Schema Logico

Leggenda: Attributo; Chiave, **Chiave Esterna**, *null

Account (Saldo Punti, Codice Fedeltà, Livello, Account Password, **E-mail**)

```
CREATE TABLE Account (
    Saldo_punti INTEGER NOT NULL DEFAULT 0,
    Codice_fedeltà INTEGER NOT NULL PRIMARY KEY,
    Livello          LIVELLI_ACCOUNT DEFAULT 'Standard' NOT NULL,
    Account_password VARCHAR(30) NOT NULL,
    Email            EMAIL  NOT NULL,
    CONSTRAINT fk_account_persona
        REFERENCES persona
);
```

Assegnazione Deposito (**Deposito, Operaio**, Inizio, *Fine)

```
CREATE TABLE Assegnazione_Deposito (
  Deposito VARCHAR(30) NOT NULL,
  Operaio CODICE_FISCALE NOT NULL,
  Inizio TIMESTAMP NOT NULL,
  Fine TIMESTAMP,
  CONSTRAINT fk_luogodilavoro_nome
    FOREIGN KEY(deposito)
      REFERENCES deposito(nome)
      ON UPDATE CASCADE,
  CONSTRAINT fk_luogodilavoro_cf
    FOREIGN KEY(operaio)
      REFERENCES operaio(impiegato)
      ON UPDATE CASCADE,
  PRIMARY KEY (deposito, operaio, inizio)
  CONSTRAINT assegnazione_deposito_check
      CHECK (fine > inizio)

);
```

Assegnazione Ufficio (**Ufficio Segretario**, Inizio *Fine,)

```
CREATE TABLE Assegnazione_Ufficio (
    Segretario codice_fiscale NOT NULL
        CONSTRAINT fk_locazione_cf
            REFERENCES segretario
            ON UPDATE CASCADE,
    Fine       DATE,
    Inizio     DATE NOT NULL,
    Ufficio    INTEGER NOT NULL
    CONSTRAINT fk_assegnazione_ufficio_ufficio
        REFERENCES ufficio,
    PRIMARY KEY (segretario, ufficio, inizio)
    CONSTRAINT assegnazione_ufficio_check
        CHECK (fine > inizio)
);
```

Assistente (**Impiegato**)

```
CREATE TABLE Assistente (
    Impiegato codice_fiscale PRIMARY KEY
        CONSTRAINT fk_assistente_impiegato
            REFERENCES impiegato
);
```

Assistenza Clienti (ID, E-Mail, Oggetto, **Lingua, Account*, Biglietto*,** Oggetto, Chiusa)

```
CREATE TABLE Assistenza_Clienti (
    Identificativo SERIAL PRIMARY KEY NOT NULL
```

```
    Email           email NOT NULL,
    Lingua          VARCHAR(20) NOT NULL
        CONSTRAINT fk_assistenza_lingua
            REFERENCES lingua
            ON UPDATE CASCADE,
    Account         INTEGER
        CONSTRAINT fk_account
            REFERENCES account,
    Biglietto       INTEGER
        CONSTRAINT fk_assistenza_biglietto
            REFERENCES biglietto,
    Oggetto         VARCHAR(100) NOT NULL,
    Chiusa          BOOLEAN DEFAULT FALSE NOT NULL
);
```

Banco Manutenzione (Identificativo, **Deposito**)

```
CREATE TABLE Banco_Manutenzione (
    Identificativo SERIAL PRIMARY KEY,
    Deposito VARCHAR(30) NOT NULL,
    CONSTRAINT fk_banco_nome
         REFERENCES deposito
         ON UPDATE CASCADE
);
```

Biglietto (ID, Prezzo, Rimborsato, Data_Acquisto**,** MetodoPagamento, Id_Pagamento, ***Account, Spostamento_Percorso, Posto, Carrozza, Treno, E-mail, Data_Viaggio, Partenza_Ordine, Arrivo_Ordine**)

```
CREATE TABLE Biglietto (
    Identificativo SERIAL DEFAULT NOT NULL PRIMARY KEY,
    Prezzo NUMERIC(6, 2) DEFAULT 0.00 NOT NULL,
    Rimborsato BOOLEAN DEFAULT FALSE NOT NULL,
    Metodopagamento metodipagamento NOT NULL,
    Id_pagamento VARCHAR(100) NOT NULL,
    Account INTEGER
        CONSTRAINT fk_biglietto_account
            REFERENCES account,
    Spostamento_percorso INTEGER NOT NULL,
    Posto SMALLINT NOT NULL,
    Carrozza SMALLINT NOT NULL,
    Email email NOT NULL
        CONSTRAINT fk_biglietto_persona
            REFERENCES persona,
    Treno SMALLINT,
    Data_acquisto TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    Data_viaggio DATE NOT NULL,
    Partenza_ordine INTEGER NOT NULL,
    Arrivo_ordine INTEGER NOT NULL,
    CONSTRAINT fk_biglietto_posto
        FOREIGN KEY (carrozza, treno, posto) REFERENCES posto (carrozza,
idtreno, numero),
    CONSTRAINT fk_biglietto_spostamento
        FOREIGN KEY (spostamento_percorso, data_viaggio) REFERENCES spostamento,
    CONSTRAINT fk_biglietto_sezione_arrivo
        FOREIGN KEY (arrivo_ordine, spostamento_percorso) REFERENCES sezione
(ordine, percorso),
    CONSTRAINT fk_biglietto_sezione_partenza
        FOREIGN KEY (spostamento_percorso, partenza_ordine) REFERENCES sezione
);
```

Binario (**Stazione**, Numero)

```
CREATE TABLE Binario (
    Stazione VARCHAR(100) NOT NULL
```

```
        CONSTRAINT binario_stazione_nome_fk
            REFERENCES stazione,
    Numero    VARCHAR(10)   NOT NULL,
    CONSTRAINT binario_pk
        PRIMARY KEY (stazione, numero)
);
```

Carrozza (Numero, IDTreno, **ClasseViaggio**)

```
CREATE TABLE Carrozza (
    Numero  SMALLINT NOT NULL,
    Idtreno SMALLINT NOT NULL
        CONSTRAINT fk_carrozza_idtreno
            REFERENCES treno
            ON UPDATE CASCADE,
    Classe_viaggio VARCHAR(30) NOT NULL
        CONSTRAINT fk_carrozza_classeviaggio
            REFERENCES classe_viaggio
            ON UPDATE CASCADE,
    PRIMARY KEY (numero, idtreno)
);
```

Cellulare (numero_telefonico, **E-mail**)

```
CREATE TABLE Cellulare (
    Email             email       NOT NULL
        CONSTRAINT fk_cellulare_email
            REFERENCES persona
            ON UPDATE CASCADE,
    Numero_Telefonico VARCHAR(10) NOT NULL,
    PRIMARY KEY (email, numero_telefonico)
);
```

Classe Viaggio (Nome, prezzo)

```
CREATE TABLE Classe_Viaggio (
    Nome            VARCHAR(30) NOT NULL,
    Prezzo          NUMERIC(6, 2) NOT NULL,
    PRIMARY KEY (nome)
);
```

Contenuto Assistenza (**Assistenza**, Timestamp Richiesta, Testo, *Testo risposta, *Timestamp risposta, ***Segretario**)

```
CREATE TABLE Contenuto_Assistenza (
    Assistenza  INTEGER NOT NULL
        CONSTRAINT fk_contenuto_assistenza_assistenza
            REFERENCES assistenza_clienti,
    Testo               TEXT      NOT NULL,
    Timestamp_richiesta TIMESTAMP NOT NULL,
    Testo_risposta      TEXT,
    Timestamp_risposta  TIMESTAMP,
    Segretario          codice_fiscale
        CONSTRAINT fk_contenuto_assistenza_segretario
            REFERENCES segretario,
    CONSTRAINT contenuto_assistenza_key
        PRIMARY KEY (assistenza, timestamp_richiesta)
);
```

Contratto (ID, Tipo, Inizio, Rescisso, Fine, Paga_Oraria, **Impiegato**)

```
CREATE TABLE Contratto (
    Identificativo SERIAL
        PRIMARY KEY,
    Tipo           tipo_contratto                NOT NULL,
```

```
    Inizio          DATE    DEFAULT (NOW())::DATE NOT NULL,
    Rescisso        BOOLEAN DEFAULT FALSE         NOT NULL,
    Paga_oraria     NUMERIC(5, 2)                 NOT NULL,
    Fine            DATE                          NOT NULL,
    Impiegato       codice_fiscale                NOT NULL
        CONSTRAINT contratto_impiegato_cf_fk
            REFERENCES impiegato,
    CONSTRAINT contratto_check
        CHECK (fine > inizio)
);
```

Deposito (**Nome**, Via, Civico, Città, Cap)

```
CREATE TABLE Deposito (
  Nome VARCHAR(30) PRIMARY KEY,
  Via VARCHAR(30) NOT NULL,
  Città VARCHAR(30) NOT NULL,
  Civico INTEGER NOT NULL,
  Cap INTEGER NOT NULL,
  CONSTRAINT fk_deposito_nome
    FOREIGN KEY(Nome)
      REFERENCES PuntoDiSosta(Nome)
      ON UPDATE CASCADE
);
```

Direzione Ufficio (**Ufficio, Segretario**, Inizio, *Fine)

```
CREATE TABLE Direzione_Ufficio (
    Ufficio    INTEGER        NOT NULL
        CONSTRAINT fk_direzione_ufficio_ufficio
            REFERENCES ufficio,
    Segretario codice_fiscale NOT NULL
        CONSTRAINT fk_direzione_ufficio_segretario
            REFERENCES segretario,
    Inizio     TIMESTAMP      NOT NULL,
    Fine       TIMESTAMP,
    CONSTRAINT direzione_ufficio_pk
        PRIMARY KEY (ufficio, segretario, inizio)
    CONSTRAINT direzione_ufficio_check
        CHECK (fine > inizio)
);
```

Ferroviere (**Impiegato**, Ruolo)

```
CREATE TABLE Ferroviere (
  Impiegato CODICE_FISCALE PRIMARY KEY,
  Ruolo RUOLO_FERROVIERE NOT NULL,
  CONSTRAINT fk_ferroviere_cf
    FOREIGN KEY(Impiegato)
      REFERENCES Impiegato(CF)
      ON UPDATE CASCADE
);
```

Impiegato (CF, N**ome, Cognome, Email,** Email Aziendale, Data di Nascita, Sesso, Iban, Tipologia)

```
CREATE TABLE Impiegato (
    CF CODICE_FISCALE  NOT NULL PRIMARY KEY,
    Nome            VARCHAR(100) NOT NULl,
    Cognome         VARCHAR(100) NOT NULL,
    Sesso           VARCHAR(100) NOT NULL,
    Email           EMAIL NOT NULL,
    Email_aziendale EMAIL  NOT NULL,
```

```
    Data_nascita    DATE NOT NULL,
    Iban            IBAN NOT NULL,
    Tipologia       tipo_impiegato NOT NULL
    CONSTRAINT fk_impiegato_persona
        FOREIGN KEY (Nome, Cognome, Email) REFERENCES Persona
);
```

Lingue Assistente (**Ferroviere, Lingua**)

```
CREATE TABLE Lingue_Ferroviere (
  Assistente CODICE_FISCALE,
  Lingua VARCHAR(20),
  CONSTRAINT fk_comunicare_cf
    FOREIGN KEY(Assistente)
      REFERENCES Assistente(Impiegato)
      ON UPDATE CASCADE,
  CONSTRAINT fk_comunicare_lingua
    FOREIGN KEY(Lingua)
      REFERENCES Lingua(Lingua)
      ON UPDATE CASCADE,
  PRIMARY KEY (Ferroviere, Lingua)
);
```

Lingua (**Lingua**)

```
CREATE TABLE Lingua (
  Lingua VARCHAR(20) NOT NULL PRIMARY KEY
);
```

Lingue Segretario (**Segretario, Lingua**)

```
CREATE TABLE Lingue_Segretario (
  Segretario CODICE_FISCALE,
  Lingua VARCHAR(20),
  CONSTRAINT fk_scrivere_cf
    FOREIGN KEY(Segretario)
      REFERENCES Segretario(impiegato)
      ON UPDATE CASCADE,
  CONSTRAINT fk_scrivere_lingua
    FOREIGN KEY(Lingua)
      REFERENCES Lingua(Lingua)
      ON UPDATE CASCADE,
  PRIMARY KEY (Segretario, Lingua)
);
```

Manutenzione (ID, Descrizione, *Fine, Inizio, **IDtreno, Banco, ***Descrizione Guasto)

```
CREATE TABLE Manutenzione (
    Identificativo SERIAL NOT NULL PRIMARY KEY,
    Descrizione     TEXT NOT NULL,
    Idtreno         INTEGER NOT NULL,
    Fine            TIMESTAMP,
    Inizio          TIMESTAMP NOT NULL,
    Banco           INTEGER NOT NULL,
    Descrizione_guasto    TEXT,
    CONSTRAINT fk_manutenzione_idtreno
        FOREIGN KEY(idtreno)
            REFERENCES treno
            ON UPDATE CASCADE,
    CONSTRAINT fk_manutenzione_banco
        FOREIGN KEY(banco)
            REFERENCES banco_riparazioni
            ON UPDATE CASCADE
    CONSTRAINT manutenzione_check
        CHECK (fine > inizio));
```

UniTrain: Definizione base di dati in SQL

Modello Treno (Nome, Velocità Max)

```
CREATE TABLE Modello_Treno (
  Nome VARCHAR(30) NOT NULL PRIMARY KEY,
  Velocita_Max SMALLINT NOT NULL
);
```

Operaio (**Impiegato**)

```
CREATE TABLE Operaio (
  Impiegato CODICE_FISCALE PRIMARY KEY,
  CONSTRAINT fk_operaio_cf
    FOREIGN KEY(Impiegato)
      REFERENCES Impiegato(CF)
      ON UPDATE CASCADE
);
```

Percorso (Identificativo, Ora Arrivo, Ora Partenza, **Partenza, Arrivo**, Tipologia)

```
CREATE TABLE Percorso (
    Identificativo SERIAL NOT NULL PRIMARY KEY,
    Ora_arrivo TIME NOT NULL,
    Ora_partenza TIME NOT NULL,
    Partenza VARCHAR(100) NOT NULL,
    Arrivo VARCHAR(100) NOT NULL,
    Tipologia      tipo_spostamento DEFAULT 'Viaggio'::tipo_spostamento NOT NULL
    CONSTRAINT fk_percorso_partenza
        FOREIGN KEY (partenza)
            REFERENCES punto_interesse,
    CONSTRAINT fk_percorso_arrivo
        FOREIGN KEY (arrivo)
            REFERENCES punto_interesse
    CONSTRAINT percorso_check
        CHECK (ora_arrivo > ora_partenza)
)
```

Persona (Nome, Cognome, E-mail, *Cellulare)

```
CREATE TABLE Persona (
    Nome      VARCHAR(100) NOT NULL,
    Cognome   VARCHAR(100) NOT NULL,
    Email     email PRIMARY KEY NOT NULL,
    Cellulare VARCHAR(10),
);
```

Personale di Bordo (**ID, Capotreno, Macchinista**)

```
CREATE TABLE Personale_Bordo (
    Identificativo SERIAL NOT NULL
        CONSTRAINT gruppo_ferrovieri_pkey
            PRIMARY KEY,
    Capotreno      codice_fiscale NOT NULL
        CONSTRAINT fk_gruppo_ferrovieri_capotreno
            REFERENCES ferroviere,
    Macchinista    codice_fiscale NOT NULL
        CONSTRAINT fk_gruppo_ferrovieri_macchinista
            REFERENCES ferroviere
);
```

Posto (Numero, **Carrozza, IDtreno,** PDA)

```
CREATE TABLE Posto (
    Numero   SMALLINT NOT NULL,
    Carrozza SMALLINT NOT NULL,
    Pda      BOOLEAN,
    Idtreno  SMALLINT NOT NULL,
    PRIMARY KEY (numero, carrozza, idtreno),
    CONSTRAINT fk_posto_treno
```

```
        FOREIGN KEY (carrozza, idtreno) REFERENCES carrozza
            ON UPDATE CASCADE
);
```

Punto Interesse (Nome, Coordinate)

```
CREATE TABLE Punto_Interesse (
  Nome VARCHAR(100) NOT NULL PRIMARY KEY,
  Coordinate VARCHAR(40) NOT NULL
);
```

Riparatori (**Manutenzione**, **Operaio**)

```
CREATE TABLE Riparatori (
  Manutenzione INT,
  Operaio CODICE_FISCALE,
  CONSTRAINT fk_riparatori_id
    FOREIGN KEY(Manutenzione)
      REFERENCES Manutenzione(identificativo)
      ON UPDATE CASCADE,
  CONSTRAINT fk_riparatori_cf
    FOREIGN KEY(Operaio)
      REFERENCES Operaio(impiegato)
      ON UPDATE CASCADE,
  PRIMARY KEY (Manutenzione, Operaio)
);
```

Segretario (**Impiegato**, Account Password)

```
CREATE TABLE Segretario (
  Impiegato CODICE_FISCALE PRIMARY KEY,
  Account_password VARCHAR(30) NOT NULL,
  CONSTRAINT fk_segretario_cf
    FOREIGN KEY(Impiegato)
      REFERENCES Impiegato(CF)
      ON UPDATE CASCADE
);
```

Servizio Assistente (**Assistente, IDgruppo**)

```
CREATE TABLE Servizio_Assistente (
    Assistente codice_fiscale NOT NULL
        CONSTRAINT fk_servizio_assistente_assistente
            REFERENCES assistente
            ON UPDATE CASCADE,
    Idgruppo   INTEGER        NOT NULL
        CONSTRAINT fk_servizio_assistente_idgruppo
            REFERENCES personale_bordo
            ON UPDATE CASCADE,
    PRIMARY KEY (assistente, idgruppo)
);
```

Sezione (**Percorso,** Ordine, **Tratta_Limit, Tratta_Inizio, Tratta_Fine**, ***Binario**, Entrata Programmata, Uscita Programmata, Sosta Programmata, Sosta?)

```
CREATE TABLE Sezione (
    Percorso            INTEGER                NOT NULL
        CONSTRAINT fk_sezione_percorso
            REFERENCES percorso,
    Tratta_limit        SMALLINT               NOT NULL,
    Tratta_inizio       VARCHAR(100)           NOT NULL,
    Tratta_fine         VARCHAR(100)           NOT NULL,
    Ordine              SMALLINT DEFAULT 0     NOT NULL,
    Entrata_programmata TIME                   NOT NULL,
    Uscita_programmata  TIME                   NOT NULL,
```

```
    "fermata?"          BOOLEAN  DEFAULT FALSE NOT NULL,
    Sosta               INTEGER  DEFAULT 0     NOT NULL,
    Binario             VARCHAR(10),
    PRIMARY KEY (percorso, ordine),
    CONSTRAINT fk_sezione_tratta
        FOREIGN KEY (tratta_inizio, tratta_limit, tratta_fine) REFERENCES tratta
(inizio, limite_velocita, fine)
            ON UPDATE CASCADE,
    CONSTRAINT sezione_binario_stazione_numero_fk
        FOREIGN KEY (tratta_fine, binario) REFERENCES binario,
    CONSTRAINT sezione_check
        CHECK (uscita_programmata > entrata_programmata)
);
```

Sezione Effettiva (Identificativo, **Percorso, Sezione_Ordine, Giorno,** Entrata, Uscita, **Binario, Stazione**)

```
CREATE TABLE Sezione_Effettiva (
    Uscita         TIME,
    Entrata        TIME     NOT NULL,
    Sezione_ordine SMALLINT NOT NULL,
    Percorso       INTEGER  NOT NULL,
    Giorno         DATE     NOT NULL,
    Identificativo BIGSERIAL
        CONSTRAINT sezione_effettiva_pk
            PRIMARY KEY,
    Binario        VARCHAR(10),
    Stazione       VARCHAR(30),
    CONSTRAINT sezione_effettiva_spostamento_fk
        FOREIGN KEY (percorso, giorno) REFERENCES spostamento,
    CONSTRAINT sezione_effettiva_sezione_ordine_percorso_fk
        FOREIGN KEY (sezione_ordine, percorso) REFERENCES sezione (ordine,
percorso),
    CONSTRAINT sezione_effettiva_binario_numero_stazione_fk
        FOREIGN KEY (binario, stazione) REFERENCES binario (numero, stazione),
    CONSTRAINT sezione_effettiva_check
        CHECK (uscita > entrata)
);
```

Spostamento (**Percorso,** Giorno, **Treno, IDgruppo**, Tipologia)

```
CREATE TABLE Spostamento (
    Percorso INTEGER  NOT NULL
        CONSTRAINT fk_spostamento_percorso
            REFERENCES percorso,
    Treno    SMALLINT NOT NULL
        CONSTRAINT fk_itinerario_treno
            REFERENCES treno
            ON UPDATE CASCADE,
    Giorno   DATE     NOT NULL
    CONSTRAINT spostamento_giorno_check
        CHECK (giorno > CURRENT_DATE),
    Idgruppo INTEGER  NOT NULL
        CONSTRAINT fk_spostamento_gruppo_ferrovieri
            REFERENCES personale_bordo
            ON UPDATE CASCADE,
    CONSTRAINT itinerario_pkey
        PRIMARY KEY (percorso, giorno)
);
```

Stazione (**Nome**, Via, Città, Civico, Cap, NumeroBinari, ADA)

```
CREATE TABLE Stazione (
  Nome VARCHAR(100) PRIMARY KEY ,
  Via VARCHAR(50) NOT NULL,
  Città VARCHAR(30) NOT NULL,
  Civico INTEGER NOT NULL,
```

```
  Cap INTEGER NOT NULL,
  NumeroBinari SMALLINT NOT NULL,
  ADA BOOLEAN NOT NULL,
  CONSTRAINT fk_stazione_nome
    FOREIGN KEY(Nome)
      REFERENCES Punto_Interesse(Nome)
      ON UPDATE CASCADE
);
```

Tratta (Limite_Velocità, **Inizio, Fine**, Lunghezza, Numero_Binari)

```
CREATE TABLE Tratta (
    Limite_velocità SMALLINT     NOT NULL,
    Inizio          VARCHAR(100) NOT NULL
        CONSTRAINT fk_tratta_inizio
            REFERENCES punto_interesse
            ON UPDATE CASCADE,
    Fine            VARCHAR(100) NOT NULL
        CONSTRAINT fk_tratta_fine
            REFERENCES punto_interesse
            ON UPDATE CASCADE,
    Lunghezza       INTEGER      NOT NULL,
    PRIMARY KEY (limite_velocità, inizio, fine)
);
```

Treno (Identificativo, **Modello_Treno**)

```
CREATE TABLE Treno (
    Identificativo SMALLINT    NOT NULL
        PRIMARY KEY,
    Modello_treno  VARCHAR(30) NOT NULL
        CONSTRAINT fk_treno_nome
            REFERENCES modello_treno
            ON UPDATE CASCADE
);
```

Turno (**Impiegato**, Giorno, Inizio, Fine, Straordinario)

```
CREATE TABLE Turno (
    Impiegato codice_fiscale NOT NULL
        CONSTRAINT fk_turno_cf
            REFERENCES impiegato
            ON UPDATE CASCADE,
    Giorno        DATE NOT NULL,
    Inizio        TIME NOT NULL,
    Fine          TIME NOT NULL,
    Straordinario BOOLEAN DEFAULT FALSE NOT NULL,
    PRIMARY KEY (impiegato, giorno, straordinario)
    CONSTRAINT turno_check
        CHECK (inizio < fine)
);
```

Ufficio (Identificativo, Via, Città, Civico, Cap, Tipologia, Posti, **Dirigente**)

```
CREATE TABLE Ufficio (
    Via VARCHAR(50) NOT NULL,
    Tipologia tipo_ufficio NOT NULL,
    Posti INTEGER NOT NULL,
    Città VARCHAR(30) NOT NULL,
    Civico INTEGER NOT NULL,
    Cap INTEGER NOT NULL,
    Identificativo SERIAL NOT NULL PRIMARY KEY
);
```

Views

UniTrain: Definizione base di dati in SQL

Assistenti_Attuali

```
CREATE VIEW assistenti_attuali AS
    SELECT *
    FROM assistente, impiegato, contratto
    WHERE assistente.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Assistente'
    AND contratto.impiegato = impiegato.cf
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
    AND contratto.rescisso = false;
```

Capotreni_Attuali

```
CREATE VIEW capotreni_attuali AS
    SELECT *
    FROM ferroviere, impiegato, contratto
    WHERE ferroviere.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Ferroviere'
    AND ferroviere.ruolo = 'Capotreno'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Macchinisti_Attuali

```
CREATE VIEW macchinisti_attuali AS
    SELECT *
    FROM ferroviere, impiegato, contratto
    WHERE ferroviere.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Ferroviere'
    AND ferroviere.ruolo = 'Macchinista'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Ferrovieri_Attuali

```
CREATE VIEW ferrovieri_attuali AS
    SELECT *
    FROM ferroviere, impiegato, contratto
    WHERE ferroviere.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Ferroviere'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Operai_Attuali

```
CREATE VIEW operai_attuali AS
    SELECT *
    FROM operaio, impiegato, contratto
    WHERE operaio.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Operaio'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Segretari_Atttuali

```
CREATE VIEW segretari_attuali AS
    SELECT *
```

```
    FROM segretario, impiegato, contratto
    WHERE segretario.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Segretario'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Types

```
CREATE TYPE tipo_ufficio AS ENUM ('Dirigenza', 'Gestione Percorsi', 'Supporto
Clienti', 'Gestione Turni', 'Risorse Umane', 'Tecnici' );
CREATE TYPE ruolo_ferroviere AS ENUM ('Capotreno', 'Macchinista');
CREATE TYPE tipo_spostamento AS ENUM ('Viaggio', 'Spostamento')
CREATE TYPE livelli_account AS ENUM  ('Standard', 'Silver', 'Gold', 'Platinum')
CREATE TYPE tipo_assistenza AS ENUM ('Generale', 'Account', 'Biglietto')
CREATE TYPE tipo_contratto AS ENUM ('Full_Time', 'Part_Time');
CREATE TYPE tipo_impiegato AS ENUM ('Ferroviere', 'Assistente', 'Segretario',
'Operaio')
```

Domains

```
CREATE DOMAIN email AS VARCHAR(100) check ( value ~ '^\w+@[a-zA-Z_]+?\.[a-zA-
Z]{2,3}$' );
CREATE DOMAIN codice_fiscale AS VARCHAR(16) check ( length(value) = 16 );
CREATE DOMAIN iban AS VARCHAR(27) check ( length(value) = 27 );
```

Trigger

ASSEGNAZIONE_DEPOSITO

-Controllare che un operaio non sia assegnato a due depositi diversi contemporaneamente

```
CREATE FUNCTION insert_assegnazione_deposito() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
    --Controlliamo se esiste già un record senza fine impostata in cui risulta
il nostro operaio
        SELECT count(*) INTO valore
        FROM assegnazione_deposito
        where assegnazione_deposito.operaio = new.operaio AND fine IS NULL;
    --Se non esiste, prendiamo il timestamp corrente e restituiamo il record
```

```
    IF (valore = 0) THEN
        new.inizio = now()::TIMESTAMP;
        RETURN (NEW);
    --Altrimenti segnalarlo
    ELSE
        RAISE EXCEPTION 'Doppia_Assegnazione' USING HINT = 'Questo impiegato è
già stato assegnato ad un deposito, ' ||
                                                    'per favore prima
rimuoverlo dal deposito e dopo inserirlo in quello nuovo';
    END IF;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER insert_assegnazione_deposito
    BEFORE INSERT ON assegnazione_deposito
    FOR EACH ROW
    EXECUTE PROCEDURE insert_assegnazione_deposito()
```

ASSEGNAZIONE_UFFICIO

-Controllare che un segretario non sia assegnato a due uffici diversi contemporaneamente

```
CREATE FUNCTION insert_assegnazione_ufficio() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
        --Prima controlliamo se l'impiegato è un impiegato attuale (e che quindi
abbia un contratto valido e/o altre mansioni)
        SELECT count(*) INTO valore
        FROM segretari_attuali
        WHERE new.segretario = segretari_attuali.cf;

        IF valore = 0 THEN
            RAISE EXCEPTION 'L''impiegato definito non è un segretario';
        END IF;

        --Se lo è, controllare che non sia assegnato ad altri uffici nel mentre
        SELECT count(*) INTO valore
        FROM assegnazione_ufficio
        where assegnazione_ufficio.segretario = new.segretario AND fine IS NULL;
    IF (valore = 0) then
        new.inizio = now()::timestamp;
        return (NEW);
    ELSE
         RAISE EXCEPTION 'Doppia_Assegnazione' USING HINT = 'Questo segretario è
già stato assegnato ad un ufficio, ' ||
                                                    'per favore prima
rimuoverlo dal suo vecchio ufficio e dopo inserirlo in quello nuovo';
    END IF;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER assegnazione_ufficio_insert
    BEFORE INSERT ON assegnazione_ufficio
    FOR EACH ROW
    EXECUTE PROCEDURE insert_assegnazione_ufficio()
```

-Se l'impiegato è un dirigente nell'ufficio nella quale se ne sta andando, impedire l'aggiunta del timestamp rappresentante la fine di assegnazione in quell'ufficio

```
CREATE FUNCTION assegnazione_ufficio_update() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
    --Se ad essere diversa è solamente la data di fine (data che in precedenza
non era stata impostata)
    IF old.fine IS NOT NULL THEN
        IF new.fine != old.fine AND old.segretario = new.segretario
               AND old.ufficio = new.ufficio AND old.inizio = new.inizio THEN
            --Allora controllare che non sia un dirigente in questo momento
            SELECT count(*) INTO valore
            FROM direzione_ufficio
            WHERE direzione_ufficio.ufficio = old.ufficio
            AND direzione_ufficio.fine IS NULL;
            IF valore > 0 THEN
                RAISE EXCEPTION 'Questo segretario è il dirigente, non puoi
rimuoverlo da qui';
            end if;
        end if;
    end if;
    return new;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER assegnazione_ufficio_update
    BEFORE UPDATE ON assegnazione_ufficio
    FOR EACH ROW
    EXECUTE PROCEDURE assegnazione_ufficio_update()
```

ASSISTENTE

- Controlliamo se l'impiegato è presente nella lista degli impiegati attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION check_assistente() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
    --Controlliamo se l'impiegato è presente nella lista degli impiegati attuali
    --(e che quindi abbia un contratto valido e non abbia altre mansioni)
        SELECT count(*) INTO valore
        FROM assistenti_attuali
        where cf = new.impiegato;
    --Se è presente restituiamo il record
    IF valore > 0 THEN
        RETURN new;
    ELSE
        RAISE EXCEPTION 'L impiegato non è un assistente';
    END IF;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_assistente
    AFTER INSERT ON assistente
    FOR EACH ROW
    EXECUTE PROCEDURE check_assistente()
```

BIGLIETTO

-Ogni volta che si aggiunge un elemento bisogna aumentare il totale del saldo punti dell'account relativo e controllare se si è saliti di livello. Nel caso, aggiornare il livello.

```
create function accumulo_punti_funct() returns trigger
    language plpgsql
as
$$
BEGIN
        UPDATE account
        SET saldo_punti = saldo_punti + floor(new.prezzo)::integer
        WHERE codice_fedeltà = new.account;
        EXECUTE controllo_livello(new.account);
    END;
$$;


create trigger accumulo_punti_trigger
    after insert
    on biglietto
    for each row
execute procedure accumulo_punti_funct();

create function rimuovi_punti_funct() returns trigger
    language plpgsql
as
$$
BEGIN
        UPDATE account
        SET saldo_punti = saldo_punti - floor(new.prezzo)::integer
        WHERE codice_fedeltà = new.account;
        EXECUTE controllo_livello(new.account);
        RETURN NULL;
    END;
$$;

create trigger rimuovi_punti_trigger
    after delete
    on biglietto
    for each row
execute procedure rimuovi_punti_funct();
```

```
CREATE FUNCTION controllo_livello(fedeltà INTEGER) RETURNS VOID
    LANGUAGE plpgsql
AS
$BODY$
    DECLARE
        punti INTEGER;
        level livelli_account;
    BEGIN
        SELECT saldo_punti INTO punti
        FROM account
        WHERE codice_fedeltà = fedeltà;
        IF (punti > 0 AND punti < 1000) THEN level = 'Standard'; END IF;
        IF (punti >= 1000 AND punti < 2500) THEN level = 'Silver'; END IF;
        IF (punti >= 2500 AND punti <5000) THEN level = 'Gold'; END IF;
        IF (punti >= 5000) THEN level = 'Platinum'; END IF;
        UPDATE account
        SET livello = level
        WHERE codice_fedeltà = fedeltà;
```

```
    END;
$BODY$
```

-Controllare che gli account per la quale accumulo punti e il passeggero definito sul biglietto siano gli stessi

```
CREATE FUNCTION controllo_persona() RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$BODY$
    DECLARE
        EMAIL_CHECK EMAIL;
    BEGIN
        IF (new.account IS NULL) THEN RETURN NEW;
        else
            SELECT email INTO EMAIL_CHECK
            FROM account
            WHERE codice_fedeltà = new.account;
            IF (new.email = EMAIL_CHECK)
                THEN RETURN new;
            ELSE
                RAISE EXCEPTION 'Inconsistenza tra account e passeggero';
            END IF;
        END IF;
    END;
$$;
$BODY$;

CREATE TRIGGER controllo_persona_trigger
BEFORE INSERT ON biglietto
FOR EACH ROW
EXECUTE PROCEDURE controllo_persona();
```

-Controllare che le fermate di arrivo e di partenza della prenotazione siano possibili all'interno degli itinerari scelti.

```
CREATE FUNCTION controllo_fermate() RETURNS trigger AS
$$
DECLARE
        da integer;
        verso integer;
        valore integer;
    BEGIN
        SELECT sezione.* INTO da
        FROM sezione
        WHERE sezione.percorso = new.spostamento_percorso
        AND ((sezione.ordine = new.partenza_ordine - 1 AND sezione."fermata?")
        OR (sezione.ordine = 0 AND sezione.tratta_inizio IN (SELECT
s2.tratta_inizio
                                                             FROM sezione s2
                                                             WHERE s2.percorso =
new.spostamento_percorso
                                                             AND s2.ordine =
new.partenza_ordine)));


        SELECT count(*) INTO verso
        FROM sezione
        WHERE sezione.percorso = new.spostamento_percorso
        AND sezione.ordine = new.arrivo_ordine
        AND sezione."fermata?" = true;
```

```
        IF (da > 0) AND (verso > 0) THEN
    --Infine controlliamo se il posto scelto dall'utente è libero o già occupato
            SELECT count(DISTINCT biglietto.posto) INTO valore
            FROM biglietto, sezione partenza, sezione arrivo
            WHERE biglietto.spostamento_percorso = new.spostamento_percorso
            AND biglietto.data_viaggio = new.data_viaggio
            AND biglietto.carrozza = new.carrozza
            AND partenza.percorso = biglietto.spostamento_percorso
            AND arrivo.percorso = biglietto.spostamento_percorso
            AND partenza.ordine = biglietto.partenza_ordine
            AND arrivo.ordine = biglietto.arrivo_ordine
            AND biglietto.posto = new.posto
            AND ((arrivo.ordine >= new.partenza_ordine AND partenza.ordine <=
new.arrivo_ordine)
            OR (partenza.ordine <= new.arrivo_ordine AND arrivo.ordine >=
new.partenza_ordine));
            --Se il risultato della query è maggiore di 1, allora il posto è
occupato su tutto o su una parte del tracciato
            IF valore > 0 THEN
                RAISE EXCEPTION 'Il posto scelto non è valido, in questo
intervallo di stazioni questo posto è occupato';
            END IF;
            RETURN new;
        ELSE
            RAISE EXCEPTION 'Le stazioni non sono raggiungibili con questo
percorso';
        end if;
$$;
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_fermate_biglietto
    BEFORE INSERT ON biglietto
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_fermate()
```

CONTENUTO ASSISTENZA

-Controlliamo se a rispondere sono solo segretari autorizzati (ovvero che lavorano null'ufficio del supporto clienti) e che parlino la lingua della persona che ha scritto il messaggio originale

```
DECLARE
        segretarioc INTEGER;
    BEGIN

        SELECT count(*) INTO segretarioc
        FROM lingue_segretario, assegnazione_ufficio, ufficio
        WHERE lingue_segretario.segretario = new.segretario
        AND assegnazione_ufficio.segretario = new.segretario
        AND assegnazione_ufficio.fine IS NULL
        AND assegnazione_ufficio.ufficio = ufficio.identificativo
        AND lingue_segretario = (SELECT lingua
                                 FROM assistenza_clienti
                                 WHERE assistenza_clienti.identificativo =
new.assistenza);

        IF (segretarioc = 1) then
                return (NEW);
            else
                RAISE EXCEPTION 'Il segretario che sta rispondendo o non parla
```

```
quella lingua o non fa parte del supporto clienti';
        END IF;
    END
```

CONTRATTO

-Controllare che un contratto per una persona non sia vada ad accavallare con un altro contratto ancora attivo

```
create function controllo_contratto() returns trigger
    language plpgsql
as
$$
DECLARE
        data_fine DATE;
    BEGIN
        SELECT MAX(fine) INTO data_fine
        FROM contratto
        WHERE contratto.impiegato = new.impiegato
        IF (new.inizio < data_fine) then
            RAISE EXCEPTION 'Accavallameno Contratti';
        END IF;
        RETURN NEW;
END
$$;

create trigger controllo_contratto_trigger
    before insert
    on contratto
    for each row
execute procedure controllo_contratto();
```

DIREZIONE_UFFICIO

-Durante l'inserimento di completare l'incarico al dirigente precedente e di controllare che lavori nell'ufficio nella quale si sta dando l'incarico

```
CREATE FUNCTION insert_direzione() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
        vecchio_capo codice_fiscale;
        old_inizio TIMESTAMP;
    BEGIN
        --Controlliamo prima se il nuovo dirigente lavora in quell'ufficio
        SELECT count(*)
        FROM assegnazione_ufficio
        WHERE assegnazione_ufficio.ufficio = new.ufficio
        AND assegnazione_ufficio.segretario = new.segretario
        AND assegnazione_ufficio.fine IS NULL;

        IF valore = 0 THEN
            RAISE EXCEPTION 'Il nuovo dirigente non lavora in quell''ufficio';
        end if;

        --Se si, ci prendiamo il vecchio direttore (se esiste)
        SELECT segretario, inizio INTO vecchio_capo, old_inizio
        FROM direzione_ufficio
        WHERE ufficio = new.ufficio
        AND fine IS NULL;
```

```
        --E se esiste impostiamo la fine del suo mandato
        IF segretario IS NOT NULL THEN
            UPDATE segretario
            SET fine = current_timestamp
            WHERE ufficio = new.ufficio
            AND inizio = old_inizio
            AND ufficio = new.ufficio;
        end if;
        return new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER insert_direzione
    BEFORE INSERT
    ON direzione_ufficio
    FOR EACH ROW
EXECUTE PROCEDURE insert_direzione();
```

FERROVIERE

-Controlliamo se il ferroviere è presente nella lista dei ferrovieri attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION insert_ferroviere() RETURNS trigger
    LANGUAGE plpgsql
AS
$$
DECLARE
        valore INTEGER;
    BEGIN
        SELECT count(*) INTO valore
        FROM ferrovieri_attuali
        where cf = new.impiegato;
    IF valore > 0 THEN
        RETURN new;
    ELSE
        RAISE EXCEPTION 'L impiegato non è un ferroviere';
    END IF;
END
$$;

CREATE TRIGGER insert_ferroviere
    AFTER INSERT
    ON ferroviere
    FOR EACH ROW
EXECUTE PROCEDURE insert_ferroviere();
```

MANUTENZIONE

-Controllare che il treno di cui si sta iniziando la manutenzione sia effettivamente presente nel deposito

```
CREATE FUNCTION controllo_deposito() RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$BODY$
    DECLARE
        USCITA TIME;
        FINE VARCHAR(100);
        POSIZIONE_BANCO VARCHAR(100);
    BEGIN
        SELECT uscita, tratta_fine AS USCITA, FINE
```

```
        from sezione_effettiva
        where identificativo =
              (SELECT max(sezione_effettiva.identificativo)
               FROM sezione_effettiva, spostamento
               WHERE spostamento.treno = NEW.idtreno
               AND sezione_effettiva.percorso = spostamento.percorso
               AND sezione_effettiva.giorno = spostamento.giorno);

        IF USCITA IS NULL THEN
            RAISE EXCEPTION 'Treno in Viaggio';
        end if;

        SELECT deposito as POSIZIONE_BANCO
        FROM banco_manutenzione
        WHERE banco_manutenzione.identificativo = NEW.banco;

        IF POSIZIONE_BANCO = FINE THEN
            RETURN NEW;
        ELSE
            RAISE EXCEPTION 'Posizione errata';
        end if;
    END;
$BODY$;

CREATE TRIGGER controllo_posizione_treno
BEFORE INSERT ON manutenzione
FOR EACH ROW
EXECUTE PROCEDURE controllo_deposito();
```

-Controllare che il banco che si sta cercando di utilizzare non sia già occupato

```
CREATE FUNCTION assegnazione_banco() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
        SELECT count(*) INTO valore
        FROM manutenzione
        where manutenzione.banco = new.banco AND fine IS NULL;
    IF (valore = 0) then
        new.inizio = now()::timestamp;
        return (NEW);
    else
        RAISE EXCEPTION 'Doppia_Assegnazione' USING HINT = 'Questo banco è già
stato assegnato ad un altro treno';
    end if;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_banco
    BEFORE INSERT ON manutenzione
    FOR EACH ROW
    EXECUTE PROCEDURE assegnazione_banco()
```

OPERAIO

-Controlliamo se l'operaio è presente nella lista degli operai attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION check_operaio() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
        SELECT count(*) INTO valore
        FROM operai_attuali
        where cf = new.impiegato;
    IF valore > 0 THEN
        return new;
    else
        RAISE EXCEPTION 'L' impiegato non è un operaio';
    end if;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_operaio
    AFTER INSERT ON operaio
    FOR EACH ROW
    EXECUTE PROCEDURE check_operaio()
```

PERCORSO

-Non modificare un percorso se è già associato ad uno spostamento e aggiornare il percorso solo con dati validi

```
CREATE FUNCTION modifica_percorso() RETURNS trigger AS
$BODY$
DECLARE
        valore INTEGER;
        uscita TIME;
        arrivo VARCHAR(100);
    BEGIN
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE spostamento.percorso = new.identificativo;
        IF (valore > 0) THEN
            RAISE EXCEPTION 'Il percorso è già associato ad uno spostamento';
        END IF;

        IF new.tipologia != old.tipologia THEN
            SELECT count(*) INTO valore
            FROM sezione
            WHERE sezione.percorso = new.identificativo;

            IF valore > 0 THEN
                RAISE EXCEPTION 'Non puoi modificare la tipologia di un percorso
che ha già delle sezioni';
            end if;
        end if;


        SELECT count(*) INTO valore
        FROM sezione
        WHERE sezione.percorso = new.identificativo;

        IF valore = 0 AND new.ora_arrivo = null AND new.ora_partenza = null
               AND new.arrivo = null and new.partenza = null THEN
            RETURN new;
        end if;
```

```
        IF (new.partenza IS NOT NULL AND new.ora_partenza IS NOT NULL) THEN
            SELECT sezione.tratta_inizio, sezione.entrata_programmata INTO
arrivo, uscita
            FROM sezione
            WHERE sezione.percorso = new.identificativo
            AND sezione.ordine = 0;

            IF (uscita != new.ora_partenza OR arrivo != new.partenza) THEN
                RAISE EXCEPTION 'Dati non validi % % % %', arrivo, new.partenza,
uscita, new.ora_partenza;
            end if;
        end if;

        IF (new.ora_arrivo IS NOT NULL AND new.arrivo IS NOT NULL) THEN
            SELECT sezione.uscita_programmata, sezione.tratta_fine INTO uscita,
arrivo
            FROM sezione
            WHERE sezione.percorso = new.identificativo
            AND sezione.ordine =    (SELECT max(ordine)
                                     FROM sezione
                                     WHERE sezione.percorso =
new.identificativo);
            IF (uscita != new.ora_arrivo and arrivo != new.arrivo) THEN
                RAISE EXCEPTION 'Dati non validi % % % %', arrivo, new.arrivo,
uscita, new.ora_arrivo;
            end if;
        end if;
        RETURN new;
    END
$$;
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_possibile_modifica_percorso
    AFTER UPDATE ON percorso
    FOR EACH ROW
    EXECUTE PROCEDURE modifica_percorso()
```

-In fase iniziale di creazione, non inserire l'ora di arrivo e l'ora di partenza

```
CREATE FUNCTION reset_arrivo_percorso() RETURNS trigger AS
$BODY$
BEGIN
        new.ora_arrivo = NULL;
        new.arrivo = NULL;
        new.ora_partenza = NULL;
        new.partenza = NULL;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER reset_arrivo
    BEFORE INSERT ON percorso
    FOR EACH ROW
    EXECUTE PROCEDURE reset_arrivo_percorso()
```

PERSONALE DI BORDO

-Controllare che il capotreno sia un capotreno e il macchinista sia un macchinista (anche durante un eventuale aggiornamento dei dati)

```
CREATE FUNCTION check_ferroviere() RETURNS trigger AS
$BODY$
    DECLARE
        macchinista INTEGER;
        capotreno INTEGER;
    BEGIN
        SELECT count(*) INTO macchinista
        FROM macchinisti_attuali
        WHERE impiegato = new.macchinista;

        SELECT count(*) INTO capotreno
        FROM capotreni_attuali
        WHERE impiegato = new.capotreno;

        IF (macchinista > 0) then
            IF (capotreno > 0) then
                return (NEW);
            else
                RAISE EXCEPTION 'Capotreno Error' USING HINT = 'Il capotreno
definito non è un capotreno';
            end if;
        else
            RAISE EXCEPTION 'Macchinista Error' USING HINT = 'Il macchinista
definito non è un macchinista';
        end if;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_ferrovieri
    BEFORE INSERT ON personale_bordo
    FOR EACH ROW
    EXECUTE PROCEDURE check_ferroviere()

CREATE TRIGGER controllo_ferrovieri _update
    BEFORE UPDATE
    ON personale_bordo
    FOR EACH ROW
EXECUTE PROCEDURE check_ferroviere();
```

SEGRETARIO

- Controlliamo se il segretario è presente nella lista dei segretari attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION check_segretario() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
        SELECT count(*) INTO valore
        FROM segretari_attuali
        where cf = new.impiegato;
    IF valore > 0 THEN
        return new;
    else
        RAISE EXCEPTION 'L' impiegato non è un segretario';
    end if;
END
```

```
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_segretario
    AFTER INSERT ON segretario
    FOR EACH ROW
    EXECUTE PROCEDURE check_segretario()
```

SPOSTAMENTO

- L'ultima stazione referenziata dal percorso deve essere anche una fermata se la tipologia del percorso è impostata a "Viaggio"

```
CREATE FUNCTION controllo_fermata_finale() RETURNS trigger AS
$BODY$
    DECLARE
        valore BOOLEAN;
        tipologia tipo_spostamento;
    BEGIN

        SELECT percorso.tipologia INTO tipologia
        FROM percorso
        WHERE percorso.identificativo = new.percorso;

        IF tipologia = 'Spostamento' THEN
            RETURN NEW;
        END IF;

        SELECT "fermata?" INTO valore
        FROM sezione
        WHERE sezione.percorso = new.percorso
        AND sezione.ordine = (SELECT max(ordine)
                              FROM sezione
                              WHERE sezione.percorso = new.percorso);

        IF NOT valore THEN
            RAISE EXCEPTION 'Lo spostamento non fa fermata nella ultima
sezione';
        END IF;

        RETURN NEW;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_fermata_finale
    BEFORE INSERT ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_fermata_finale()
```

-Controllare che i ferrovieri stiano lavorando durante il loro turno e che non siano già assegnati ad un altro spostamento

```
CREATE FUNCTION controllo_turno_ferroviere() RETURNS trigger AS
$BODY$
    DECLARE
        assistenti RECORD;
        inizio_percorso TIME;
        fine_percorso TIME;
```

```
        inizio_turno TIME;
        fine_turno TIME;
        gruppi INTEGER;
    BEGIN
        SELECT count(*) INTO gruppi
        FROM spostamento
        WHERE idgruppo = NEW.idgruppo;
        IF gruppi > 0 THEN
                RAISE EXCEPTION 'Questo gruppo è già stato assegnato';
        END IF;


        SELECT ora_arrivo, ora_partenza INTO fine_percorso, inizio_percorso
        FROM percorso
        WHERE new.percorso = percorso.identificativo;

        SELECT inizio, fine INTO inizio_turno, fine_turno
        FROM turno, personale_bordo
        WHERE new.idgruppo = personale_bordo.identificativo
        AND turno.impiegato = personale_bordo.capotreno;


        IF (inizio_turno > inizio_percorso OR inizio_turno IS NULL) OR
(fine_turno < fine_percorso OR fine_turno IS NULL) THEN
                    RAISE EXCEPTION 'Il capotreno non ha il turno in quel
periodo';
                end if;

        SELECT inizio, fine INTO inizio_turno, fine_turno
        FROM turno, personale_bordo
        WHERE new.idgruppo = personale_bordo.identificativo
        AND turno.impiegato = personale_bordo.macchinista;

        RAISE NOTICE 'Inizio Percorso %, Fine Percorso %', inizio_percorso,
fine_percorso;
        RAISE NOTICE 'inizio_turno %, fine_turno %', inizio_turno, fine_turno;

        IF (inizio_turno > inizio_percorso OR inizio_turno IS NULL) OR
(fine_turno < fine_percorso OR fine_turno IS NULL) THEN
                    RAISE EXCEPTION 'Il macchinista non ha il turno in quel
periodo';
                end if;

        FOR assistenti IN
                        SELECT servizio_assistente.assistente
                        FROM personale_bordo, servizio_assistente
                        WHERE servizio_assistente.idgruppo = new.idgruppo
            loop
                SELECT inizio, fine INTO inizio_turno, fine_turno
                FROM turno, personale_bordo
                WHERE new.idgruppo = personale_bordo.identificativo
                AND turno.impiegato = assistenti.assistente;

        IF (inizio_turno > inizio_percorso OR inizio_turno IS NULL) OR
(fine_turno < fine_percorso OR fine_turno IS NULL) THEN
                    RAISE EXCEPTION 'L''assistente % non ha il turno in quel
periodo', assistenti.assistente;
                end if;
            end loop;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;
```

```
CREATE TRIGGER controllo_turno_ferroviere
    BEFORE UPDATE ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_turno_ferroviere()

CREATE TRIGGER controllo_turno_ferroviere_insert
    BEFORE INSERT
    ON spostamento
    FOR EACH ROW
EXECUTE PROCEDURE controllo_turno_ferroviere();
```

-Controllare che gli spostamenti non si vadano ad accavallare su una determinata tratta

```
CREATE FUNCTION controllo_accavallamento() RETURNS trigger AS
$BODY$
    DECLARE
        f RECORD;
        valore INTEGER;
    BEGIN
        FOR f IN
                SELECT *
                FROM sezione
                WHERE sezione.percorso = new.percorso
            LOOP
                SELECT count(*) INTO valore
                FROM sezione, percorso p1, spostamento
                WHERE spostamento.giorno = new.giorno
                AND spostamento.percorso = p1.identificativo
                AND sezione.percorso = p1.identificativo
                AND uscita_programmata <= f.uscita_programmata
                AND entrata_programmata >= f.entrata_programmata;

                IF valore > 0 THEN
                    RAISE EXCEPTION 'Il treno durante questo viaggio si scontra
durante il viaggio con un altro treno nella sezione numero %', f.ordine;
                end if;
            end loop;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_accavallamento_spostamento
    BEFORE INSERT ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_accavallamento()
```

-Un treno può partire solo dal punto di arrivo dell'ultimo viaggio e solo se il percorso è stato definito

```
CREATE FUNCTION controllo_partenza_treno() RETURNS trigger AS
$BODY$
DECLARE
        ora_arrivo_percorso TIME;
        partenza_percorso VARCHAR(100);
        ora_partenza_check TIME;
        arrivo_check VARCHAR(100);
        giorno_check DATE;
        valore INTEGER;
    BEGIN
```

```
        SELECT partenza, ora_partenza INTO partenza_percorso
,ora_arrivo_percorso
        FROM percorso
        WHERE identificativo = new.percorso;

        -- Se il percorso non contiene tratte, allora impossibile
        IF ora_partenza_check = NULL THEN
            RAISE EXCEPTION 'Il percorso non è ancora stato definito, definisci
prima un percorso';
        end if;

        -- Se il treno è un nuovo treno, basta che parta da un deposito
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE spostamento.treno = new.treno;

        IF valore = 0 THEN
            SELECT count(*) INTO valore
            FROM deposito, percorso
            WHERE percorso.identificativo = new.percorso
            AND deposito.nome = percorso.partenza;
            IF valore > 0 THEN
                RETURN new;
            end if;
        end if;

        -- Altrimenti ci prendiamo i dati dell'ultimo spostamento del treno
        SELECT p0.arrivo, p0.ora_arrivo, s0.giorno INTO arrivo_check,
ora_partenza_check, giorno_check
        FROM spostamento s0, percorso p0
        WHERE s0.percorso = p0.identificativo
        AND s0.treno = new.treno
        AND s0.giorno = (SELECT max(s1.giorno)
                         FROM spostamento s1
                         WHERE s1.treno = new.treno)
        AND p0.ora_arrivo >= all (SELECT ora_partenza
                                  FROM percorso p1, spostamento s1
                                  WHERE s1.percorso = p1.identificativo
                                  AND s1.treno = new.treno
                                  AND s1.giorno = (SELECT MAX(s2.giorno)
                                                   FROM spostamento s2
                                                   WHERE s2.treno = new.treno));

        IF (arrivo_check = partenza_percorso) THEN
            IF (giorno_check < new.giorno) THEN
                RETURN new;
            ELSE
                IF (giorno_check = new.giorno AND ora_arrivo_percorso >
ora_partenza_check) THEN
                    RETURN NEW;
                ELSE
                    RAISE EXCEPTION 'Il treno a quella ora è ancora in viaggio';
                end if;
            end if;
        end if;
        RAISE EXCEPTION 'Il percorso non è compatibile con la ultima posizione
finale del treno';
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_partenza_treno
    BEFORE INSERT ON spostamento
```

```
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_partenza_treno()

CREATE TRIGGER controllo_partenza_treno_update
    BEFORE UPDATE
    ON spostamento
    FOR EACH ROW
EXECUTE PROCEDURE controllo_partenza_treno();
```

-Non cancellare gli spostamenti effettuati nel passato o che non siano l'ultimo spostamento programmato per quel treno

```
CREATE FUNCTION check_delete_spostamento() RETURNS trigger AS
$BODY$
    DECLARE
        orario TIME;
        giorno_max DATE;
        orario_max TIME;
    BEGIN
        --Controlliamo che la data non sia nel passato (vuol dire che lo
spostamento è stato già effettuato)
        IF old.giorno < current_date THEN
            RAISE EXCEPTION 'Non puoi cancellare spostamenti effettuati nel
passato';
        end if;
        --Se il giorno è lo stesso, controlliamo che l'orario sia maggiore
dell'ora attuale
        -- (altrimenti vorrebbe dire che il treno è già partito)
        IF old.giorno = current_date THEN
            SELECT ora_partenza INTO orario
            FROM percorso
            WHERE identificativo = old.percorso;
            IF orario < current_time THEN
                RAISE EXCEPTION 'Non puoi cancellare spostamenti effettuati nel
passato';
            end if;
        end if;

        SELECT max(spostamento.giorno), max(percorso.ora_partenza) INTO
giorno_max, orario_max

        FROM spostamento, percorso
        WHERE spostamento.percorso = percorso.identificativo
        AND spostamento.treno = new.treno;

        --Infine controlliamo che sia l'ultimo viaggio programmato per quel
treno, altrimenti dare errore
        IF giorno_max > new.giorno THEN
            RAISE EXCEPTION 'Non puoi cancellare uno spostamento con degli
spostamenti futuri che dipendono da questo';
        end if;
        IF giorno_max = new.giorno THEN
            IF orario_max > orario THEN
                RAISE EXCEPTION 'Non puoi cancellare uno spostamento con degli
spostamenti futuri che dipendono da questo';
            end if;
        end if;
        return old;
    END
  LANGUAGE plpgsql;

create trigger check_delete_spostamento
    before delete
```

```
    on spostamento
    for each row
execute procedure check_delete_spostamento();
```

SEZIONE EFFETTIVA

-Quando si inserisce un record, il valore di entrata deve essere maggiore del valore dell'uscita precedente, se esiste

```
CREATE FUNCTION check_insert_sezione_effettiva() RETURNS trigger AS
$BODY$
    DECLARE
        partenza TIME;
    BEGIN
        IF NEW.sezione_ordine = 0 THEN
            SELECT entrata_programmata INTO partenza
            FROM sezione
            WHERE sezione.percorso = new.percorso
            AND sezione.ordine = 0;

            IF new.entrata >= partenza THEN
                RETURN new;
            ELSE
                RAISE EXCEPTION 'Non puoi partire prima della partenza';
            end if;
        end if;

        SELECT uscita INTO partenza
        FROM sezione_effettiva
        WHERE sezione_ordine = new.sezione_ordine-1
        AND sezione_effettiva.percorso = new.percorso
        AND sezione_effettiva.giorno = new.giorno;

        IF new.entrata >= partenza THEN
                RETURN new;
            ELSE
                RAISE EXCEPTION 'Inconsistenza sugli orari di uscita e entrata
tra due sezioni';
            end if;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER check_insert_sezione_effettiva
    BEFORE INSERT ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE check_insert_sezione_effettiva()
```

-Quando si aggiorna un record, il valore di uscita deve essere maggiore del valore di entrata

```
CREATE FUNCTION check_update_sezione_effettiva() RETURNS trigger AS
$BODY$
    DECLARE
        arrivo TIME;
    BEGIN
        SELECT entrata INTO arrivo
        FROM sezione_effettiva
        WHERE sezione_ordine = new.sezione_ordine-1
        AND sezione_effettiva.percorso = new.percorso
        AND sezione_effettiva.giorno = new.giorno;

        IF new.uscita >= arrivo THEN
```

```
                RETURN new;
            ELSE
                RAISE EXCEPTION 'Inconsistenza sugli orari di uscita e entrata
nella sezione';
            end if;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER check_update_sezione_effettiva
    BEFORE UPDATE ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE check_update_sezione_effettiva()
```

-Non si può cancellare un record di sezione_effettiva

```
CREATE FUNCTION stop_remove() RETURNS trigger AS
$BODY$
    BEGIN
        RAISE EXCEPTION 'Questo record non può essere rimosso';
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER block_remove_sezione_effettiva
    BEFORE DELETE ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE stop_remove()
```

-Quando si aggiorna il campo uscita_effettiva di una sezione bisogna controllare che sia l'ultima dell'itinerario. Nel caso sia l'ultima, controllare che i ferrovieri interessati non abbiano superato l'orario del turno. Nel caso, aggiungere lo straordinario

```
CREATE FUNCTION check_straordinario() RETURNS trigger AS
$BODY$
    DECLARE
        ordine_massimo INTEGER;
        ora_arrivo TIME;
        fine_turno TIME;
        impiegati RECORD;
        valore INTEGER;
    BEGIN
        SELECT max(ordine) INTO ordine_massimo
        FROM sezione
        WHERE sezione.percorso = new.percorso;

        IF new.sezione_ordine != ordine_massimo THEN
            RETURN new;
        END IF;

        SELECT sezione.uscita_programmata INTO ora_arrivo
        FROM sezione
        WHERE sezione_ordine = ordine_massimo
        AND sezione.percorso = new.percorso;

        IF ora_arrivo >= new.uscita THEN
            RETURN new;
        END IF;
        --Ci prendiamo i codici fiscali di tutti gli impiegati presenti sul
treno durante lo spostamento e iteriamo su essi
        FOR impiegati IN SELECT impiegato.cf
                        FROM impiegato
                        WHERE cf = (SELECT ferroviere.impiegato
                        FROM ferroviere, spostamento, personale_bordo
```

```
                        WHERE new.percorso = spostamento.percorso
                        AND new.giorno = spostamento.giorno
                        AND spostamento.idgruppo = personale_bordo.identificativo
                        AND (capotreno = ferroviere.impiegato
                        OR macchinista = ferroviere.impiegato))
                        OR cf =     (SELECT assistente.impiegato
                         FROM assistente, spostamento, servizio_assistente
                         WHERE new.percorso = spostamento.percorso
                         AND new.giorno = spostamento.giorno
                         AND spostamento.idgruppo = servizio_assistente.idgruppo
                       AND servizio_assistente.assistente = assistente.impiegato)
LOOP
        --Ci prendiamo la fine del turno dell'impiegato che stiamo analizzando
            SELECT fine INTO fine_turno
            FROM turno
            WHERE impiegati.cf = turno.impiegato
            AND new.giorno = turno.giorno
            AND turno.straordinario = false;

            --Nel caso ci sia bisogno di aggiungere uno straordinario
            IF fine_turno < new.uscita THEN
                SELECT count(*) INTO valore
                FROM turno
                WHERE impiegati.cf = turno.impiegato
                AND new.giorno = turno.giorno
                AND turno.straordinario = true;
                --Rimuoviamo l'eventuale già presente straordinario...
                IF valore > 0 THEN
                    DELETE FROM turno
                    WHERE impiegati.cf = turno.impiegato
                    AND turno.giorno = new.giorno
                    AND turno.straordinario = true;
                END IF;
                --E lo "aggiorniamo" aggiungendo quello nuovo (o semplicemente
ne aggiungiamo uno nuovo)
                INSERT INTO turno (impiegato, giorno, inizio, fine,
straordinario)
                VALUES (impiegati.cf, new.giorno, fine_turno, new.uscita, true);
            END IF;
        END LOOP;

        RETURN NEW;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER check_straordinario
    BEFORE UPDATE ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE check_straordinario()
```

SEZIONE

-Impostiamo correttamente l'ordine della sezione

```
CREATE FUNCTION set_ordine() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
        SELECT count(*) INTO valore
        FROM sezione
```

```
        WHERE percorso = new.percorso;
        new.ordine = valore;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER set_ordine
    BEFORE INSERT ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE set_ordine()
```

-Non si possono modificare le sezioni

```
CREATE FUNCTION update_sezione() RETURNS trigger AS
$BODY$
    DECLARE
    BEGIN
        RAISE EXCEPTION 'Non puoi modificare le sezioni, devi prima
cancellarle';
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER modify_sezione
    AFTER UPDATE ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE update_sezione()
```

-Non si può cancellare una sezione che è già associata ad uno spostamento o che abbia valore di ordine minore del massimo valore per quel percorso.
Nel caso si possa cancellare la sezione, modificare i dati del percorso.

```
CREATE FUNCTION delete_sezione() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
        sezione_precedente VARCHAR(100);
        uscita_precedente TIME;
    BEGIN
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE percorso = new.percorso;

        IF valore > 0 THEN
            RAISE EXCEPTION 'Non puoi rimuovere sezioni a questo percorso';
        end if;

        SELECT count(*) INTO valore
        FROM sezione
        WHERE sezione.percorso = old.percorso;

        IF valore = 0 THEN
            UPDATE percorso
            SET partenza = null, ora_partenza = null, arrivo = null, ora_arrivo
= null
            WHERE percorso.identificativo = old.percorso;
        end if;

        IF valore = old.ordine THEN
            SELECT sezione.tratta_fine, uscita_programmata INTO
sezione_precedente, uscita_precedente
            FROM sezione
            WHERE sezione.ordine = valore - 1
```

```
            AND sezione.percorso = old.percorso;

            UPDATE percorso
            SET arrivo = sezione_precedente, ora_arrivo = uscita_precedente
            WHERE identificativo = old.percorso;
        ELSE
            RAISE EXCEPTION 'Non puoi rimuovere questa sezione';
        end if;
        RETURN old;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER delete_sezione
    AFTER DELETE ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE delete_sezione()
```

-Ogni volta che si aggiunge una sezione, aggiornare l'ora di arrivo e il punto di interesse di arrivo e la compatibilità con la precedente
Inoltre non si possono aggiungere sezioni ad un percorso che ha già uno spostamento
Inoltre controllare se la sezione è compatibile con la tipologia di percorso specificato
Inoltre se si passa in una stazione, controllare che si sia specificato un binario dove passare

```
CREATE FUNCTION insert_sezione() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
        sezione_precedente VARCHAR(100);
        uscita_precedente TIME;
        sosta_precedente INTEGER;
        tipologia2 tipo_spostamento;
    BEGIN
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE percorso = new.percorso;

        IF valore > 0 THEN
            RAISE EXCEPTION 'Non puoi aggiungere sezioni a questo percorso';
        end if;

        IF new.ordine > 0 THEN
            SELECT sezione.tratta_fine, uscita_programmata, sosta INTO
sezione_precedente, uscita_precedente, sosta_precedente
            FROM sezione
            WHERE ordine = new.ordine - 1
            AND sezione.percorso = new.percorso;

            uscita_precedente = uscita_precedente + ((sosta_precedente ||
'MINUTE')::INTERVAL);
            IF sezione_precedente != new.tratta_inizio OR uscita_precedente !=
new.entrata_programmata THEN
                RAISE EXCEPTION 'La sezione non è compatibile con la
precedente';
            end if;
        ELSE
            UPDATE percorso
            SET partenza = new.tratta_inizio, ora_partenza =
new.entrata_programmata
            WHERE identificativo = new.percorso;
        end if;
```

```
        SELECT count(*) INTO valore
        FROM deposito
        WHERE nome = new.tratta_inizio or nome = new.tratta_fine;

        SELECT percorso.tipologia INTO tipologia2
        FROM percorso
        WHERE percorso.identificativo = new.percorso;

        IF valore > 0 AND tipologia2 = 'Viaggio' THEN
            RAISE EXCEPTION 'Non puoi visitare dei depositi mentre fai servizio
viaggiatori';
        end if;

        SELECT count(*) INTO valore
        FROM stazione
        WHERE nome = new.tratta_fine;

        IF valore > 0 AND new.binario IS NULL THEN
            RAISE EXCEPTION 'Specificare su quale binario il treno farà
sosta/passerà';
        END IF;

        IF new."fermata?" THEN
            IF valore = 0 THEN
                RAISE EXCEPTION 'La tratta finale non è una stazione, pertanto
non possiamo fare fermata viaggiatori';
            end if;

            IF tipologia2 = 'Spostamento' THEN
                RAISE EXCEPTION 'Non puoi fare fermate viaggiatori durante uno
spostamento';
            end if;
        end if;

        UPDATE percorso
        SET arrivo = new.tratta_fine, ora_arrivo = new.uscita_programmata
        WHERE identificativo = new.percorso;

        RETURN new;
    END
$$;
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER insert_sezione
    AFTER INSERT ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE insert_sezione()
```

TURNO

-Controllare che il turno di un impiegato ricada in un periodo in cui lui ha il contratto

```
CREATE FUNCTION contratto_valido() RETURNS TRIGGER AS
$BODY$
DECLARE
        contratti INTEGER;
        tempo_fine TIME;
        tipol tipo_contratto;
    BEGIN
        SELECT MAX(fine) INTO tempo_fine
        FROM turno
```

```
        WHERE impiegato = new.impiegato
        AND giorno = new.giorno
        GROUP BY giorno;

        IF (tempo_fine IS NOT NULL AND new.fine < tempo_fine) then
            RAISE EXCEPTION 'Accavallameno Turni';
        END IF;

        SELECT contratto.identificativo, contratto.tipo INTO contratti, tipol
        FROM contratto
        WHERE contratto.impiegato = new.impiegato
        AND new.giorno > contratto.inizio
        AND new.giorno < contratto.fine;

        IF (contratti IS NOT NULL) THEN
            IF tipol = 'Part_Time' AND (new.inizio - new.fine) > interval '4
hours' THEN
                RAISE EXCEPTION 'Un part_time non può lavorare più di 4 ore';
            end if;
            RETURN NEW;
        ELSE RAISE EXCEPTION 'No Contratto';
        END IF;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_contratto_trigger
    BEFORE INSERT ON turno
    FOR EACH ROW
    EXECUTE PROCEDURE contratto_valido()
```

-Non cancellare turni nel passato o uno nel futuro in cui un ferroviere è impegnato

```
CREATE FUNCTION check_delete_turno() RETURNS trigger AS
$BODY$
DECLARE
    valore INTEGER;
BEGIN
        --Controlliamo che la data non sia nel passato (vuol dire che il turno è
stato già effettuato)
        IF old.giorno < current_date THEN
            RAISE EXCEPTION 'Non puoi cancellare un turno effettuato nel
passato';
        end if;
        --Se il giorno è lo stesso, controlliamo che l'orario di uscita sia
maggiore dell'ora attuale
        -- (altrimenti vorrebbe dire che il turno è finito)
        IF old.giorno = current_date THEN
            IF old.fine < current_time THEN
                IF old.straordinario = false THEN
     RAISE EXCEPTION 'Non puoi cancellare un turno effettuato nel passato';
END IF;
            end if;
        end if;

        SELECT count(*) INTO valore
        FROM ((spostamento
        INNER JOIN personale_bordo on spostamento.idgruppo =
personale_bordo.identificativo)
        INNER JOIN servizio_assistente on spostamento.idgruppo =
servizio_assistente.idgruppo)
        WHERE spostamento.giorno=old.giorno AND
(personale_bordo.capotreno=old.impiegato OR
personale_bordo.macchinista=old.impiegato OR
```

```
servizio_assistente.assistente=old.impiegato);
        IF valore>0 THEN
            RAISE EXCEPTION 'Non puoi cancellare un turno futuro già assegnato';
        end if;

        return old;
    END
$BODY$
  LANGUAGE plpgsql;

create trigger check_delete_turno
    before delete
    on turno
    for each row
execute procedure check_delete_turno();
```

-Non modificare turni nel futuro in cui un ferroviere è occupato

```
CREATE FUNCTION check_update_turno() RETURNS trigger
    LANGUAGE plpgsql
AS
$$
DECLARE
        valore INTEGER;
    BEGIN

        SELECT count(*) INTO valore
        FROM spostamento
        INNER JOIN personale_bordo on spostamento.idgruppo =
personale_bordo.identificativo
        INNER JOIN servizio_assistente on spostamento.idgruppo =
servizio_assistente.idgruppo
        INNER JOIN percorso on spostamento.percorso =
public.percorso.identificativo
        WHERE spostamento.giorno=old.giorno AND
(personale_bordo.capotreno=old.impiegato OR
personale_bordo.macchinista=old.impiegato OR
servizio_assistente.assistente=old.impiegato)
        AND (new.fine<old.fine AND new.fine<public.percorso.ora_arrivo) OR
(new.inizio>old.inizio AND new.inizio>public.percorso.ora_partenza );
        IF valore>0 THEN
            RAISE EXCEPTION 'Non puoi modificare un turno rimuovendo orari già
assegnati';
        end if;

        return new;
    END

$$;
CREATE TRIGGER check_update_turno
    BEFORE UPDATE
    ON turno
    FOR EACH ROW
EXECUTE PROCEDURE check_update_turno();
```