Documentazione progetto "*Soccorso*" – Laboratorio Basi di Dati

A.A. 2024/2025

Membri del Team

Cognome e Nome	Matricola	Email
Scappa Marco	279487	marco.scappa@student.univaq.it
Candeloro Leonardo	281112	leonardo.candeloro@student.univaq.it

Data di consegna del progetto: *11/06/2025*

Sommario

A – Contributi al progetto

Questa tabella riporta i contributi dei due componenti del gruppo al progetto. Lo sviluppo del progetto è stato portato avanti da entrambi i mebri del progetto in egual modo.

Membro gruppo	Contributo
Scappa Marco	- DDL - Creazione dei Trigger - Intreffciamento JDBC - Contributo creazione procedure, funzioni e viste (QL) - Implementazione query SQL in Java - Realizzazione documentazione
Candeloro Leonardo	- DDL - Creazione trigger - Interfacciamento JDBC - Contributo creazione procedure, funzioni e viste (QL) - Implementazione query SQL in Java - Realizzazione documentazione

B – Struttura progetto

I diversi file del progetto (SQL) hanno i seguenti scopi:

- **tables.sql** contiene le istruzioni necessarie a creare tutte le tabelle necessarie al progetto
- **dropTables.sql** contiene (in uno specifico ordine) le istruzioni necessrie a eliminare le tabelle dalla base di dati
- **triggers.sql** contiene tutti i trigger definiti all'interno della base di dati per garantire l'integrità dei dati
- **views.sql** contiene delle viste utili per effettuare alcune operazioni all'interno della base di dati
- **operations/insertOperations.sql** contiene una procedura per ogni tabella che consente l'inserimento di una riga in quest'ultima
- **operations/selectOperations.sql** contiene due procedure per ogni tabelle utili a selezionare tutte le righe di una tabella oppure una specifica riga tramite ID (chiave primaria)
- **operations/projectOperations.sql** contiene le procedure e funzioni che implementano le fiunzionalità richieste dalla specifica, oltre a funzioni ausiliarie
- **soccorso/src** contiene tutti i file necessari a eseguire la connessione alla base di dati tramite JDBC

1 – Formalizzazione dei requisiti

1.1 – Specifiche del progetto

Il database ***Soccorso*** rappresenta una generica base di dati per la ricezione e la gestione di ***richieste di soccorso***. La tipologia di soccorso offerto non ci interessa: ci concentreremo solo sul modo generale di realizzare questo tipo di database.

Il database conterrà per prima cosa le informazioni su due categorie di utenza: gli ***amministratori*** (che configureranno il sistema, smisteranno le richieste e le monitoreranno) e gli ***operatori*** (a cui verranno inviate le richieste e che le gestiranno in prima persona). Gli amministratori potranno creare account per nuovi amministratori ed operatori. Per entrambe le categorie di utenza, oltre ai dati anagrafici, dovrà essere possibile inserire delle informazioni extra quali le ***patenti*** possedute (A, B, C, nautica,...) e una lista (generica) di ***abilità*** (ad esempio un operatore potrebbe avere un diploma infermieristico, un altro potrebbe essere un elettricista, ecc.) utili per deciderne l'assegnazione alle missioni.

Per effettuare le operazioni di soccorso, gli operatori avranno a disposizione dei ***mezzi*** (auto, ambulanze, autopompe... dipende dal tipo di emergenza che verrà gestita effettivamente dal sito) e dei ***materiali*** (kit medici, scale, estintori,...). Tali elementi saranno censiti nel database (sempre per essere molto generici, essi avranno solo un nome e una descrizione), e gli amministratori potranno aggiungerli, modificarli o eliminarli.

Le ***richieste*** di soccorso immagazzinate nel database dovranno essere necessariamente accompagnate da una breve descrizione, dall'indicazione della posizione (indirizzo, coordinate, ecc.), dal nome e dell'indirizzo email del ***segnalante***, e potranno essere opzionalmente corredate da una foto. Inoltre, per evitare spam e attacchi vari, il sistema dovrà tenere traccia quantomeno dell'indirizzo IP di origine delle richieste. Infine, ogni richiesta *inviata*, prima di diventare *attiva*, dovrà essere *convalidata*. A questo scopo, alla richiesta verrà associata una stringa lunga e casuale che sarà poi usata per costruire un link inviato per email al segnalante. Cliccando tale link, la richiesta verrà marcata come attiva nel database.

Le richieste, in base al loro stato di gestione, potranno essere in stato attivo (inviate e convalidate), in corso (gestite) e chiuso (concluse). Le richieste attive potranno essere ignorate (annullate) o gestite creando una ***missione***. Tale missione avrà associati la richiesta scatenante, un obiettivo, una posizione, una ***squadra*** (composta da almeno un operatore ***caposquadra*** e da zero o più altri operatori), zero o più mezzi, zero o più materiali, oltre che ovviamente un timestamp di inizio.

Gli amministratori potranno in ogni momento inserire degli ***aggiornamenti*** (blocchi di testo descrittivo) in una missione, ciascuno associato con il timestamp di immissione).

Infine, gli amministratori (a seguito di un'opportuna comunicazione da parte degli operatori) potranno marcare una missione come conclusa (chiusa), inserendo data/ora di fine, un generico ***livello si successo*** (anche questo dipendente dal tipo di soccorso, possiamo genericamente usare un numero che va da 0=fallimento a 5=successo pieno) e dei commenti opzionali relativi all'intervento eseguito.

Ogni elemento coinvolto nelle missioni (operatori, mezzi, materiali) dovrà essere dotato anche di un proprio **storico** delle missioni in cui è stato coinvolto.

1.1.1 – Operazioni da realizzare

Di seguito sono illustrate schematicamente le operazioni previste sulla base di dati, ciascuna da realizzare tramite una query (o, se necessario, tramite più query, *opzionalmente* racchiuse in una *stored procedure*). Ovviamente, ogni ulteriore raffinamento o arricchimento di queste specifiche aumenterà il valore del progetto.

1. *Inserimento* di una richiesta di soccorso.
2. *Creazione* di una missione connessa a una richiesta di soccorso attiva.
3. *Chiusura* di una missione.
4. *Estrazione* della lista degli operatori non coinvolti in missioni in corso.
5. *Calcolo* del numero di missioni svolte da un operatore.
6. *Calcolo* del tempo medio di svolgimento delle missioni (*dalla creazione alla chiusura*) in un anno specifico o per ciascun caposquadra.
7. *Calcolo* del numero di richieste provenienti da un certo soggetto segnalante (identificato dall'indirizzo email) o da un certo indirizzo IP nelle ultime 36 ore.
8. *Calcolo* del tempo totale di impiego in missione di un certo operatore (*cioè somma delle durata delle missioni in cui è stato coinvolto*)
9. *Estrazione* delle missioni svoltesi negli ultimi tre anni nello stesso luogo di una missione data.
10. *Estrazione* della lista delle richieste di soccorso chiuse con risultato non totalmente positivo (livello di successo minore di 5).
11. *Estrazione* degli operatori maggiormente coinvolti nelle richieste di soccorso chiuse con risultato non totalmente positivo (*calcolate come alla query precedente*).
12. *Estrazione* dello storico delle missioni in cui è stato coinvolto un certo mezzo.
13. *Calcolo* delle ore d'uso di un certo materiale (*supponiamo che il tempo d'uso uso corrisponda alla durata totale della missione in cui è stato assegnato*).

È possibile inserire procedure di gestione addizionali che si ritengano utili.

1.2 – Analisi dei requisiti

Dall'analisi della specifica abbiamo identificato le seguenti entità:

- **Abilità**
- **Patente**
- **Operatore**
- **Amministratore**
- **Richiesta**
- **Missione**
- **Squadra**
- **Mezzo**
- **Materiale**

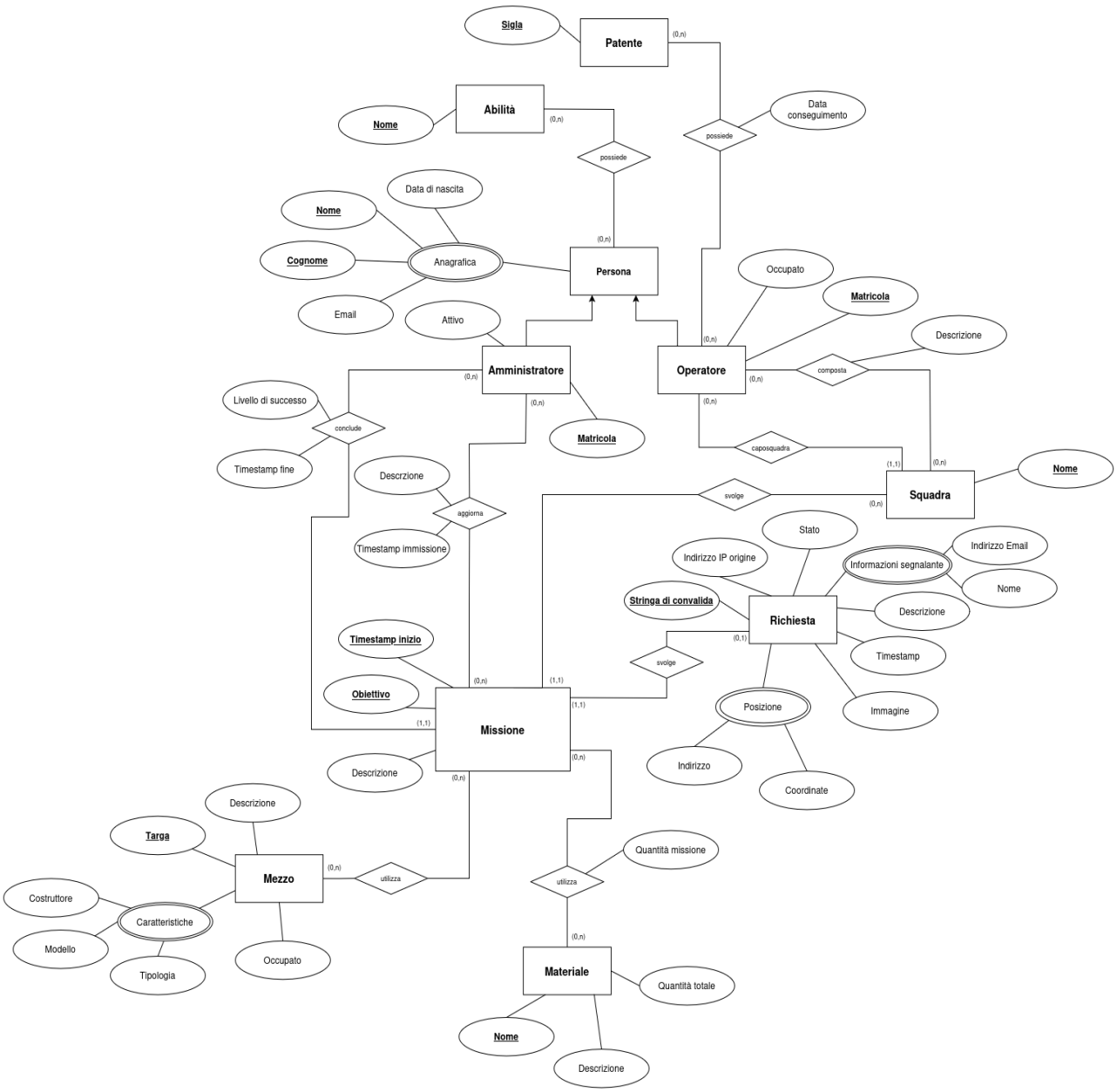
Per ogni entità abbiamo individuato degli attributi che le definisca e descriva, e che verranno riportati qui:

- **Abilità**
  - Nome
- **Patente**
  - Sigla
- **Operatore**
  - Nome
  - Cognome
  - Data di nascita
  - Occupato
  - email
  - matricola
- **Amministratore**
  - Nome
  - Cognome
  - Data di Nascita
  - email
  - Matricola
- **Richiesta**
  - Stringa di convalida
  - Indirizzo IP di origine
  - stato
  - Nome del segnalante
  - Email del segnalante
  - Data e ora di arrivo
  - Immagine
  - Didascalia immagine
  - Descrizione
  - Indirizzo
  - Coordinate
- **Missione**
  - Data e ora di inizio
  - Obiettivo
  - Descrizione
- **Squadra**
  - Nome

- **Mezzo**
  - Targa
  - Costruttore
  - Modello
  - Tipologia
  - Descrizione
  - Occupato

- **Materiale**
  - Nome
  - Descrizione
  - Quantità totale

2 – Modello Entità-Relazione (ER)



In caso la lettura del diagramma entità relazione risultasse difficoltosa da questa documentazione, ne abbiamo fornita una versione in alta qualità a questa location: *./soccorso/documentazione/modelli/* sotto il nome di **ER_soccorso.svg**.

3 – Formalizzazione dei vincoli non esprimibili nel modello ER

Di seguito elenchiamo i vincoli che sono stati identificati durante la creazione del diagramma ER:

Abilità
- Il suo attributo "*nome*" deve essere **non nullo** e **unico** in quanto chiave primaria

Sigla
- Il suo attributo "*sigla*" deve essere **non nullo** e **unico** in quanto chiave primaria

Operatore
- Il suo attributo "*matricola*" deve essere **non nullo** e **unico** in quanto chiave primaria
- L'attributo "*data di nascita*" deve avere un valore tale che l'età dell'operatore sia compresa tra 18 e 70 anni
- L'attributo "*occupato*" serve a gestire la disponibilità degli operatori
  - quando una viene creata una missione tutti gli operatori della squadra associata a tale missione verranno settati come "occupati"
  - quando la missione termina (o viene annullata) tutti gli operatori verranno settati come "non occupati"

Amministratore
- Il suo attributo "*matricola*" deve essere **non nullo** e **unico** in quanto chiave primaria
- L'attributo "*data di nascita*" deve avere un valore tale che l'età dell'operatore sia compresa tra 18 e 70 anni

Richiesta
- Il suo attributo "*stringa di convalida*" deve essere **non nullo** e **unico** in quanto chiave primaria
- L'attributo "*stato*" è settato **non nullo** e di default "in attesa",
  - una missione può essere creata (relativa a quella richiesta) solo se la richiesta è in stato di "convalidata"
  - una richiesta non convalidata andrà in stato "ignorata"
  - quando viene creata una missione relativa ad una richiesta questa va in stato "in corso"
  - una richiesta su cui è stata creata una missione che poi viene cancellata dal database va in stato di "annullata"
  - infine una richiesta va in stato "terminata" quando viene aggiunta una conclusione alla missione che era stata generata da tale richiesta
- L'attributo "*email segnalante*" è reso **non nullo** in quanto fondamentale per la convalida delle richiesta di soccorso
- Gli attributi "*indirizzo*" e "*coordinate*" sono stati resi **non nulli** dato che sono indispensabili per gestire la richiesta di soccorso

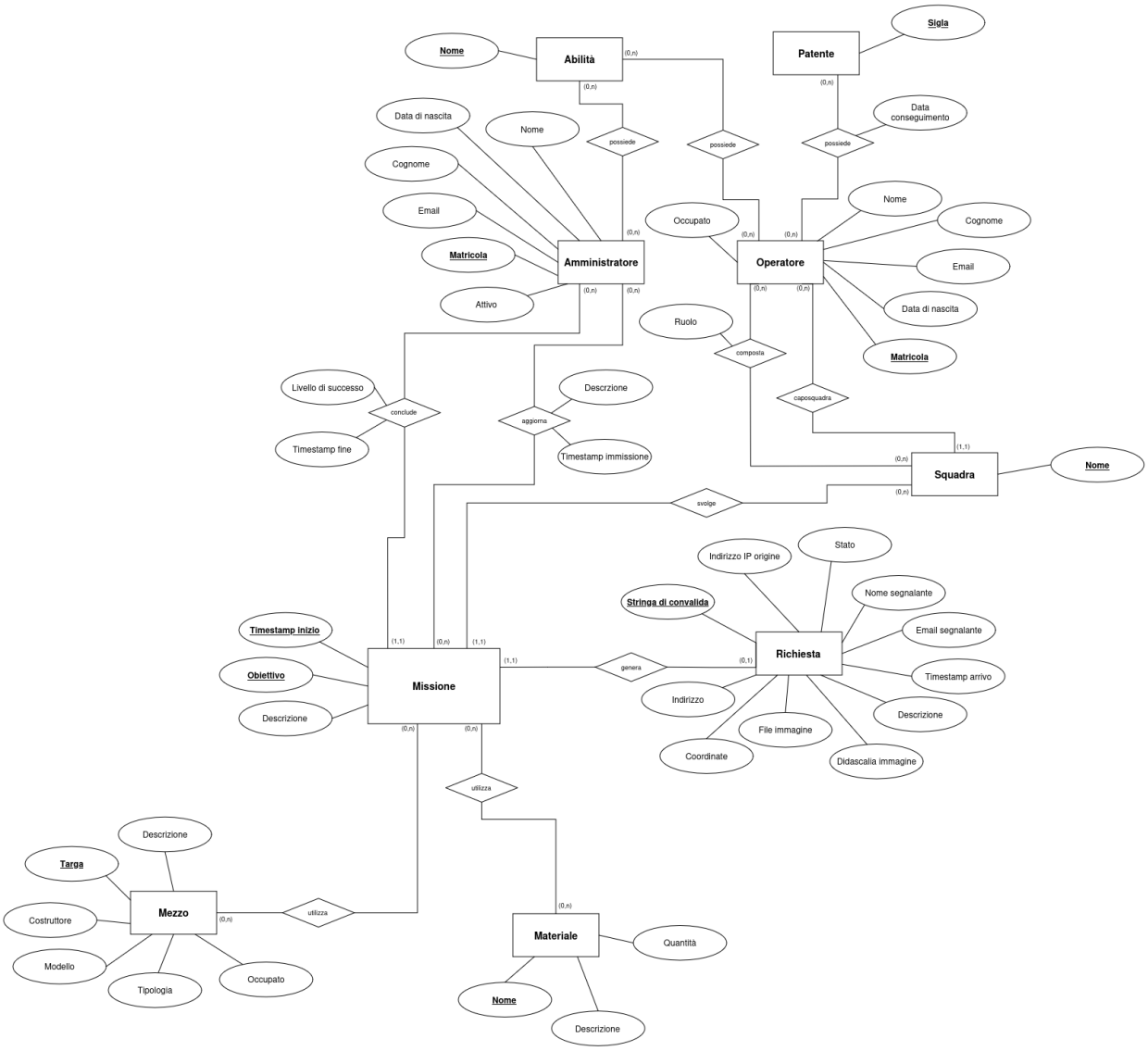
Missione
- Gli attributi "*data e ora inizio*" e "*obiettivo*" sono resi **unici** e **non nulli** in quanto insieme formano la chiave primaria

Squadra
- L'attributo "*nome*" è reso **unico** e **non nullo** in quanto chiave primaria

Mezzo
- L'attributo "*targa*" è reso **unico** e **non nullo** in quanto chiave primaria dell'entità
-

Materiale
- L'attributo "*nome*" è reso **unico** e **non nullo** in quanto chiave primaria

4 – Ristruttirazione del modello ER

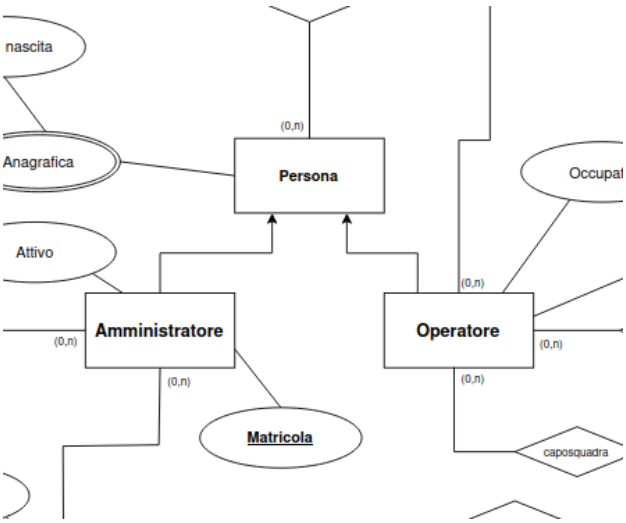


Come nel caso della versione non ristrutturata, una copia con qualità migliore è disponibile alla seguente location: ./soccorso/documentazione/modelli/ER_soccorso_ristrutturato.png.

4.1 – Dettagli ristrutturazione

Riposrtiamo qui le modifiche apportate dal diagramma ER non ristrutturato.

1) <u>Eliminazione gerarchia Persona – Operatore – Amministratore</u>



Discutiamo inizialmente la soluzione adottata per rimuovere la gerarchia tra *Persona* – *Operatore* – *Amministratore.*
Per rimuovere la gerarchia abbiamo optato per una **fusione Genitore – Figli**: abbiamo optato per questa soluzione dato che le entità *Operatore* e *Amministratore* avevano ruoli troppo diversi all'interno della base di dati per aggregate tutto dentro *Persona*.

Con questa soluzione possiamo affidare ad *Amministratore* e *Operatore* compiti diversi all'interno della base di dati, cosa che effettuando una fusione **Figli – Genitore** sarebbe stata impossibile. Difatti in questo modo un *Operatore* può

- Fare parte di una squadra
- Essere caposquadra di una certa squadra
- Partecipare a missioni

Mentre un *Amministratore* può

- Gestire richieste di soccorso
- Aggiornare missioni in corso
- Concludere missioni in corso

La fusione che abbiamo adottato (genitore – figli) comporta però una duplicazione degli attributi dell'entità *Persona*, quali "*Nome*", "*Cognome*", "*Data di nascita", "*Email*"* e "*Matricola",* e una duplicazione della relazione con l'entità *Abilità*.

2) Eliminazione attributi composti

Abbimo esculo dalla versione ristrutturata del nostro modello ER gli attributi composti, esplodendo i diversi campi dell'attributo e collegandoli all'entità corrispondente.
Gli attributi composti si trovavano:

- **Persona**: attributo "*anagrafica"*
- **Richiesta**
  - attributo "*informazioni segnalante*"
  - attributo "*posizione*"
- **Mezzo**: attributo "*caratteristiche*"

5 – Modello relazionale

Entità
- **Abilità**(ID, nome) [*nome Unique*]
- **Patente**(ID, sigla) [*sigla Unique*]
- **Amministratore**(ID, nome, cognome, data_di_nascita, email, matricola, attivo) [*matricola Unique*]
- **Operatore**(ID, nome, cognome, occupato, data_di_nascita, email, matricola) [*matricola Unique*]
- **Squadra**(ID, ID_operatore_caposquadra, nome) [*nome Unique*]
- **Richiesta**(ID, ID_immagine, stringa_convalida, indirizzo_IP_origine, stato, nome_segnalante, email_segnalante, timestamp_arrivo, descrizione, indirizzo, coordinate, file_immagine, didascalia_immagine) [*stringa_convalida Unique*]
- **Missione**(ID, ID_squadra, ID_richiesta, Timestamp_inizio, obiettivo, descrizione) [*timestamp_inizio* + *obiettivo Unique*]
- **Mezzo**(ID, targa, costruttore, modello, tipologia, descrizione, occupato) [*targa Unique*]
- **Materiale**(ID, nome, descrizione, quantità_totale) [*nome Unique*]

Relazioni
- **Amministratore_abilità**(ID_amministratore, ID_abilità)
- **Operatore_abilità**(ID_operatore, ID_abilità)
- **Operatore_patente**(ID_operatore, ID_patente)
- **Operatore_squadra**(ID_operatore, ID_squadra, ruolo)
- **Amministratore_aggiorna**(ID, ID_amministratore, ID_missione, messaggio_aggiornamento, timestamp_immissione)
- **Amministratore_conclude**(ID_amministratore, ID_missione, livello_successo, timestamp_fine_missione)
- **Missione_mezzo**(ID_missione, ID_mezzo)
- Missione_materiale(ID_missione, ID_materiale, quantita_missione)

6 – Progettazione fisica

6.1 – Implementazione del modello relazionale

Riportiamo tutto il codice SQL per la creazione della base di dati "Soccorso".
Tutto il codice riportato qui è disponibile nel file **./soccorso/sql/tables.sql**.

```sql
drop schema if exists soccorso;
create schema if not exists soccorso;

use soccorso;

drop table if exists aggiornamenti;
drop table if exists conclusioni;
drop table if exists missioneMezzo;
drop table if exists missioneMateriale;
drop table if exists missione;
drop table if exists squadraOperatore;
drop table if exists squadra;
drop table if exists abilitaAmministratore;
drop table if exists abilitaOperatore;
drop table if exists patenteOperatore;
drop table if exists operatore;
drop table if exists amministratore;
drop table if exists patente;
drop table if exists richiesta;
drop table if exists abilita;
drop table if exists mezzo;
drop table if exists materiale;
drop table if exists operatore;

drop user if exists 'admim'@'localhost';

create user 'admin'@'localhost' identified by 'pippo';
grant all privileges on soccorso.* to 'admin'@'localhost';

```

```sql
CREATE TABLE `operatore` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `nome` varchar(30) DEFAULT NULL,
  `cognome` varchar(30) DEFAULT NULL,
  `data_nascita` date NOT NULL,
  `occupato` tinyint(1) NOT NULL DEFAULT '0',
  `email` varchar(40) DEFAULT NULL,
  `matricola` int unsigned NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `matricola` (`matricola`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `amministratore` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `nome` varchar(30) DEFAULT NULL,
  `cognome` varchar(30) DEFAULT NULL,
  `data_nascita` date NOT NULL,
  `email` varchar(30) DEFAULT NULL,
  `matricola` int unsigned NOT NULL,
  `attivo` boolean NOT NULL DEFAULT FALSE,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `matricola` (`matricola`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `abilita` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `nome` varchar(30) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `nome` (`nome`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `patente` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `sigla` varchar(5) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `sigla` (`sigla`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `richiesta` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `stringa_convalida` varchar(20) NOT NULL,
  `indirizzo_ip_origine` varchar(15) NOT NULL,
  `stato` enum('in_attesa','convalidata','in_corso','terminata','annullata','ignorata')
       NOT NULL DEFAULT 'in_attesa',
  `nome_segnalante` varchar(20) NOT NULL,
  `email_segnalante` varchar(40) NOT NULL,
  `timestamp_arrivo` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `file_immagine` blob DEFAULT NULL,
  `didascalia_immagine` varchar(30) DEFAULT NULL,
  `descrizione` text DEFAULT NULL,
  `indirizzo` varchar(30) NOT NULL,
  `coordinate` varchar(20) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `stringa_convalida` (`stringa_convalida`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `squadra` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `nome` varchar(20) DEFAULT NULL,
  `ID_operatore_caposquadra` int unsigned NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `operatore_caposquadra` (`ID_operatore_caposquadra`),
  CONSTRAINT `operatore_caposquadra` FOREIGN KEY (`ID_operatore_caposquadra`)
       REFERENCES `operatore` (`ID`) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `missione` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `timestamp_inizio` datetime NOT NULL,
  `obiettivo` text NOT NULL,
  `descrizione` text,
  `ID_squadra` int unsigned NOT NULL,
  `ID_richiesta` int unsigned NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `richiesta_associata` (`ID_richiesta`),
  KEY `squadra_associata` (`ID_squadra`),
  UNIQUE KEY (`timestamp_inizio`, `ID_squadra`),
  CONSTRAINT `richiesta_associata` FOREIGN KEY (`ID_richiesta`)
    REFERENCES `richiesta` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `squadra_associata` FOREIGN KEY (`ID_squadra`)
    REFERENCES `squadra` (`ID`) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `mezzo` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `targa` varchar(7) NOT NULL,
  `costruttore` varchar(30) NOT NULL,
  `modello` varchar(30) NOT NULL,
  `tipologia` varchar(30) NOT NULL,
  `descrizione` text DEFAULT NULL,
  `occupato` tinyint(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `targa` (`targa`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `materiale` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `nome` varchar(20) NOT NULL,
  `descrizione` text DEFAULT NULL,
  `quantita_totale` int NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `nome` (`nome`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `abilitaAmministratore` (
  `ID_amministratore` int unsigned NOT NULL,
  `ID_abilita` int unsigned NOT NULL,
  PRIMARY KEY (`ID_amministratore`,`ID_abilita`),
  CONSTRAINT `abilita_amministratore` FOREIGN KEY (`ID_abilita`) REFERENCES `abilita` (`ID`)
  CONSTRAINT `amministratore_abilita` FOREIGN KEY (`ID_amministratore`) REFERENCES `amminis
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `abilitaOperatore` (
  `ID_abilita` int unsigned NOT NULL,
  `ID_operatore` int unsigned NOT NULL,
  PRIMARY KEY (`ID_abilita`,`ID_operatore`),
  CONSTRAINT `abilita_operatore` FOREIGN KEY (`ID_abilita`) REFERENCES `abilita` (`ID`) ON
  CONSTRAINT `operatore_abilita` FOREIGN KEY (`ID_operatore`) REFERENCES `operatore` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `patenteOperatore` (
  `ID_operatore` int unsigned NOT NULL,
  `ID_patente` int unsigned NOT NULL,
  `data_conseguimento` date NOT NULL,
  PRIMARY KEY (`ID_operatore`,`ID_patente`),
  CONSTRAINT `operatore_patente` FOREIGN KEY (`ID_operatore`) REFERENCES `operatore` (`ID`)
  CONSTRAINT `patente_operatore` FOREIGN KEY (`ID_patente`) REFERENCES `patente` (`ID`) ON
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `squadraOperatore` (
  `ID_operatore` int unsigned NOT NULL,
  `ID_squadra` int unsigned NOT NULL,
  `ruolo` varchar(20) NOT NULL,
  PRIMARY KEY (`ID_operatore`, `ID_squadra`),
  CONSTRAINT `operatore_squadra` FOREIGN KEY (`ID_operatore`)
    REFERENCES `operatore` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `squadra_operatore` FOREIGN KEY (`ID_squadra`)
    REFERENCES `squadra` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `aggiornamenti` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `ID_amministratore` int unsigned NOT NULL,
  `ID_missione` int unsigned NOT NULL,
  `messaggio_aggiornamento` text NOT NULL,
  `timestamp_immissione` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`ID`),
  KEY `amministratore_missione_agg` (`ID_amministratore`),
  KEY `missione_amministratore_agg` (`ID_missione`),
  CONSTRAINT `amministratore_missione_agg` FOREIGN KEY (`ID_amministratore`)
    REFERENCES `amministratore` (`ID`) ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT `missione_amministratore_agg` FOREIGN KEY (`ID_missione`)
    REFERENCES `missione` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `conclusioni` (
  `ID_missione` int unsigned NOT NULL,
  `ID_amministratore` int unsigned NOT NULL,
  `livello_successo` smallint NOT NULL,
  `timestamp_fine` datetime NOT NULL,
  PRIMARY KEY (`ID_missione`,`ID_amministratore`),
  CONSTRAINT `amministratore_missione_conc` FOREIGN KEY (`ID_amministratore`)
    REFERENCES `amministratore` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `missione_amministratore_conc` FOREIGN KEY (`ID_missione`)
    REFERENCES `missione` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `check_livello_successo` CHECK ((`livello_successo` between 0 and 5))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `missioneMezzo` (
  `ID_missione` int unsigned NOT NULL,
  `ID_mezzo` int unsigned NOT NULL,
  PRIMARY KEY (`ID_missione`,`ID_mezzo`),
  CONSTRAINT `mezzo_missione` FOREIGN KEY (`ID_mezzo`)
    REFERENCES `mezzo` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `missione_mezzo` FOREIGN KEY (`ID_missione`)
    REFERENCES `missione` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
CREATE TABLE `missioneMateriale` (
  `ID_missione` int unsigned NOT NULL,
  `ID_materiale` int unsigned NOT NULL,
  `quantita` int unsigned NOT NULL DEFAULT '1',
  PRIMARY KEY (`ID_missione`,`ID_materiale`),
  CONSTRAINT `materiale_missione` FOREIGN KEY (`ID_materiale`)
    REFERENCES `materiale` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `missione_materiale` FOREIGN KEY (`ID_missione`)
    REFERENCES `missione` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

6.2 – Implementazione dei vincoli

Oltre ai vincoli di chiave esterna e qualche vincolo check all'interno della tabelle, sono stati effettivamente aggiunti svariati vincoli per quanto riguarda l'integrità dei dati sotto forma di trigger. Data l'elevata lunghezza del file (oltre 800 righe) risulta impossibile riportare qui il codice relativo a questi vincoli. Di conseguenza per visionare il codice sarà opportuno recarsi al file **./soccorso/sql/triggers.sql** dove sono disponibili tutti i trigger realizzati.

6.3 – Implementazione funzionalità richieste

In questa sezione verranno riportate le implementazioni delle funzionalità richieste dalla spacifica (1.1.1) sotto forma di codice SQL. Si noti che moltre di queste operazioni, per funzilnare correttamente, lavorano in

sincronia con dei trigger specifici (principalmente le operazioni che richiedono l'inserimento di dati nella tabelle).

Tutte le operazioni sono state realizzate tramite **procedure, funzioni oppure viste** all'interno della base di dati. In caso si vogliano visionare il codice delle operazioni recarsi a **./soccorso/sql/operations/...**

Operazione 1 – `Inserimento di una richietsa di sosscorso`

```
-- per l'inserimento della richiesta abbiamo creato 2 funzioni differenti: la prima che inserisce la richiesta con il
--timestamp di default (che è quello attuale), e la seconda che invece prende in input un dato timestamp aggiuntivo, in modo da poter aggiungere se necessario richieste non arrivate sul momento.
CREATE DEFINER=`root`@`localhost` FUNCTION `aggiungi_richiesta_timestamp_corrente`(stringa_convalida varchar(20),
    indirizzo_ip_origine varchar(12), nome_segnalante varchar(30),
email_segnalante varchar(40), descrizione text, indirizzo varchar(30), coordinate varchar(20)) RETURNS int
    DETERMINISTIC
        begin
            declare ID_toReturn int unsigned;

            insert into richiesta(stringa_convalida, indirizzo_ip_origine, nome_segnalante, email_segnalante, descrizione, indirizzo, coordinate)
            values (stringa_convalida, indirizzo_ip_origine, nome_segnalante, email_segnalante, descrizione, indirizzo, coordinate);

            set ID_toReturn = last_insert_id();
            return ID_toReturn;
        end$

CREATE DEFINER=`root`@`localhost` FUNCTION `aggiungi_richiesta_timestamp_personalizzato`(stringa_convalida varchar(20),
    indirizzo_ip_origine varchar(12), nome_segnalante varchar(30),
email_segnalante varchar(40), timestamp_arrivo timestamp, descrizione text, indirizzo varchar(30), coordinate varchar(20)) RETURNS int
    DETERMINISTIC
        begin
            declare ID_toReturn int unsigned;

            insert into richiesta(stringa_convalida, indirizzo_ip_origine, nome_segnalante, email_segnalante, timestamp_arrivo, descrizione, indirizzo, coordinate)
            values (stringa_convalida, indirizzo_ip_origine, nome_segnalante, email_segnalante, timestamp_arrivo, descrizione, indirizzo, coordinate);

            set ID_toReturn = last_insert_id();
            return ID_toReturn;
        end$
```

La seguente operazione (sviluppata in due varianti) permette l'inserimento di una richiesta di soccorso all'interno dela base di dati tramite **funzione** la quale restituisce alla fine l'ID della richiesta appena aggiunta. Entrambe le funzioni sono visionabili qui: **./soccorso/sql/operations/insertOperations.sql**.

Operazione 2 – Creazione di una missione connessa a una richiesta di soccorso attiva

```
CREATE DEFINER=`root`@`localhost` FUNCTION `aggiungi_missione`(timestamp_inizio datetime, obiettivo text,
    descrizione text, ID_squadra int, ID_richiesta int) RETURNS int
    DETERMINISTIC
        begin
            declare ID_toReturn int unsigned;

            INSERT INTO missione (timestamp_inizio, obiettivo, descrizione, ID_squadra, ID_richiesta)
            VALUES (timestamp_inizio, obiettivo, descrizione, ID_squadra, ID_richiesta);

            set ID_toReturn = last_insert_id();
            return ID_toReturn;
        end$
```

Questa operazione permette l'inserimento di una missione nella base di dati.
La procedura funziona anche grazie all'aiuto dei trigger, in particolar modo un trigger he verifica che la richiesta associata alla missione sia in stato "*convalidata*" prima di aggiungere la missione. In caso contrario viene segnalato un errore in inserimento.
La funzione è disponibile qui: **./soccorso/sql/operations/insertOperations.sql**.

Operazione 3 – Chiusura di una missione

```
CREATE DEFINER=`root`@`localhost` FUNCTION `aggiungi_conclusione`(id_missione int unsigned, id_amministratore int unsigned,
    livello_successo smallint, timestamp_fine datetime) RETURNS int
    DETERMINISTIC
        begin
            declare ID_toReturn int unsigned;

            INSERT INTO conclusioni (ID_missione, ID_amministratore, livello_successo, timestamp_fine)
            VALUES (id_mission, id_amministratore, livello_successo, timestamp_fine);

            set ID_toReturn = last_insert_id();
            return ID_toReturn;
        end$
```

Per chiusere una missione, tutto ciò che dobbiamo fare è aggiungere una conclusione, tutto il resto viene gestito dai trigger, in particolare, successivamente all'aggiunta di una riga nella tabella conclusioni, i trigger andranno a

- settare lo stato della richiesta associata a "*terminata*"
- liberare tutti gli operatori della squadra associata (settare a "*FALSE*" il loro campo "*occupato"*)
- restituire tutti i materiali in utilizzo dalla missione sulla tabella "*missioneMateriale*" aggiornando la loro quantità totale
- liberare tutti i mezzi utilizzati per quella missione

La procedura è disponibile qui: **./soccorso/sql/Operations/insertOperations.sql**.

Operazione 4 – Estrazione degli operatori non impegnati in missioni in corso

```
-- Operatori non occupati
CREATE VIEW select_operatori_non_occupati AS
SELECT *
FROM operatore o
WHERE o.occupato = FALSE;
```

Questa operazione è stata implementata tramite vista: dato che il campo "*occupato*" in Operatore è **dinamico**, per estrarre gli operatori non occuparti in missioni in corso ci basterà estrarre gli operatori con occupato = FALSE.
La seguente vista è disponibile qui: **./soccorso/sql/views.sql**.

Operazione 5 – Calcolo del numero di missioni svolte da un operatore

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `conteggio_missioni_terminate_operatore`(in ID_operatore int)
    begin
        select count(distinct m.ID) as numero_missioni_terminate
        from squadraOperatore so
        join missione m on so.ID_squadra = m.ID_squadra
        join richiesta r on m.ID_richiesta = r.ID
        where so.ID_operatore = ID_operatore
        and r.stato = 'terminata';
    end$
```

Questa procedura calcola le missioni **terminate** di un certoi operatore, dato in input un id relatico ad un operatore. Questo numero viene calcolato tramite la funzione **COUNT( )**, che restituisce il numero di ID **distinti** di missioni terminate (JOIN con richiesta e verifica dello stato).
Una seconda variante potrebbe essere realizzata facendo JOIN con la teballa conclusioni, così da prendere solo missioni terminate.
Questa procedura è disponibile qui: **./soccorso/sql/operations/projectOperations.sql**.

Operazione 6 - Calcolo del tempo medio di svolgimento delle missioni (dalla creazione alla chiusura) in un anno specifico

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `tempo_medio_missione_anno`(in anno int)
    begin
        select avg (timestampdiff(second, m.timestamp_inizio, c.timestamp_fine)) / 3600 as tempo_medio_missione_in_ore
        from missione m
        join conclusioni c on m.ID = c.ID_missione
        where year (c.timestamp_fine) = anno;
    end$
```

La procedura calcola il tempo medio della durata delle missioni tramite la funzione **AVG( )** e usa la funzione **TIMESTAMPDIFF( )** per calcolare la differenza (in secondi) tra il timestamp di fine missione (in conclusioni) e il timestamp di inizio missione (in missione), per poi dividre per 3600, così da ottenere il risultato in ore.
Questa procedura è disponibile qui: **./soccorso/sql/operations/projectOperations.sql**.

Opedrazione 7 - Calcolo del numero di richieste provenienti da un certo soggetto segnalante (identificato dall'indirizzo email) o da un certo indirizzo IP nelle ultime 36 ore

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `calcolo_numero_richieste_email_segnalante`(in email_segnalante varchar(40))
    begin
        SELECT count(distinct r.ID)
        FROM richiesta r
        WHERE r.email_segnalante = email_segnalante
            AND r.timestamp_arrivo >= NOW() - interval 36 HOUR;
    end$

CREATE DEFINER=`root`@`localhost` PROCEDURE `calcolo_numero_richieste_indirizzo_ip`(in indirizzo_ip varchar(15))
    begin
        SELECT count(distinct r.ID)
        FROM richiesta r
        WHERE r.indirizzo_ip_segnalante = indirizzo_ip
            AND r.timestamp_arrivo >= NOW() - interval 36 HOUR;
    end$
```

Entrambe le varianti di questa operazioni sono state implementate tramite procedure.

Queste procedure sono disponibili qui: **./soccorso/sql/operations/projectOperations.sql**.

Operazione 8 - Calcolo del tempo totale di impiego in missione di un certo operatore

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `calcolo_tempo_totale_operatore`(in id_operatore int)
    begin
        SELECT o.nome, o.cognome, o.matricola, SUM(timestampdiff(SECOND, m.timestamp_inizio, c.timestamp_fine)) / 3600 AS tempo_totale_ore
        FROM missione m
        JOIN conclusione c ON m.ID = c.ID_missione
        JOIN squadraOperatore so ON so.ID_squadra = m.ID_squadra
        JOIN operatore o ON so.ID_operatore = o.ID
        WHERE so.ID_operatore = id_operatore
        GROUP BY o.ID;
    end$
```

Questa procedura calcola il tempo totale di impiego in missione di un certo operatore identificato univocamente tramite ID. La differenza tra timestasmp è calcolata tramite **TIMESTAMPDIFF( )** che vengono poi sommati tramite la funzione **SUM( )**.
Questa procedura è disponibile qui: **./soccorso/sql/operations/projectOperations.sql**.

Operazione 9 - Estrazione delle missioni svoltesi negli ultimi tre anni nello stesso luogo di una missione data.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `missioni_stesso_luogo_last3years`(in id_missione int)
    begin
        SELECT m1.*
        FROM missione m1
        JOIN richiesta r1 ON r1.ID = m1.ID_richiesta
        JOIN missione m2 ON m2.ID = id_missione
        JOIN richiesta r2 ON r2.ID = m2.ID_richiesta
        WHERE r1.indirizzo = r2.indirizzo
            AND r1.coordinate = r2.coordinate
            AND m1.timestamp_inizio >= NOW() - INTERVAL 3 YEAR
            AND m1.ID != m2.ID; -- serve as escluderela missione passata in input
    end$
```

Questa procedura tramite dei JOIN prende le missioni che hannmo stesso luogo di una missione passate in input (tramite ID) per poi filtrare negli ultimi 3 anni tramite il timestamp. L'ultima riga serve a non racchiudere la missione passata in input.
Questa procedura è disponibile qui: **./soccorso/sql/operations/projectOperations.sql**.

Operazione 10 - Estrazione della lista delle richieste di soccorso chiuse con risultato non totalmente positivo (livello di successo minore di 5)

```
-- Missioni con esito non totalmente positivo
CREATE VIEW select_missioni_con_esito_non_totalmente_positivo AS
SELECT m.ID, m.ID_squadra, m.timestamp_inizio, m.obiettivo, c.livello_successo, c.timestamp_fine
FROM conclusioni c
JOIN missione m ON c.ID_missione = m.ID
WHERE c.livello_successo < 5;
```

Qusta operazione è stata implementata tramite vista, dato che non vi era nessun parametro.
Questa vista è disponibile qui: **./soccorso/sql/operations/views.sql**.

Operazione 11 - Estrazione degli operatori maggiormente coinvolti nelle richieste di soccorso chiuse con risultato non totalmente positivo

```sql
-- Operatori maggiormente coinvolti nelle richieste di soccorso chiuse con risultato non totalmente positivo
CREATE VIEW operatori_missioni_con_esito_non_totalmente_positivo AS
SELECT o.matricola, o.nome, o.cognome, COUNT(distinct smp.ID) AS conteggio_missioni
FROM select_missioni_con_esito_non_totalmente_positivo smp
JOIN squadra s ON smp.ID_squadra = s.ID
JOIN squadraOperatore so ON s.ID = so.ID_squadra
JOIN operatore o ON so.ID_operatore = o.ID
GROUP BY o.matricola
ORDER BY conteggio_missioni DESC;
```

Questa operazione è stata realizzata tramite vista dato che non vi è nessun parametro.
Si noti che questa vista utilizza un'altra vista come sotto-query per prendere le missioni cocn esito non totalmente positivo.
Questa vista è disponibile qui: **./soccorso/sql/operations/views.sql**.

Operazione 12 - Estrazione dello storico delle missioni in cui è stato coinvolto un certo mezzo

```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `storico_missioni_mezzo`(in id_mezzo int unsigned)
    begin
        select mz.targa, mz.costruttore, mz.modello, m.ID as ID_missione, m.timestamp_inizio, m.obiettivo,
               m.descrizione, r.coordinate, r.indirizzo, r.stato as stato_richiesta
        from mezzo mz
        join missioneMezzo mm on mz.ID = mm.ID_mezzo
        join missione m on mm.ID_missione = m.ID
        join richiesta r on m.ID_richiesta = r.ID
        where mz.ID = id_mezzo;
    end$
```

Questa procedura seleziona lo storico delle missioni (terminate) in cui è stato utilizzato un certo mezzo, passato in input tramite ID.
Questa procedura è disponibile qui: **./soccorso/sql/operations/projectOperations.sql**.

Operazione 13 - Calcolo delle ore d'uso di un certo materiale

```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `calcolo_tempo_uso_materiale`(in id_materiale int)
    begin
        SELECT ma.nome as nome_materiale, SUM(timestampdiff(SECOND, m.timestamp_inizio, c.timestamp_fine)) / 3600 AS tempo_uso_materiale
        FROM missioneMateriale mm
        JOIN missione m ON mm.ID_missione = m.ID
        JOIN conclusione c ON m.ID = c.ID_missione
        JOIN materiale ma on mm.ID_materiale = ma.ID
        WHERE mm.ID_materiale = id_materiale;
        GROUP BY ma.ID;
    end$
```

Questa procedura calcola il tempo di utilizzo di uno specifico materiale, passato in input tramite ID. La somma viene effettuata con la funzione **SUM( )** e i tempi da sommare vengono calcolati tramite **TIMESTAMPDIFF( )**.
Questa procedura è disponibile qui: **./soccorso/sql/operations/projectOperations.sql**.

Oltre alle operazioni qui riportate, nei file **selectOperations.sql** e **insertOperations.sql** sono presenti procedure e funzioni che arricchiscono la base di dati, quali funzioni di selezione e funzioni di inserimento.