VERIFICA_DATA;
- Tabella monitorata: SEGNALAZIONI;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER VERIFICA_DATA
BEFORE INSERT ON SEGNALAZIONI
FOR EACH ROW
DECLARE
    -- Variabile per memorizzare il time stamp attuale
    data_ora_attuale SEGNALAZIONI.Data_Ora%TYPE;
    -- Dichiarazione dell'eccezione
```

```

        DATA_ORA_FUTURA EXCEPTION;
BEGIN
    SELECT CURRENT_TIMESTAMP INTO data_ora_attuale FROM DUAL;
    IF (data_ora_attuale < :NEW.Data_Ora)
    THEN
        RAISE DATA_ORA_FUTURA;
    END IF;
EXCEPTION
    WHEN DATA_ORA_FUTURA
    THEN
        DBMS_OUTPUT.PUT_LINE('Inserimento rifiutato: data o ora non valida.');
```

```

        RAISE_APPLICATION_ERROR(-20900,'Errore inserimento di data o ora futura.');
```

```

END;
```

4.5.6 _triggerSEN_per_verifica_titolodistudi

- Nome trigger: VERIFICA_TITOLODISTUDI;
- Tabella monitorata: COMPETENZE;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER VERIFICA_TITOLODISTUDI
BEFORE INSERT ON COMPETENZE
FOR EACH ROW

DECLARE
    titolo_esistente number := 0;
BEGIN
    SELECT COUNT(*)
    INTO titolo_esistente
    FROM TITOLIDISTUDI T
    WHERE T.Specializzazione = :NEW.SpecTitolo;
    IF (titolo_esistente = 0)
    THEN
        INSERT INTO TITOLIDISTUDI VALUES (:NEW.SpecTitolo) ;
        DBMS_OUTPUT.PUT_LINE('Inserito il titolo di studi ' || :NEW.SpecTitolo
            || ' nella tabella TITOLIDISTUDI');
```

```

    END IF;
END;
```

4.5.7 _triggerSEN_per_insert_operatore

- Nome trigger: VINCOLI_OPERATORI_INSERT;
- Tabella monitorata: OPERATORI;
- Modo di esecuzione: AFTER;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER VINCOLI_OPERATORI_INSERT
AFTER INSERT ON OPERATORI
FOR EACH ROW
```



```

BEGIN
    UPDATE DIPARTIMENTI D SET D.NumeroOperatori = D.NumeroOperatori + 1
    WHERE D.ID = :NEW.IDDip;

    DBMS_OUTPUT.PUT_LINE('Aggiornato numero operatori in tabella DIPARTIMENTI
        per il dipartimento con ID ' || :NEW.IDDip);
END;

```

4.5.8 _triggerSEN_per_insert_coinvolgimento

- Nome trigger: VINCOLI_COINVOLGIMENTI_INSERT;
- Tabella monitorata: COINVOLGIMENTI;
- Modo di esecuzione: BEFORE;
- Evento monitorato: INSERT;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER VINCOLI_COINVOLGIMENTI_INSERT
BEFORE INSERT ON COINVOLGIMENTI
FOR EACH ROW
DECLARE
    -- Numero massimo di dipartimenti coinvolti in una segnalazione
    max_Dipartimenti constant number := 6;

    -- Variabile in cui inserire il numero di dipartimenti coinvolti in una segnalazione
    numero_Dipartimenti_coinvolti number := 0;

    -- Dichiarazione dell'eccezione
    OverflowDipartimenti exception;

BEGIN
    SELECT COUNT(*) INTO numero_Dipartimenti_coinvolti
    FROM Coinvolgimenti C
    WHERE C.CodSegnalazione = :NEW.CodSegnalazione;

    DBMS_OUTPUT.PUT_LINE('Verifica inserimento del coinvolgimento.');
```

IF (numero_Dipartimenti_coinvolti >= max_Dipartimenti)

 THEN RAISE OverflowDipartimenti;

END IF;

-- Inserimento corretto

DBMS_OUTPUT.PUT_LINE('Inserimento accettato.');

```

EXCEPTION
    WHEN OverflowDipartimenti
    THEN DBMS_OUTPUT.PUT_LINE('Inserimento rifiutato: overflow di dipartimenti
        coinvolti nella segnalazione numero ' || :NEW.CodSegnalazione || '.');
    RAISE_APPLICATION_ERROR(-20000,'Errore massimo numero di dipartimenti
        coinvolti in una segnalazione.');
```

END;

4.5.9 _triggerSEN_per_update_operatore

- Nome trigger: VINCOLI_OPERATORI_UPDATE;

- Tabella monitorata: OPERATORI;
- Modo di esecuzione: AFTER;
- Evento monitorato: UPDATE;
- Granularità: *row-level*.

```
CREATE OR REPLACE TRIGGER VINCOLI_OPERATORI_UPDATE
AFTER UPDATE ON OPERATORI
FOR EACH ROW

BEGIN
    -- Incremento del numero di operatori del nuovo dipartimento
    UPDATE DIPARTIMENTI D SET D.NumeroOperatori = D.NumeroOperatori + 1
    WHERE D.ID = :NEW.IDDip;

    -- Decremento del numero di operatori del vecchio dipartimento
    UPDATE DIPARTIMENTI D SET D.NumeroOperatori = D.NumeroOperatori - 1
    WHERE D.ID = :OLD.IDDip;

    DBMS_OUTPUT.PUT_LINE('Aggiornato numero operatori in tabella DIPARTIMENTI per
        i dipartimenti con ID ' || :NEW.IDDip || ' e ' || :OLD.IDDip);
END;
```

4.5.10 _triggerSEN_per_delete_operatore

- Nome trigger: VINCOLI_OPERATORI_DELETE;
- Tabella monitorata: OPERATORI;
- Modo di esecuzione: AFTER;
- Evento monitorato: DELETE;
- Granularità: *row-level*.

```
CREATE OR REPLACE TRIGGER VINCOLI_OPERATORI_DELETE
AFTER DELETE ON OPERATORI
FOR EACH ROW

BEGIN
    UPDATE DIPARTIMENTI D SET D.NumeroOperatori = D.NumeroOperatori - 1
    WHERE D.ID = :OLD.IDDip;

    DBMS_OUTPUT.PUT_LINE('Aggiornato numero operatori in tabella DIPARTIMENTI per
        il dipartimento con ID ' || :OLD.IDDip);
END;
```

4.5.11 _triggerSEN_per_storico_segnalazioni

- Nome trigger: CONSERVA_SEGNALAZIONI;
- Tabella monitorata: UTENTI;
- Modo di esecuzione: AFTER;
- Evento monitorato: DELETE;
- Granularità: *row-level*.

```

CREATE OR REPLACE TRIGGER CONSERVA_SEGNALAZIONI
AFTER DELETE ON UTENTI
FOR EACH ROW
DECLARE
    tCodice SEGNALAZIONI.Codice%TYPE;
    tTipologiaEmergenza SEGNALAZIONI.TipologiaEmergenza%TYPE;

    tVia SEGNALAZIONI.Via%TYPE;
    tNumeroCivico SEGNALAZIONI.NumeroCivico%TYPE;
    tCittà SEGNALAZIONI.Città%TYPE;
    tProvincia SEGNALAZIONI.Provincia%TYPE;
    tCAP SEGNALAZIONI.CAP%TYPE;
    tData_Ora SEGNALAZIONI.Data_Ora%TYPE;

    cursor t_set IS SELECT  S.Codice, S.TipologiaEmergenza,
                           S.Via, S.NumeroCivico, S.Città,
                           S.Provincia, S.CAP, S.Data_Ora
    FROM      SEGNALAZIONI S
    WHERE     S.NumTelefonicoUtente=:old.NumeroDiTelefono;
BEGIN
    OPEN t_set;
    LOOP
        FETCH t_set INTO      tCodice, tTipologiaEmergenza,
                               tVia, tNumeroCivico, tCittà, tProvincia, tCAP, tData_Ora;
        EXIT WHEN t_set%NOTFOUND;
        INSERT INTO STORICO_SEGNALAZIONI
        VALUES( tCodice, :OLD.NumeroDiTelefono, :OLD.Nome, :OLD.Cognome, tTipologiaEmergenza,
                tProvincia, tCittà, tCAP, tVia, tNumeroCivico, tData_Ora);
    END LOOP;
    CLOSE t_set;
    DELETE FROM SEGNALAZIONI WHERE NumTelefonicoUtente=:old.NumeroDiTelefono;
END;

```