Graboid Studio

Implementazione Fisica

NOME CORSO

Basi di Dati

Autori

Menozzi Matteo (176906)
Patrini Andrea (176907)
Turci Sologni Enrico (176187)

16 luglio 2024

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Indice

1 Implementazione Fisica **2**
- 1.1 Scelta particolare di alcune chiavi primarie . . . 2
- 1.2 Schema Fisico . . . 2

2 Funzioni **10**
- 2.1 Introduzione alle Funzioni . . . 10
- 2.2 Implementazione delle Funzioni . . . 10

3 Procedure **13**
- 3.1 Introduzione alle Procedure . . . 13
- 3.2 Implementazione delle Procedure . . . 13

4 Trigger **25**
- 4.1 Introduzione ai Trigger . . . 25
- 4.2 Implementazione dei Triggers . . . 25
  - 4.2.1 Trigger derivati da Vincoli . . . 25
  - 4.2.2 Trigger necessari alle Procedure . . . 29

5 Query **34**
- 5.1 Introduzione alle Query . . . 34
- 5.2 Implementazione delle Query . . . 34
  - 5.2.1 Artista . . . 34
  - 5.2.2 Tecnico . . . 35
  - 5.2.3 Operatore . . . 36

6 Indci **40**
- 6.1 Introduzione agli Indici . . . 40
- 6.2 Implementazione e Motivazione degli Indici . . . 40
  - 6.2.1 Indice sull'attributo Artista dell'entità Ordine . . . 40
  - 6.2.2 Indice sull'attributo Data-Inizio dell'entità Produzione . . . 43
  - 6.2.3 Indice sull'attributo Giorno dell'entità Prenotazione . . . 44
  - 6.2.4 Indice sull'attributo Data di Registrazione dell'entità Canzone . . . 45
  - 6.2.5 Indice sull'attributo Data di Registrazione dell'entità artista . . . 46

1 Implementazione Fisica

1.1 Scelta particolare di alcune chiavi primarie

Abbiamo scelto di sostituire alcune chiavi primarie composte con un codice seriale per semplificare significativamente le query. Questo approccio ci consente di evitare chiavi composte molto lunghe, rendendo più agevole la gestione delle tabelle. Per mantenere l'unicità originariamente garantita dalle chiavi primarie composte, abbiamo convertito questi attributi in vincoli UNIQUE, come mostrato di seguito.

Consideriamo l'esempio:

```
CREATE TABLE CANZONE (
    -- Nuovo codice seriale
    codice SERIAL PRIMARY KEY,

    -- Attributi che componevano la vecchia chiave primaria
    titolo VARCHAR(255) NOT NULL,
    produzione INTEGER NOT NULL,

    -- Unicità della vecchia chiave primaria mantenuta
    CONSTRAINT unique_canzone UNIQUE (titolo, produzione),

    ...
);
```

Abbiamo applicato questa tecnica alle seguenti tabelle: *PRODUZIONE, CANZONE, PARTECIPAZIONE, ORDINE, PACCHETTO*. Le tabelle dipendenti sono state aggiornate per riferirsi ai nuovi codici seriali, mantenendo l'integrità referenziale del database. Tali tabelle sono: *CONDURRE, PAGAMENTO, PACCHETTO, PRENOTAZIONE, LAVORA_A, ORARIO*.

1.2 Schema Fisico

```
CREATE TABLE ARTISTA (
    nome_arte VARCHAR(255) PRIMARY KEY,
    data_di_registrazione DATE NOT NULL
);

CREATE TABLE GRUPPO (
    artista VARCHAR(255) PRIMARY KEY,
    data_formazione DATE NOT NULL,
    FOREIGN KEY (artista) REFERENCES ARTISTA(nome_arte)
    ON UPDATE CASCADE ON DELETE RESTRICT
);

CREATE TABLE SOLISTA (
    artista VARCHAR(255),
    codice_fiscale CHAR(16) UNIQUE,
    nome VARCHAR(255),
    cognome VARCHAR(255),
    data_di_nascita DATE NOT NULL,
    gruppo VARCHAR(255),
    data_adesione DATE,
```

```
    PRIMARY KEY (artista),
    FOREIGN KEY (artista) REFERENCES ARTISTA(nome_arte) ON UPDATE CASCADE ON DELETE
    ↪  CASCADE,
    FOREIGN KEY (gruppo) REFERENCES GRUPPO(artista) ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT data_di_nascita_corretta CHECK(date_part('year',
    ↪  data_di_nascita)>date_part('year', CURRENT_DATE)-100)
);

CREATE TABLE PARTECIPAZIONE_PASSATA (
    gruppo VARCHAR(255) NOT NULL,
    solista VARCHAR(255) NOT NULL,
    data_adesione DATE NOT NULL,
    data_fine_adesione DATE NOT NULL,
    PRIMARY KEY (gruppo, solista),
    FOREIGN KEY (gruppo) REFERENCES GRUPPO(artista) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (solista) REFERENCES SOLISTA(artista) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT data_fine_adesione_corretta CHECK(data_fine_adesione >= data_adesione)
);

CREATE TABLE EMAIL_A (
    email VARCHAR(255) PRIMARY KEY,
    artista VARCHAR(255) NOT NULL,
    FOREIGN KEY (artista) REFERENCES ARTISTA(nome_arte) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE TELEFONO_A (
    numero VARCHAR(15),
    artista VARCHAR(255),
    PRIMARY KEY (numero),
    FOREIGN KEY (artista) REFERENCES ARTISTA(nome_arte) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE TIPO_PRODUZIONE (
    nome VARCHAR(25) PRIMARY KEY,
    CHECK (nome IN ('Album', 'Singolo', 'EP'))
);

CREATE TABLE GENERE (
    nome VARCHAR(255) PRIMARY KEY
);

CREATE TABLE PRODUZIONE (
    -- Unique code
    codice SERIAL PRIMARY KEY,

    -- Old Primary Key
    titolo VARCHAR(255) NOT NULL,
    artista VARCHAR(255) NOT NULL,

    -- Old Primary Key Uniqueness Maintained
```

```
    CONSTRAINT unique_produzione UNIQUE (titolo, artista),
    FOREIGN KEY (artista) REFERENCES ARTISTA(nome_arte) ON UPDATE CASCADE ON DELETE
    ↪  RESTRICT,

    -- Other
    data_inizio DATE NOT NULL,
    data_fine DATE,
    stato VARCHAR(13) NOT NULL,
    tipo_produzione VARCHAR(25),
    genere VARCHAR(255),
    FOREIGN KEY (tipo_produzione) REFERENCES TIPO_PRODUZIONE(nome) ON UPDATE CASCADE ON
    ↪  DELETE SET NULL,
    FOREIGN KEY (genere) REFERENCES GENERE(nome) ON UPDATE CASCADE ON DELETE SET NULL,
    CHECK (data_inizio <= data_fine),
    CHECK (stato IN ('Produzione', 'Pubblicazione'))
);

CREATE TABLE CANZONE (
    -- Unique code
    codice SERIAL PRIMARY KEY,

    -- Old Primary Key
    titolo VARCHAR(255) NOT NULL,
    produzione INTEGER NOT NULL,

    -- Old Primary Key Uniqueness Maintained
    CONSTRAINT unique_canzone UNIQUE (titolo, produzione),
    FOREIGN KEY (produzione) REFERENCES PRODUZIONE(codice) ON UPDATE CASCADE ON DELETE
    ↪  RESTRICT,
    -- Other
    testo TEXT,
    data_di_registrazione DATE,
    lunghezza_in_secondi INTEGER,
    nome_del_file VARCHAR(255),
    percorso_di_sistema VARCHAR(255),
    estensione VARCHAR(10),
    CONSTRAINT lunghezza_in_secondi_corretta CHECK (lunghezza_in_secondi > 0)
);

CREATE TABLE PARTECIPAZIONE (
    -- Unique code
    codice SERIAL PRIMARY KEY,

    -- Old Primary Key
    solista VARCHAR(255) NOT NULL,
    canzone INTEGER NOT NULL,

    -- Old Primary Key Uniqueness Maintained
    CONSTRAINT unique_partecipazione UNIQUE (solista, canzone),
    FOREIGN KEY (solista) REFERENCES SOLISTA(artista) ON UPDATE CASCADE ON DELETE CASCADE,
```

```
    FOREIGN KEY (canzone) REFERENCES CANZONE(codice) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE PRODUTTORE (
    solista VARCHAR(255) PRIMARY KEY,
    FOREIGN KEY (solista) REFERENCES SOLISTA(artista)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE CONDURRE (
    produttore VARCHAR(255),
    produzione INTEGER,
    PRIMARY KEY (produttore, produzione),
    FOREIGN KEY (produttore) REFERENCES PRODUTTORE(solista) ON UPDATE CASCADE ON DELETE
    ↪  CASCADE,
    FOREIGN KEY (produzione) REFERENCES PRODUZIONE(codice) ON UPDATE CASCADE ON DELETE
    ↪  CASCADE
);

CREATE TABLE OPERATORE (
    codice_fiscale CHAR(16) PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    cognome VARCHAR(255) NOT NULL,
    data_di_nascita DATE NOT NULL,
    data_di_assunzione DATE NOT NULL,
    iban VARCHAR(34) UNIQUE,
    CONSTRAINT data_di_nascita_corretta CHECK(date_part('year',
    ↪  data_di_nascita)>date_part('year', CURRENT_DATE)-100)
);

CREATE TABLE EMAIL_O (
    email VARCHAR(255) NOT NULL,
    operatore CHAR(16) NOT NULL,
    PRIMARY KEY (email),
    FOREIGN KEY (operatore) REFERENCES OPERATORE(codice_fiscale) ON UPDATE CASCADE ON
    ↪  DELETE CASCADE
);

CREATE TABLE TELEFONO_O (
    numero VARCHAR(15) NOT NULL,
    operatore CHAR(16) NOT NULL,
    PRIMARY KEY (numero),
    FOREIGN KEY (operatore) REFERENCES OPERATORE(codice_fiscale) ON UPDATE CASCADE ON
    ↪  DELETE CASCADE
);

CREATE TABLE ORDINE (
    -- Unique code
    codice SERIAL PRIMARY KEY,
```

```
    -- Old Primary Key
    timestamp TIMESTAMP NOT NULL,
    artista VARCHAR(255) NOT NULL,

    -- Old Primary Key Uniqueness Maintained
    CONSTRAINT unique_ordine UNIQUE (timestamp, artista),
    FOREIGN KEY (artista) REFERENCES ARTISTA(nome_arte) ON UPDATE CASCADE ON DELETE
    ↪  RESTRICT,

    -- Other
    annullato BOOLEAN NOT NULL,
    operatore CHAR(16) NOT NULL,
    FOREIGN KEY (operatore) REFERENCES OPERATORE(codice_fiscale) ON UPDATE CASCADE ON
    ↪  DELETE RESTRICT
);

CREATE TABLE METODO (
    nome VARCHAR(255) PRIMARY KEY
);

CREATE TABLE PAGAMENTO (

    ordine INTEGER PRIMARY KEY,
    FOREIGN KEY (ordine) REFERENCES ORDINE(codice),

    -- Other
    stato VARCHAR(50) NOT NULL CHECK (stato IN ('Da pagare', 'Pagato')), -- Da pagare,
    ↪  Pagato
    costo_totale DECIMAL(10, 2),
    metodo VARCHAR(255),
    FOREIGN KEY (metodo) REFERENCES METODO(nome) ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT costo_totale_maggiore CHECK(costo_totale > 0)

);

CREATE TABLE TIPOLOGIA (
    nome VARCHAR(255) PRIMARY KEY,
    valore DECIMAL(10, 2) NOT NULL,
    n_giorni INTEGER NOT NULL, -- 1 giorno, 7 giorni, 30 giorni
    CONSTRAINT check_nome_tipologia CHECK (nome IN ('Giornaliera', 'Settimanale',
    ↪  'Mensile')),
    CONSTRAINT check_valore_positivo CHECK (valore > 0),
    CONSTRAINT check_n_giorni CHECK (n_giorni = 1 OR n_giorni = 7 OR n_giorni = 30)
);

CREATE TABLE PACCHETTO (
    ordine INTEGER PRIMARY KEY,
    FOREIGN KEY (ordine) REFERENCES ORDINE(codice) ON UPDATE CASCADE ON DELETE CASCADE,

    tipologia VARCHAR(255) NOT NULL,
```

```
    FOREIGN KEY (tipologia) REFERENCES TIPOLOGIA(nome) ON UPDATE CASCADE ON DELETE
    ↪ RESTRICT,

    -- Other
    n_giorni_prenotati_totali INTEGER
);

CREATE TABLE ORARIO (
    ordine INTEGER PRIMARY KEY,
    FOREIGN KEY (ordine) REFERENCES ORDINE(codice) ON UPDATE CASCADE ON DELETE CASCADE,

    n_ore_prenotate_totali INTEGER,
    valore DECIMAL(10, 2),
    CONSTRAINT valore_maggiore CHECK (valore > 0)
);

CREATE TABLE SALA (
    piano INTEGER,
    numero INTEGER,
    PRIMARY KEY (piano, numero)
);

CREATE TABLE PRENOTAZIONE (
    codice SERIAL PRIMARY KEY,
    annullata BOOLEAN NOT NULL,
    giorno DATE NOT NULL,
    tipo BOOLEAN NOT NULL,
    pacchetto INTEGER,
    sala_piano INTEGER NOT NULL,
    sala_numero INTEGER NOT NULL,
    FOREIGN KEY (pacchetto) REFERENCES PACCHETTO(ordine) ON UPDATE CASCADE ON DELETE
    ↪ RESTRICT,
    FOREIGN KEY (sala_piano, sala_numero) REFERENCES SALA(piano, numero) ON UPDATE
    ↪ CASCADE ON DELETE RESTRICT
);

CREATE TABLE ORARIA (
    prenotazione INTEGER PRIMARY KEY,
    orario INTEGER UNIQUE NOT NULL, --one to one
    FOREIGN KEY (prenotazione) REFERENCES PRENOTAZIONE(codice) ON UPDATE CASCADE ON
    ↪ DELETE CASCADE,
    FOREIGN KEY (orario) REFERENCES ORARIO(ordine) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE FASCIA_ORARIA (
    oraria INTEGER NOT NULL,
    orario_inizio TIME,
    orario_fine TIME,
    PRIMARY KEY (oraria, orario_inizio),
    FOREIGN KEY (oraria) REFERENCES ORARIA(prenotazione) ON UPDATE CASCADE ON DELETE
    ↪ CASCADE,
```

```
    CONSTRAINT orario_fine_maggiore CHECK(orario_fine>orario_inizio),
    CONSTRAINT check_orario CHECK (
        (orario_inizio >= '08:00' AND orario_fine <= '12:00')
        OR (orario_inizio >= '14:00' AND orario_fine <= '23:00'))
);

CREATE TABLE TIPO_TECNICO (
    nome VARCHAR(64) PRIMARY KEY,
    CHECK (nome IN ('Fonico', 'Tecnico del Suono', 'Tecnico del Suono_AND_Fonico'))
);

CREATE TABLE TECNICO (
    codice_fiscale CHAR(16) PRIMARY KEY,
    sala_piano INT NOT NULL,
    sala_numero INT NOT NULL,
    tipo_tecnico VARCHAR(64) NOT NULL,
    nome VARCHAR(255) NOT NULL,
    cognome VARCHAR(255) NOT NULL,
    data_di_nascita DATE NOT NULL,
    data_di_assunzione DATE NOT NULL,
    iban VARCHAR(34),
    FOREIGN KEY (sala_piano, sala_numero) REFERENCES SALA(piano, numero) ON UPDATE
    ↪  CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (tipo_tecnico) REFERENCES TIPO_TECNICO(nome) ON UPDATE CASCADE ON DELETE
    ↪  RESTRICT
);

CREATE TABLE EMAIL_T (
    email VARCHAR(255) PRIMARY KEY,
    tecnico CHAR(16) NOT NULL,
    FOREIGN KEY (tecnico) REFERENCES TECNICO(codice_fiscale) ON UPDATE CASCADE ON DELETE
    ↪  CASCADE
);

CREATE TABLE TELEFONO_T (
    numero VARCHAR(15) PRIMARY KEY,
    tecnico CHAR(16) NOT NULL,
    FOREIGN KEY (tecnico) REFERENCES TECNICO(codice_fiscale) ON UPDATE CASCADE ON DELETE
    ↪  CASCADE
);

CREATE TABLE LAVORA_A (
    tecnico CHAR(16),
    canzone INTEGER,
    PRIMARY KEY (tecnico, canzone),
    FOREIGN KEY (tecnico) REFERENCES TECNICO(codice_fiscale) ON UPDATE CASCADE ON DELETE
    ↪  RESTRICT,
    FOREIGN KEY (canzone) REFERENCES CANZONE(codice) ON UPDATE CASCADE ON DELETE CASCADE
);
```

```sql
CREATE TABLE ATTREZZATURA (
    codice SERIAL PRIMARY KEY,
    modello VARCHAR(64),
    costruttore VARCHAR(255)
);

CREATE TABLE AVERE (
    piano INTEGER NOT NULL,
    numero INTEGER NOT NULL,
    attrezzo INTEGER NOT NULL,

    PRIMARY KEY (piano, numero, attrezzo),
    FOREIGN KEY (attrezzo) REFERENCES ATTREZZATURA(codice) ON UPDATE CASCADE ON DELETE
    ↪ RESTRICT,
    FOREIGN KEY (piano, numero) REFERENCES SALA(piano, numero) ON UPDATE CASCADE ON
    ↪ DELETE RESTRICT
);
```

2 Funzioni

2.1 Introduzione alle Funzioni

Sono descritte in seguito una serie di funzioni in linguaggio *PL/pgSQL*, le quali sono utilizzate da diverse procedure al fine di automatizzare la gestione dei dati.

2.2 Implementazione delle Funzioni

```
/* CALCOLA L'ETA DELL'ARTISTA
 * La funzione calcola l'età dell'artista del quale è stato fornito il codice fiscale.
 *
 * INPUT:   cd      INT
 * OUTPUT:  età     DECIMAL
 */
CREATE OR REPLACE FUNCTION CalcolaEtaArtista(cd VARCHAR(16)) RETURNS INT LANGUAGE plpgsql
↪  AS $$
DECLARE
    data_nascita DATE;
BEGIN
    SELECT data_di_nascita INTO data_nascita FROM SOLISTA WHERE codice_fiscale = cd;
        return (date_part('year', CURRENT_DATE) - date_part('year', data_nascita));
END
$$;

-- Chiamata alla Funzione
SELECT CalcolaEtaArtista('VCTFNC90A01H501X');
```

```
/* CONTROLLA IL TIPO DI UN ORDINE
 * La funzione controlla il tipo di un ordine.
 * Un ordine può essere di tipo Orario o di tipo Giornaliero.
 *
 * INPUT:   ordine_id   INT
 * OUTPUT:
 *          TRUE        BOOL        :se orario
 *          FALSE       BOOL        :se giornaliero
 */
CREATE OR REPLACE FUNCTION ControllaTipoOrdine(ordine_id INT) RETURNS boolean LANGUAGE
↪  plpgsql AS $$
DECLARE
    tupla INT;
BEGIN
        SELECT codice INTO tupla FROM ordine, orario WHERE codice = ordine_id AND codice
        ↪  = ordine;
    IF (tupla) IS NOT NULL THEN RETURN TRUE; ELSE RETURN FALSE; END IF;
END
$$;
```

```sql
-- Chiamata alla Funzione
SELECT ControllaTipoOrdine(1);
```

```sql
/* CALCOLA IL COSTO TOTALE DI UN ORDINE
 * La funzione calcola il costo totale dato il codice univoco di un ordine.
 * Il calcolo del costo totale cambia a seconda della tipologia dell'ordine.
 *
 * INPUT:   ordine_id       INTEGER
 * OUTPUT:  costo_totale    DECIMAL(10, 2)
 */
CREATE OR REPLACE FUNCTION CalcolaCostoTotale(ordine_id INT) RETURNS DECIMAL(10, 2)
↪  LANGUAGE plpgsql AS $$
DECLARE
    costo_totale DECIMAL(10, 2);
BEGIN
    -- controllo la tipologia dell'ordine
    IF ControllaTipoOrdine(ordine_id)
    THEN -- se di tipo Orario
        SELECT n_ore_prenotate_totali * valore INTO costo_totale
        FROM ordine, orario WHERE codice = ordine_id AND codice = ordine;
    ELSE -- se di tipo Pacchetto
        SELECT t.valore INTO costo_totale FROM ordine AS o, pacchetto AS p, tipologia AS t
        WHERE o.codice = ordine_id AND o.codice = p.ordine AND p.tipologia = t.nome;
    END IF;
    RETURN costo_totale;
END
$$;

-- Chiamata alla Funzione
SELECT CalcolaCostoTotale(6);
```

```sql
/* CONTA NUMERO DI CANZONI
 * La funzione conta il numero di canzoni della produzione di cui è fornito il codice.
 *
 * INPUT:   codice_produzione   INTEGER
 * OUTPUT:  numero_canzoni      INTEGER
 */
CREATE OR REPLACE FUNCTION conta_canzoni_di_una_produzione(codice_produzione INTEGER)
↪  RETURNS INTEGER AS $$
DECLARE
    numero_canzoni INTEGER;
BEGIN
    -- Conta il numero di canzoni per la produzione specificata
    SELECT COUNT(*) INTO numero_canzoni
    FROM canzone WHERE produzione = codice_produzione;

    RETURN numero_canzoni;
END;
```

```
$$ LANGUAGE plpgsql;

-- Chiamata alla Funzione
SELECT conta_canzoni_di_una_produzione(1);
```

```
/* CALCOLA LUNGHEZZA CANZONI DI UNA PRODUZIONE
 * Dato il codice di una produzione, la funzione calcola la
 * lunghezza media in secondi delle sue canzoni.
 *
 * INPUT:   codice_produzione   INTEGER
 * OUTPUT:  lunghezza_in_secondi_media     NUMERIC
 */
CREATE OR REPLACE FUNCTION
↪  calcola_lunghezza_media_canzoni_di_una_produzione(codice_produzione INTEGER) RETURNS
↪  NUMERIC AS $$
DECLARE
    lunghezza_in_secondi_totale INTEGER;
    numero_canzoni INTEGER;
    lunghezza_in_secondi_media NUMERIC;
BEGIN
    -- Calcola la somma delle lunghezze delle canzoni per la produzione specificata
    SELECT SUM(lunghezza_in_secondi) INTO lunghezza_in_secondi_totale
    FROM canzone
    WHERE produzione = codice_produzione;

    SELECT conta_canzoni_di_una_produzione(codice_produzione) INTO numero_canzoni;
    -- Calcola la lunghezza media delle canzoni
    IF numero_canzoni > 0 THEN
        lunghezza_in_secondi_media := lunghezza_in_secondi_totale / numero_canzoni;
    ELSE
        -- Se non ci sono canzoni, la lunghezza media è 0
        lunghezza_in_secondi_media := 0;
    END IF;

    RETURN lunghezza_in_secondi_media;
END;
$$ LANGUAGE plpgsql;

-- Chiamata alla Funzione
SELECT calcola_lunghezza_media_canzoni_di_una_produzione(1);
```

3 Procedure

3.1 Introduzione alle Procedure

Di seguito vengono descritte delle procedure in linguaggio *PL/pgSQL*, progettate per automatizzare la gestione dei dati. Ogni procedura viene dettagliata con il suo scopo e utilizzo specifico.

3.2 Implementazione delle Procedure

```
/* AGGIORNA IL COSTO TOTALE
 * Ricalcola ed aggiorna il costo totale di un ordine.
 * Da eseguire una sola volta dopo la creazione di un ordine di tipo "Pacchetto".
 * Da eseguire tutte le volte che si inseriscono delle nuove "Prenotazioni
 ↪  orarie"/"Fasce orarie".
 *
 * INPUT:   ordine_id   INT
 */
CREATE OR REPLACE PROCEDURE AggiornaCostoOrdine(ordine_id INT) LANGUAGE plpgsql AS $$
BEGIN
    UPDATE PAGAMENTO
    SET costo_totale = CalcolaCostoTotale(ordine_id)
    WHERE ordine = ordine_id;
EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nell aggiornamento del costo totale di un ordine: %',
        ↪  SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla Procedura
CALL AggiornaCostoOrdine(1);
```

```
/* AGGIUNGE UN ARTISTA
 * La funzione crea il record di un artista con i parametri passati come argomento.
 *
 * INPUT:   nome_arte              VARCHAR(50)
 * INPUT:   data_registrazione     DATE
 */
CREATE OR REPLACE PROCEDURE AggiungiArtista(nome_arte VARCHAR(50), data_registrazione
↪  DATE) LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO ARTISTA (nome_arte, data_di_registrazione)
    VALUES (nome_arte, data_registrazione);
END
$$;

-- Chiamata alla Procedura
CALL AggiungiArtista('Tremore', '2024-06-08');
```

```sql
/* CREA UN NUOVO ORDINE DI TIPO PACCHETTO
 * La procedura crea un ordine di tipo pacchetto.
 *
 * INPUT:   operatore_codice_fiscale    VARCHAR(16)
 * INPUT:   artista_nome_arte           VARCHAR(255)
 * INPUT:   pacchetto_nome              VARCHAR(255)
 */
CREATE OR REPLACE PROCEDURE CreaOrdinePacchetto(
    operatore_codice_fiscale VARCHAR(16),
    artista_nome_arte VARCHAR(255),
    pacchetto_nome VARCHAR(255)) LANGUAGE plpgsql AS $$
DECLARE
    ordine_id INT;
BEGIN
    -- inserimento ordine
        INSERT INTO ORDINE (timestamp, artista, annullato, operatore)
        VALUES (CURRENT_TIMESTAMP, artista_nome_arte, FALSE, operatore_codice_fiscale)
        RETURNING codice INTO ordine_id;

    -- collegamento pacchetto
    INSERT INTO PACCHETTO (ordine, tipologia, n_giorni_prenotati_totali)
    VALUES (ordine_id, pacchetto_nome, 0);

    -- inserimento pagamaneto
    INSERT INTO PAGAMENTO (ordine, stato, costo_totale, metodo)
    VALUES(ordine_id, 'Da pagare', CalcolaCostoTotale(ordine_id), NULL);
EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nella creazione di un ordine di tipo pacchetto: %',
        ↪  SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla Procedura
CALL CreaOrdinePacchetto('OPRABC90A01H501X', 'TrioUVW', 'Mensile');
```

```sql
/* CREA UNA PRENOTAZIONE DI TIPO GIORNALIERA
 * Procedura che inserisce una nuova prenotazione associandola al
 * ordine di tipo pacchetto il cui id è dato come argomento.
 *
 * INPUT:   pacchetto_id                INT
 * INPUT:   giorno                      DATE
 * INPUT:   sala_piano                  INT
 * INPUT:   sala_numero                 INT
 */
CREATE OR REPLACE PROCEDURE CreaPrenotazioneGiornaliera(
    pacchetto_id INT,
    giorno DATE,
```

```
    sala_piano INT,
    sala_numero INT
)LANGUAGE plpgsql AS $$
BEGIN
    -- inserimento prenotazione
    INSERT INTO PRENOTAZIONE (annullata, giorno, tipo, pacchetto, sala_piano, sala_numero)
    VALUES (FALSE, giorno, TRUE, pacchetto_id, sala_piano, sala_numero);

    -- aggiornamento contatore giorni prenotati
    UPDATE PACCHETTO
    SET n_giorni_prenotati_totali = n_giorni_prenotati_totali + 1
    WHERE ordine = pacchetto_id;
EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nella creazione di una prenotazione giornaliera: %',
        ↪  SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla Procedura
CALL CreaPrenotazioneGiornaliera(1, '2023-01-10', 2, 2); -- Funzionante
CALL CreaPrenotazioneGiornaliera(1, '2022-01-01', 2, 2); -- Eccezione: Indietro nel tempo
CALL CreaPrenotazioneGiornaliera(1, '2023-05-01', 2, 2); -- Eccezione: Più di 90 giorni
```

```
/* CREA UN ORDINE DI TIPO ORARIO E RELATIVA PRENOTAZIONE
 * La procedura produce un ordine di tipo orario e la rispettiva prenotazione.
 *
 * INPUT:   operatore_codice_fiscale    VARCHAR(16)
 * INPUT:   artista_nome_arte           VARCHAR(255)
 * INPUT:   costo_ora                   DECIMAL(10, 2)
 * INPUT:   giorno                      DATE
 * INPUT:   orari_inizio                TIME[]
 * INPUT:   orari_fine                  TIME[]
 * INPUT:   sala_piano                  INT
 * INPUT:   sala_numero                 INT
 */
CREATE OR REPLACE PROCEDURE CreaOrdineEPrenotazioneOrarie(
    operatore_codice_fiscale VARCHAR(16),
    artista_nome_arte VARCHAR(255),
    costo_ora DECIMAL(10, 2),
    giorno DATE,
    orari_inizio TIME[],
    orari_fine TIME[],
    sala_piano INT,
    sala_numero INT) LANGUAGE plpgsql AS $$
DECLARE
    ordine_id INT;
    prenotazione_id INT;
```

```
    ore_prenotate_totali INT;
BEGIN
    ordine_id := CreaOrdineOrario(
        operatore_codice_fiscale,
        artista_nome_arte,
        costo_ora
    );
    CALL CreaPrenotazioneOraria(
        ordine_id, giorno,
        orari_inizio,
        orari_fine,
        sala_piano,
        sala_numero
    );
EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nella creazione di un ordine e di una prenotazione
        ↪  oraria: %', SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla Procedura
CALL CreaOrdineEPrenotazioneOrarie(
    'OPRABC90A01H501X', 'BandABC', 30,
    '2023-02-03',
    '{08:00, 14:00}'::time[],
    '{12:00, 18:00}'::time[],
    2, 2
);
```

```
/* CREA UN NUOVO ORDINE DI TIPO ORARIO
 * La funzione produce un ordine di tipo orario. E restituisce il suo id.
 *
 * INPUT:   operatore_codice_fiscale   VARCHAR(16)
 * INPUT:   artista_nome_arte          VARCHAR(255)
 * INPUT:   costo_ora                  DECIMAL(10, 2)
 *
 * OUTPUT:  ordine_id                  INT
 */
CREATE OR REPLACE FUNCTION CreaOrdineOrario(
    operatore_codice_fiscale VARCHAR(16),
    artista_nome_arte VARCHAR(255),
    costo_ora DECIMAL(10, 2)) RETURNS INT LANGUAGE plpgsql AS $$
DECLARE
    ordine_id INT;
BEGIN
    -- inserimento ordine
        INSERT INTO ORDINE (timestamp, artista, annullato, operatore)
```

```
        VALUES (CURRENT_TIMESTAMP, artista_nome_arte, FALSE, operatore_codice_fiscale)
        RETURNING codice INTO ordine_id;

    -- collegamento orario
    INSERT INTO ORARIO (ordine, n_ore_prenotate_totali, valore)
    VALUES (ordine_id, 0, costo_ora);

    -- inserimento pagamaneto
    INSERT INTO PAGAMENTO (ordine, stato, costo_totale, metodo)
    VALUES(ordine_id, 'Da pagare', NULL, NULL);

    RETURN ordine_id;
EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nella creazione di un ordine orario: %', SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla Procedura
SELECT CreaOrdineOrario(
    'OPRABC90A01H501X', 'BandDEF', 30
);
```

```
/* CREA PRENOTAZIONE PER ORDINE ORARIO
 * La procedura produce la prenotazione per un ordine orario.
 *
 * INPUT:   ordine_id       INT
 * INPUT:   giorno          DATE
 * INPUT:   orari_inizio    TIME[]
 * INPUT:   orari_fine      TIME[]
 * INPUT:   sala_piano      INT
 * INPUT:   sala_numero     INT
 */
CREATE OR REPLACE PROCEDURE CreaPrenotazioneOraria(
    ordine_id INT,
    giorno DATE,
    orari_inizio TIME[],
    orari_fine TIME[],
    sala_piano INT,
    sala_numero INT) LANGUAGE plpgsql AS $$
DECLARE
    prenotazione_id INT;
    ore_prenotate_parziali INT;
    ore_prenotate_totali INT;
BEGIN
    -- Controllo lunghezze array di orari
    IF array_length(orari_inizio, 1) != array_length(orari_fine, 1) THEN
        RAISE EXCEPTION 'Gli array di orari di inizio e fine devono avere la stessa
        ↪  lunghezza.';
```

```
    END IF;

    -- Controllo lunghezze array di orari
    IF array_length(orari_inizio, 1) IS NULL OR array_length(orari_fine, 1) IS NULL THEN
        RAISE EXCEPTION 'Gli array di orari di inizio e fine devono contenere almento una
        ↪  fascia oraria.';
    END IF;

    -- inserimento prenotazione
    INSERT INTO PRENOTAZIONE (annullata, giorno, tipo, pacchetto, sala_piano, sala_numero)
    VALUES (FALSE, giorno, FALSE, NULL, sala_piano, sala_numero)
    RETURNING codice INTO prenotazione_id;

    -- inserimento oraria
    INSERT INTO ORARIA (prenotazione, orario)
    VALUES (prenotazione_id, ordine_id);

    -- inserimento fasce orarie
    ore_prenotate_totali := 0;
    FOR i IN 1..array_length(orari_inizio, 1) LOOP
        INSERT INTO FASCIA_ORARIA (oraria, orario_inizio, orario_fine)
        VALUES (prenotazione_id, orari_inizio[i], orari_fine[i]);
        -- numero di ore della fascia oraria
        SELECT EXTRACT(HOUR FROM orari_fine[i] - orari_inizio[i]) INTO
        ↪  ore_prenotate_parziali;
        ore_prenotate_totali := ore_prenotate_totali + ore_prenotate_parziali;
    END LOOP;

    -- aggiornamento numero totale di ore
    UPDATE orario AS o
    SET n_ore_prenotate_totali = ore_prenotate_totali
    WHERE o.ordine = ordine_id;

    -- aggiornamento del costo del pagamento
    UPDATE PAGAMENTO AS p
    SET costo_totale = CalcolaCostoTotale(ordine_id)
    WHERE p.ordine = ordine_id;
EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nella creazione di una prenotazione oraria: %', SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla Procedura
CALL CreaPrenotazioneOraria(
    10, '2023-02-03', -- ordine_id = 10
    '{08:00, 14:00}'::time[],
    '{10:00, 18:00}'::time[],
    2, 2
```

```
);
```

```
/* CREA UN SOLISTA
 * Crea un nuovo artista e un solista associato nel database.
 *
 * INPUT:   nome_arte            VARCHAR(255)
 * INPUT:   data_di_registrazione DATE
 * INPUT:   codice_fiscale       CHAR(16)
 * INPUT:   nome                 VARCHAR(255)
 * INPUT:   cognome              VARCHAR(255)
 * INPUT:   data_di_nascita      DATE
 * INPUT:   email                VARCHAR(255)
 * INPUT:   telefono             CHAR(15)
 */
CREATE OR REPLACE PROCEDURE CreaArtistaSolista(
    nome_arte VARCHAR(255),
    data_di_registrazione DATE,
    codice_fiscale CHAR(16),
    nome VARCHAR(255),
    cognome VARCHAR(255),
    data_di_nascita DATE,
    email VARCHAR(255),
    telefono VARCHAR(15)
)
LANGUAGE plpgsql AS $$
BEGIN
    -- Inserimento nella tabella Artista
    INSERT INTO Artista (nome_arte, data_di_registrazione)
    VALUES (nome_arte, data_di_registrazione);

    -- Inserimento nella tabella Solista
    INSERT INTO Solista (artista, codice_fiscale, nome, cognome, data_di_nascita)
    VALUES (nome_arte, codice_fiscale, nome, cognome, data_di_nascita);

    -- Inserimento nella tabella EMAIL_A
    INSERT INTO EMAIL_A (email, artista)
    VALUES (email, nome_arte);

    -- Inserimento nella tabella TELEFONO_A
    INSERT INTO TELEFONO_A (numero, artista)
    VALUES (telefono, nome_arte);

EXCEPTION
    WHEN OTHERS THEN -- Gestione degli errori
        RAISE EXCEPTION 'Errore nella creazione dell artista e del solista: %', SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;
```

```sql
-- Chiamata alla Procedura
CALL CreaArtistaSolista(
    'PopStar',
    '2024-06-08',
    'PSMRTN90B05C351K',
    'Giulia',
    'Bianchi',
    '1990-02-05',
    'giulia.bianchi@example.com',
    '098762222'
);
```

```sql
/* CREA UN ARTISTA E UN GRUPPO
 * Crea un nuovo artista e un gruppo associato nel database.
 *
 * INPUT:   nome_arte            VARCHAR(255)
 * INPUT:   data_di_registrazione DATE
 * INPUT:   email                VARCHAR(255)
 * INPUT:   telefono             CHAR(15)
 * INPUT:   data_formazione      DATE
 */
CREATE OR REPLACE PROCEDURE CreaArtistaGruppo(
    nome_arte VARCHAR(255),
    data_di_registrazione DATE,
    email VARCHAR(255),
    telefono VARCHAR(15),
    data_formazione DATE
)
LANGUAGE plpgsql AS $$
DECLARE
BEGIN
    -- Inserimento nella tabella Artista
    INSERT INTO Artista (nome_arte, data_di_registrazione)
    VALUES (nome_arte, data_di_registrazione);

    -- Inserimento nella tabella EMAIL_A
    INSERT INTO EMAIL_A (email, artista)
    VALUES (email, nome_arte);

    -- Inserimento nella tabella TELEFONO_A
    INSERT INTO TELEFONO_A (numero, artista)
    VALUES (telefono, nome_arte);

    -- Inserimento nella tabella Gruppo
    INSERT INTO Gruppo (artista, data_formazione)
    VALUES (nome_arte, data_formazione);

EXCEPTION
    -- Gestione degli errori
```

```
    WHEN OTHERS THEN
        -- Solleva l'eccezione per visualizzare l'errore
        RAISE EXCEPTION 'Errore nella creazione dell artista e del gruppo: %', SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore

END
$$;

-- Chiamata alla procedura
CALL CreaArtistaGruppo(
    'Artista1234',
    '2024-06-09',
    'artista1234@email.com',
    '1234567333',
    '2024-01-01'
);
```

```
/* CREA PARTECIPAZIONE SOLISTA GRUPPO
 * Gestisce la partecipazione di un solista a un gruppo nel database.
 *
 * INPUT:   nome_arte_solista       VARCHAR(255)
 * INPUT:   gruppo_nuovo            VARCHAR(255)
 * INPUT:   data_adesione_corrente  DATE
 */
CREATE OR REPLACE PROCEDURE CreaPartecipazioneSolistaGruppo(
    nome_arte_solista VARCHAR(255),
    gruppo_nuovo VARCHAR(255),
    data_adesione_corrente DATE
)
LANGUAGE plpgsql AS $$
DECLARE
    gruppo_vecchio VARCHAR(255);
BEGIN
    -- Cerca il gruppo attuale del solista
    SELECT gruppo INTO gruppo_vecchio
    FROM SOLISTA
    WHERE artista = nome_arte_solista;

    -- Se il solista non è in nessun gruppo, il gruppo diventa il suo corrente
    IF gruppo_vecchio IS NULL THEN
        UPDATE SOLISTA
        SET gruppo = gruppo_nuovo, data_adesione = data_adesione_corrente
        WHERE artista = nome_arte_solista;
    ELSE
        -- Se il solista è già nel gruppo specificato, non fare nulla
        IF gruppo_vecchio = gruppo_nuovo THEN -- partecipazione corrente
            RAISE NOTICE 'Il solista è già nel gruppo specificato.';
        ELSE
            -- Se il solista è già in un gruppo, aggiorna la partecipazione passata e
            ↪  crea una nuova partecipazione
```

```
            INSERT INTO PARTECIPAZIONE_PASSATA (gruppo, solista, data_adesione,
            ↪  data_fine_adesione)
            VALUES (gruppo_vecchio, nome_arte_solista, data_adesione_corrente,
            ↪  CURRENT_DATE);

            UPDATE SOLISTA
            SET gruppo = gruppo_nuovo, data_adesione = data_adesione_corrente
            WHERE artista = nome_arte_solista;
        END IF;
    END IF;

    EXCEPTION
        -- Gestione degli errori
        WHEN OTHERS THEN
            -- Solleva l'eccezione per visualizzare l'errore
            RAISE EXCEPTION 'Errore nella creazione della partecipazione del solista al
            ↪  gruppo: %', SQLERRM;
            ROLLBACK;    -- Annulla la transazione in caso di errore
END
$$;

-- Chiamata alla procedura
CALL CreaPartecipazioneSolistaGruppo('PopStar', 'Artista1234', '2024-06-08'); --
↪  Funzionante
CALL CreaPartecipazioneSolistaGruppo('PopStar', 'Artista1234', '2024-06-08'); --
↪  Notifica: Il solista è già nel gruppo specificato
CALL CreaPartecipazioneSolistaGruppo('PopStar', 'BandABC',     '2024-06-08'); --
↪  Funzionante
CALL CreaPartecipazioneSolistaGruppo('PopStar', 'Group123',    '2024-06-08'); --
↪  Funzionante
```

```
/*
 * AGGIUNGI CANZONE
 * Aggiunge una nuova canzone al database e gestisce la relazione "lavora_a" con solisti.
 *
 * INPUT:   titolo                  VARCHAR(255)    - Titolo della canzone
 * INPUT:   produzione_id           INT             - Id della produzione
 * INPUT:   testo                   TEXT            - Testo della canzone
 * INPUT:   data_di_registrazione   DATE            - Data di registrazione della
 ↪  canzone
 * INPUT:   lunghezza_in_secondi    INT             - Lunghezza in secondi della
 ↪  canzone
 * INPUT:   nome_del_file           VARCHAR(255)    - Nome del file della canzone
 * INPUT:   percorso_di_sistema     VARCHAR(255)    - Percorso di sistema del file
 ↪  della canzone
 * INPUT:   estensione              VARCHAR(10)     - Estensione del file della
 ↪  canzone
 * INPUT:   solisti_nome_arte       VARCHAR[]       - Array di nomi d'arte dei
 ↪  solisti che partecipano alla creazione della canzone
```

```
 * INPUT:   codice_fiscale_tecnico      CHAR(16)        - Codice fiscale del tecnico che
 ↪  lavora sulla canzone
 */
CREATE OR REPLACE PROCEDURE AggiungiCanzoneEPartecipazioni(
    titolo VARCHAR(255),
    produzione_id INT,
    testo TEXT,
    data_di_registrazione DATE,
    lunghezza_in_secondi INT,
    nome_del_file VARCHAR(255),
    percorso_di_sistema VARCHAR(255),
    estensione VARCHAR(10),
    solisti_nome_arte VARCHAR[],
    codice_fiscale_tecnico CHAR(16)
)
LANGUAGE plpgsql AS $$
declare
    codice_canzone INT;
    solista_nome VARCHAR(255);
BEGIN
    -- Inserisce la canzone
    INSERT INTO CANZONE (titolo, produzione, testo, data_di_registrazione,
    ↪  lunghezza_in_secondi, nome_del_file, percorso_di_sistema, estensione)
    VALUES (titolo, produzione_id, testo, data_di_registrazione, lunghezza_in_secondi,
    ↪  nome_del_file, percorso_di_sistema, estensione)
    RETURNING codice INTO codice_canzone;

    -- Inserisce la partecipazione dei solisti
    FOREACH solista_nome IN ARRAY solisti_nome_arte LOOP

        -- Inserisce la partecipazione solo se il solista esiste nella tabella SOLISTA
        -- Cerca l'artista (solista) per nome d'arte
        PERFORM artista
        FROM SOLISTA
        WHERE artista = solista_nome;

        -- Se l'artista (solista) esiste, inserisce la partecipazione nella canzone
        IF FOUND THEN

            -- Verifica se il solista è già presente nella canzone
            INSERT INTO PARTECIPAZIONE (solista, canzone)
            VALUES (solista_nome, codice_canzone);

        ELSE
            -- Solleva un avviso se l'artista (solista) non esiste
            RAISE NOTICE 'Solista non trovato con nome d arte %, non è stato possibile
            ↪  aggiungere la partecipazione.', solista_nome;
        END IF;
    END LOOP;
```

```
    -- Aggiunge il tecnico alla tabella Lavora_a
    INSERT INTO LAVORA_A (tecnico, canzone)
    VALUES (codice_fiscale_tecnico, codice_canzone);

EXCEPTION
    -- Gestione degli errori
    WHEN OTHERS THEN
        -- Solleva l'eccezione per visualizzare l'errore
        RAISE EXCEPTION 'Errore durante l aggiunta della canzone: %', SQLERRM;
        ROLLBACK;    -- Annulla la transazione in caso di errore
END;
$$;

-- Chiamata alla procedura
CALL AggiungiCanzoneEPartecipazioni(
    'Titolo della Canzone',
    2,
    'Testo della canzone',
    '2024-06-09',
    300, -- Lunghezza in secondi
    'nome_file.mp3',
    '/percorso/di/sistema',
    'mp3', -- Estensione del file
    ARRAY['SoloPQR', 'PopStar'], -- Array di nomi d'arte dei solisti
    'TCNAUD85M01H501Z' -- Codice fiscale del tecnico
);
```

4 Trigger

4.1 Introduzione ai Trigger

Di seguito vengono descritti una serie di vincoli implementati nel database. Questi vincoli, realizzati attraverso l'uso di trigger e funzioni *PL/pgSQL*, automatizzano il controllo delle condizioni non esprimibili tramite lo schema fisico del database, prevenendo l'inserimento o l'aggiornamento di dati non validi.

4.2 Implementazione dei Triggers

4.2.1 Trigger derivati da Vincoli

```
/* RV1: Non è necessario utilizzare i giorni di un pacchetto tutti in fila ma possono
↪  essere sfruttati nell arco di 90 giorni.
 * IMPLEMENTAZIONE: Impedire la creazione di una prenotazione dopo 90 giorni dall
 ↪  effettuazione dell ordine.
 * Impedire inoltre di inserire una prenotazione in una data antecedente alla data di
 ↪  effettuazione dell ordine.
 */
CREATE OR REPLACE FUNCTION check_ordine_orario()
RETURNS TRIGGER AS $$
DECLARE
    ordine_timestamp TIMESTAMP;
    differenza_giorni INTEGER;
BEGIN
    IF NEW.tipo THEN
        -- Otteniamo la data di effettuazione dell'ordine
        SELECT timestamp INTO ordine_timestamp
        FROM ordine WHERE codice = NEW.PACCHETTO;

        -- Calcoliamo la differenza dei giorni
        differenza_giorni := (NEW.giorno::date - ordine_timestamp::date);

        -- Controlliamo che la differenza non sia negativa
        IF differenza_giorni < 0 THEN
            RAISE EXCEPTION 'Non è possibile creare una prenotazione che vada indietro
            ↪  nel tempo.';
        END IF;

        -- Controlliamo se la differenza è maggiore di 90 giorni
        IF differenza_giorni > 90 THEN
            RAISE EXCEPTION 'Non è possibile creare una prenotazione dopo 90 giorni dall
            ↪  effettuazione dell ordine.';
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER RV1
BEFORE INSERT ON prenotazione
```

```
FOR EACH ROW WHEN (NEW.tipo = TRUE)
EXECUTE FUNCTION check_ordine_orario();
```

```
/* RV2: Una produzione una volta pubblicata diventa immutabile quindi non è più possibile
↪  aggiungere canzoni.
 * IMPLEMENTAZIONE: Questo vincolo può essere implementato con un trigger che si attiva
 ↪  prima dell'inserimento
 * di una produzione.
 */
CREATE OR REPLACE FUNCTION controlla_produzione_immutabile() RETURNS TRIGGER AS $$
BEGIN
    PERFORM artista
    FROM PRODUZIONE
    WHERE codice = NEW.produzione AND stato = 'Pubblicazione';

    -- Se la produzione interessata è già stata pubblicata
    IF FOUND THEN
        RAISE EXCEPTION 'Impossibile aggiungere canzoni a una produzione in stato di
        ↪  Pubblicazione (codice produzione: %).', NEW.produzione;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER RV2
BEFORE INSERT ON CANZONE
FOR EACH ROW
EXECUTE FUNCTION controlla_produzione_immutabile();
```

```
/* RV3: L'entità Singolo comprende da una a tre canzoni, l'entità Extended Play comprende
↪  un massimo di 5 canzoni e
 * l'entità Album non ha un limite al numero di canzoni fintanto che la durata
 ↪  complessiva stia sotto l'ora.
 */
CREATE OR REPLACE FUNCTION check_tipo_produzione() RETURNS TRIGGER AS $$
DECLARE
    tipo_produzione VARCHAR(25);
    numero_canzoni INT;
    lunghezza_totale_secondi INT;
BEGIN

    -- Selezioniamo la tipologia della produzione (tra Album / EP / Singolo) in cui la
    ↪  canzone compare
    SELECT p.tipo_produzione INTO tipo_produzione FROM produzione AS p WHERE codice =
    ↪  NEW.produzione;

    IF tipo_produzione = 'Album' THEN
```

```
        -- Controlliamo se con l'aggiunta di una nuova canzone sforiamo il limite
        ↪  superiore di secondi (3600)
        SELECT SUM(lunghezza_in_secondi) INTO lunghezza_totale_secondi FROM CANZONE WHERE
        ↪  produzione = NEW.produzione;

        IF (lunghezza_totale_secondi + NEW.lunghezza_in_secondi > 3600)
            THEN RAISE EXCEPTION 'Impossibile aggiungere canzone. La durata complessiva
            ↪  di un Album deve essere tra mezz ora e un ora.';
        END IF;

    ELSE

        -- Calcoliamo il numero delle canzoni
        SELECT COUNT(*) INTO numero_canzoni FROM CANZONE WHERE produzione = NEW.produzione;

        IF tipo_produzione = 'Singolo' THEN
            -- Controlliamo se con l'aggiunta di una canzone sforiamo il limite superiore
            ↪  (di 3 canzoni)
            IF (numero_canzoni + 1 > 3)
                THEN RAISE EXCEPTION 'Impossibile aggiungere canzone. Un Singolo può
                ↪  contenere al massimo 3 canzoni.';
            END IF;

        ELSEIF tipo_produzione = 'EP' THEN
            -- Controlliamo se con l'aggiunta di una canzone sforiamo il limite superiore
            ↪  dalle 5 canzoni
            IF (numero_canzoni + 1 > 5)
                THEN RAISE EXCEPTION 'Impossibile aggiungere canzone. Un Extended Play
                ↪  può contenere al massimo 5 canzoni.';

            END IF;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER RV3
BEFORE INSERT ON CANZONE
FOR EACH ROW
EXECUTE FUNCTION check_tipo_produzione();
```

```
/* RV4: Un solista può partecipare ad una canzone soltanto se: non è il compositore della
↪  produzione
 * nella quale compare la canzone interessata; non fa parte del gruppo che ha composto la
 ↪  produzione
 * nella quale si trova la canzone interessata.
 */
```

```sql
CREATE OR REPLACE FUNCTION check_participation_rule()
RETURNS TRIGGER AS $$
DECLARE
    gruppo_del_solista VARCHAR(255);
BEGIN

    -- Si controlla se il solista è a capo della produzione di cui fa parte la canzone
    PERFORM FROM produzione as p
    JOIN artista as a ON p.artista = NEW.solista
    JOIN canzone as c ON p.codice = c.produzione
    WHERE c.codice = NEW.canzone;

    -- Controllo per il solista a capo
    IF (FOUND) THEN
        RAISE EXCEPTION 'Un solista non può partecipare a una canzone di cui è già a
        ↪  capo.';
    END IF;

    -- Selezioniamo il nome del gruppo corrente del solista
    SELECT gruppo INTO gruppo_del_solista FROM solista AS s WHERE s.artista = new.solista;
    IF (gruppo_del_solista IS NOT NULL) THEN

        -- Controlliamo se il solista fa parte del gruppo creatore della produzione e di
        ↪  conseguenza della canzone
        PERFORM FROM canzone AS c
        JOIN produzione AS p ON c.produzione = p.codice
        JOIN artista AS a ON p.artista = a.nome_arte
        WHERE c.codice = NEW.canzone AND a.nome_arte = gruppo_del_solista;

        -- Controllo per il solista a capo
        IF (FOUND) THEN
            RAISE EXCEPTION 'Il solista fa parte del gruppo creatore della produzione e
            ↪  di conseguenza della canzone.';
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER RV4
BEFORE INSERT ON PARTECIPAZIONE
FOR EACH ROW
EXECUTE FUNCTION check_participation_rule();
```

4.2.2 Trigger necessari alle Procedure

```
/* T1: Dato un certo ordine, controllare che il numero di giorni prenotati non superi il
↪  numero di
 * giorni prenotabili offerti dal pacchetto.
 */
CREATE OR REPLACE FUNCTION ControllaGiorniPrenotati()
RETURNS TRIGGER AS $$
DECLARE
    numero_giorni_prenotati INT;
    numero_giorni_prenotabili INT;
    nome_tipologia VARCHAR(255);
BEGIN

    -- Recuperiamo il numero di giorni già prenotati e il numero massimo di giorni
    ↪  prenotabili
    SELECT p.n_giorni_prenotati_totali, t.n_giorni INTO numero_giorni_prenotati,
    ↪  numero_giorni_prenotabili
    FROM pacchetto as p
    JOIN tipologia as t ON t.nome = p.tipologia
    WHERE p.ordine = NEW.pacchetto;

    -- Se il numero di giorni prenotati più il corrente supera il numero massimo di
    ↪  giorni prenotabili
    IF numero_giorni_prenotati + 1 > numero_giorni_prenotabili THEN
        RAISE EXCEPTION 'Numero di giorni prenotati supera il massimo consentito per la
        ↪  tipologia.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T1
BEFORE INSERT ON prenotazione
FOR EACH ROW WHEN (NEW.tipo = TRUE)
EXECUTE FUNCTION ControllaGiorniPrenotati();
```

```
/* T2: Una sala può avere al massimo due tecnici. Uno di tipo Fonico e l'altro di tipo
↪  Tecnico del Suono.
 * Può anche darsi che una sala contenga soltanto un tecnico di tipo: "Tecnico del
 ↪  Suono_AND_Fonico".
 */
CREATE OR REPLACE FUNCTION check_max_tecnici()
RETURNS TRIGGER AS $$
DECLARE
    numero_fonici INTEGER;
    numero_tecnici INTEGER;
    numero_tecnico_e_fonico INTEGER;
```

```
BEGIN
    -- Conta dei Fonici
    SELECT COUNT(codice_fiscale) INTO numero_fonici
    FROM TECNICO WHERE sala_piano = NEW.sala_piano AND sala_numero = NEW.sala_numero AND
    ↪  tipo_tecnico = 'Fonico';

    -- Conta dei Tecnici del Suono
    SELECT COUNT(codice_fiscale) INTO numero_tecnici
    FROM TECNICO WHERE sala_piano = NEW.sala_piano AND sala_numero = NEW.sala_numero AND
    ↪  tipo_tecnico = 'Tecnico del Suono';

    -- Conta dei Tecnico del Suono_AND_Fonico
    SELECT COUNT(codice_fiscale) INTO numero_tecnico_e_fonico
    FROM TECNICO WHERE sala_piano = NEW.sala_piano AND sala_numero = NEW.sala_numero AND
    ↪  tipo_tecnico = 'Tecnico del Suono_AND_Fonico';

    IF ((numero_fonici + numero_tecnici) = 2)
        THEN RAISE EXCEPTION 'Impossibile aggiungere un tecnico, la sala comprende già un
        ↪  Tecnico del Suono e un Fonico.';
    ELSEIF (numero_fonici = 1 AND NEW.tipo_tecnico = 'Fonico')
        THEN RAISE EXCEPTION 'Impossibile aggiungere il tecnico, la sala comprende già un
        ↪  Fonico.';
    ELSEIF (numero_tecnici = 1 AND NEW.tipo_tecnico = 'Tecnico del Suono')
        THEN RAISE EXCEPTION 'Impossibile aggiungere il tecnico, la sala comprende già un
        ↪  Tecnico del Suono.';
    ELSEIF (numero_tecnico_e_fonico = 1)
        THEN RAISE EXCEPTION 'Impossibile aggiungere un tecnico, la sala comprende già un
        ↪  Tecnico del Suono_AND_Fonico.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T2
BEFORE INSERT OR UPDATE ON TECNICO
FOR EACH ROW
EXECUTE FUNCTION check_max_tecnici();
```

```
/* T3: Se un ordine è già stato pagato allora il campo "annullato" non può essere
↪  impostato a FALSE.
 */
CREATE OR REPLACE FUNCTION check_ordine_pagato()
RETURNS TRIGGER AS $$
BEGIN
    -- Se l'ordine è stato pagato otteniamo un record
    PERFORM FROM ORDINE
    JOIN PAGAMENTO ON ordine = NEW.codice
    WHERE stato = 'Pagato';
```

```
    -- Verifica se l'ordine è stato pagato
    IF FOUND THEN

        -- Se l'ordine è stato pagato, impedisce che il campo annullato sia impostato su
        ↪  TRUE
        RAISE EXCEPTION 'Non è possibile annullare un ordine già pagato.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T3
BEFORE UPDATE ON ORDINE
FOR EACH ROW WHEN (NEW.annullato = TRUE AND OLD.annullato = FALSE)
EXECUTE FUNCTION check_ordine_pagato();
```

```
/* T4: Annullando una prenotazione giornaliera dobbiamo andare a decrementare di uno il
↪  numero di giorni
 * prenotati totali di un ordine di tipo pacchetto.
 */
CREATE OR REPLACE FUNCTION decrementa_giorni_pacchetto()
RETURNS TRIGGER AS $$
BEGIN
    -- Decrementa il numero di giorni prenotati totali del pacchetto associato
    UPDATE PACCHETTO
    SET n_giorni_prenotati_totali = n_giorni_prenotati_totali - 1
    WHERE ordine = OLD.pacchetto;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T4
AFTER UPDATE ON prenotazione
FOR EACH ROW WHEN (OLD.tipo = TRUE AND NEW.annullata = TRUE AND OLD.annullata = FALSE)
EXECUTE FUNCTION decrementa_giorni_pacchetto();
```

```
/* T5: Aggiorna il numero totale di ore prenotate dopo la modifica di una fascia oraria.
 */
CREATE OR REPLACE FUNCTION aggiorna_ore_prenotate()
RETURNS TRIGGER AS $$
DECLARE
    nuove_ore_prenotate INTERVAL;
    vecchie_ore_prenotate INTERVAL;
BEGIN
    -- Calcolo delle ore prenotate
```

```
    nuove_ore_prenotate := NEW.orario_fine - NEW.orario_inizio;
    vecchie_ore_prenotate := OLD.orario_fine - OLD.orario_inizio;

    -- Aggiorna il numero totale di ore prenotate nella tabella ORARIO
    UPDATE ORARIO
    SET n_ore_prenotate_totali =
        n_ore_prenotate_totali + EXTRACT(HOUR FROM nuove_ore_prenotate) - EXTRACT(HOUR
        ↪  FROM vecchie_ore_prenotate)
    WHERE ordine = NEW.oraria;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T5
AFTER UPDATE ON FASCIA_ORARIA
FOR EACH ROW WHEN (NEW.orario_inizio != OLD.orario_inizio OR NEW.orario_fine !=
↪  OLD.orario_fine)
EXECUTE FUNCTION aggiorna_ore_prenotate();
```

```
/* T6: Una prenotazione può essere annullata solo se fa riferimento ad una data futura.
 */
CREATE OR REPLACE FUNCTION check_data_futura_per_prenotazione()
RETURNS TRIGGER AS $$
BEGIN
    -- Verifica se la data della prenotazione è futura
    IF NEW.giorno < CURRENT_DATE THEN
        -- Se la data non è futura, impedisce l'annullamento
        RAISE EXCEPTION 'Non è possibile annullare una prenotazione passata.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T6
AFTER UPDATE ON prenotazione
FOR EACH ROW WHEN (NEW.annullata = TRUE AND OLD.annullata = FALSE)
EXECUTE FUNCTION check_data_futura_per_prenotazione();
```

```
/* T7: Un ordine può essere annullato solo se fa riferimento ad una data futura.
 */
CREATE OR REPLACE FUNCTION check_data_futura_per_ordine()
RETURNS TRIGGER AS $$
BEGIN
    -- Verifica se la data della prenotazione è futura
    IF NEW.timestamp < CURRENT_DATE THEN
        -- Se la data non è futura, impedisce l'annullamento
```

```
        RAISE EXCEPTION 'Non è possibile annullare un ordine passato.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T7
AFTER UPDATE ON ordine
FOR EACH ROW WHEN (NEW.annullato = TRUE AND OLD.annullato = FALSE)
EXECUTE FUNCTION check_data_futura_per_ordine();
```

5 Query

5.1 Introduzione alle Query

Sono state implementate diverse query *SQL* per rispondere a esigenze specifiche degli utenti. Di seguito sono illustrate alcune delle query più significative, suddivise in base agli utilizzatori del sistema su cui si basa il database: *Artista, Tecnico e Operatore*.

5.2 Implementazione delle Query

5.2.1 Artista

```
/* A1) ELENCARE LE PRODUZIONI COMPOSTE DA UN DETERMINATO ARTISTA
 * Vengono elencate tutte le informazioni sulle produzioni composte da un determinato
 ↪  artista.
 */
SELECT * FROM produzione WHERE artista = 'SoloXYZ';
```

```
/* A2) ELENCARE GLI ARTISTI CHE HANNO PARTECIPATO ALLA CREAZIONE DI UNA CANZONE
 * Data una certa canzone vengono mostrate le informazioni degli artisti che hanno
 ↪  partecipato ad essa.
 */
SELECT * FROM solista WHERE artista IN (
    SELECT p.solista FROM partecipazione as p WHERE p.canzone = 1
);
```

```
/* A3) VISUALIZZARE UN ELENCO DELLE PRENOTAZIONI EFFETTUATE TRA TUTTI GLI ORDINI DATO UN
 ↪  ARTISTA
 * Vengono visualizzati i dettagli delle prenotazioni effettuate da un artista.
 */
CREATE OR REPLACE VIEW informazioni_oraria_or AS
SELECT * FROM ordine
JOIN orario ON ordine.codice = orario.ordine
JOIN oraria ON oraria.orario = orario.ordine;

CREATE OR REPLACE VIEW informazioni_o_or AS
SELECT * FROM ordine AS o
JOIN pacchetto AS pa ON o.codice = pa.ordine;

(
    SELECT codice, giorno, annullata FROM prenotazione
    WHERE codice IN (

        SELECT i.prenotazione
        FROM informazioni_oraria_or as i
        WHERE i.artista = 'SoloXYZ'
    )
)
union
```

```
(
    SELECT codice, giorno, annullata FROM prenotazione
    WHERE pacchetto IN (
        SELECT i.ordine
        FROM informazioni_o_or as i
        WHERE i.artista = 'SoloXYZ'
    )
);
```

5.2.2 Tecnico

```
/* T1) ELENCARE LE CANZONI A CUI LAVORA UN TECNICO
 * Viene visualizzato un elenco di canzoni a cui lavora un tecnico; viene richiesto di
 ↪  visualizzare il titolo,
 * la produzione di cui fa parte se è in una produzione, la lunghezza, data di
 ↪  registrazione, testo.
 */
SELECT titolo, produzione, lunghezza_in_secondi, data_di_registrazione, testo FROM LAVORA_A
JOIN CANZONE on LAVORA_A.canzone = CANZONE.codice
WHERE LAVORA_A.tecnico = 'TCNAUD85M01H501Z';
```

```
/* T2) INSERIRE I DATI DI UNA CANZONE
 * Inserimento informazioni generali della canzone titolo, lunghezza, data di
 ↪  registrazione, testo, e il file
 * associato contenente il nome, il percorso di sistema e l'estensione.
 */
INSERT INTO Canzone (titolo, produzione, testo, data_di_registrazione,
↪  lunghezza_in_secondi, nome_del_file, percorso_di_sistema, estensione) VALUES ...;
```

```
/* T3) CREAZIONE DI UNA PRODUZIONE
 * Inserimento del titolo, viene indicato l'artista che la possiede e viene impostato lo
 ↪  stato di "produzione",
 * la data di inizio.
 */
INSERT INTO PRODUZIONE (titolo, artista, data_inizio, data_fine, stato, tipo_produzione,
↪  genere) VALUES ...;
```

```
/* T4) PUBBLICAZIONE DI UNA PRODUZIONE
 * Viene impostata per una data produzione lo stato di "Pubblicazione" che va ad indicare
 ↪  l'immutabilità di
 * essa e viene impostata la data di termine delle registrazioni.
 */
UPDATE PRODUZIONE as p
```

```
SET stato = 'Pubblicazione', data_fine = CURRENT_DATE
WHERE codice = 2;
```

```
/* T5) INSERIRE UNA CANZONE IN UNA PRODUZIONE
 * Una volta terminata la registrazione di una Canzone essa può inserita nella Produzione
 ↪  di cui fa parte.
 */
UPDATE canzone
SET produzione = 2
WHERE codice = 2 AND (
    (SELECT stato FROM PRODUZIONE WHERE codice = 2) != 'Pubblicazione'
);
```

5.2.3 Operatore

```
/* O1) CREARE UN ORDINE
 * L'operatore registra gli ordini previo accordo con il cliente tramite chiamata
 ↪  telefonica.
 */
[IMPLEMENTATA TRAMITE PROCEDURA]
```

```
/* O2) ELENCO INFORMAZIONI DI UN DATO ORDINE
 * Vengono visualizzate tutte le informazioni di interesse di un dato ordine.
 */
SELECT s.codice, s.timestamp, s.artista, s.annullato, s.operatore, p.stato, p.metodo,
↪  p.costo_totale
FROM pagamento AS p
JOIN (
    SELECT o.codice, o.timestamp, o.artista, o.annullato, o.operatore
    FROM ordine AS o
    WHERE o.codice = 1
) AS s ON s.codice = p.ordine;
```

```
/* O3) ANNULLARE UN ORDINE
 * L'operatore può annullare un ordine se il cliente lo richiede o non paga dopo aver
 ↪  chiamato l'operatore.
 */
UPDATE ORDINE
SET annullato = TRUE
WHERE codice = 1;
```

```
/* O4) CREARE UNA PRENOTAZIONE
 * L'operatore registra le prenotazioni previo accordo con il cliente tramite chiamata
 ↪  telefonica.
```

```
 */
[IMPLEMENTATA TRAMITE PROCEDURA]
```

```
/* 05) ANNULLARE UNA PRENOTAZIONE
 * L'operatore può annullare una prenotazione previo accordo con il cliente tramite
 ↪  chiamata telefonica.
 */
UPDATE PRENOTAZIONE
SET annullata = TRUE
WHERE codice = 1;
```

```
/* 06) VISUALIZZARE LE INFORMAZIONI RELATIVE AL PAGAMENTO DI UN ORDINE
 * Vengono visualizzate le informazioni nome, cognome del cliente, data in cui è stata
 ↪  effettuato l'ordine,
 * lo stato "pagato", "da pagare", costo totale dell'ordine.
 */

-- Caso del Solista
CREATE OR REPLACE VIEW informazioni_ordine_solista AS
SELECT o.codice, s.nome, s.cognome, t.numero, o.timestamp
FROM ordine AS o
JOIN artista    AS a ON a.nome_arte = o.artista
JOIN solista    AS s ON a.nome_arte = s.artista
JOIN telefono_a AS t ON a.nome_arte = t.artista;

select * from informazioni_ordine_solista
JOIN PAGAMENTO AS p ON p.ordine = <codice_ordine>;

-- Caso del Gruppo
CREATE OR REPLACE VIEW informazioni_ordine_gruppo AS
SELECT o.codice, a.nome_arte, t.numero, o.timestamp
FROM ORDINE AS o
JOIN ARTISTA    AS a ON a.nome_arte = o.artista
JOIN GRUPPO     AS g ON a.nome_arte = g.artista
JOIN TELEFONO_A AS t ON a.nome_arte = t.artista;

select * from informazioni_ordine_gruppo
JOIN PAGAMENTO  AS p ON p.ordine = <codice_ordine>;
```

```
/* 07) ELENCARE GLI ORDINI CHE NON SONO ANCORA STATI PAGATI
 * Viene visualizzato un elenco di ordini non pagati e informazioni di chi ha fatto
 ↪  l'ordine: nome, cognome,
 * telefono, data di effettuazione dell'ordine e il costo totale. Gli stati possibili
 ↪  sono 'Pagato', 'Da Pagare'.
 */
```

```sql
-- Caso del Solista
CREATE OR REPLACE VIEW ordini_non_pagati_solisti AS
SELECT p.ordine, sub.nome, sub.cognome, sub.numero, sub.timestamp
FROM PAGAMENTO AS p
JOIN (
    SELECT o.codice, s.nome, s.cognome, te.numero, o.timestamp
    FROM ORDINE AS o, ARTISTA AS a, SOLISTA AS s, TELEFONO_A AS te
    WHERE o.artista = a.nome_arte
    AND s.artista = a.nome_arte
        AND te.artista = a.nome_arte
) as sub ON p.ordine = sub.codice WHERE p.stato = 'Da pagare';

select * from ordini_non_pagati_solisti;

-- Caso del Gruppo
CREATE OR REPLACE VIEW ordini_non_pagati_gruppo AS
SELECT p.ordine, sub.nome_arte, sub.numero, sub.timestamp
FROM PAGAMENTO AS p
JOIN (
    SELECT a.nome_arte, o.codice, te.numero, o.timestamp
    FROM ORDINE AS o, ARTISTA AS a, GRUPPO AS g, TELEFONO_A AS te
    WHERE o.artista = a.nome_arte
    AND g.artista = a.nome_arte
        AND te.artista = a.nome_arte
) as sub ON p.ordine = sub.codice WHERE p.stato = 'Da pagare';

select * from ordini_non_pagati_gruppo;
```

```sql
/* 08) ELENCARE LE SALE DISPONIBILI
 * Viene visualizzato un elenco di tutte le sale libere per una certa data, ora di inizio
 ↪  e ora di fine
 */
(
    SELECT s.numero, s.piano
    FROM SALA s
    WHERE (s.numero, s.piano) NOT IN (

        SELECT DISTINCT p.sala_numero, p.sala_piano
        FROM PRENOTAZIONE p
        JOIN ORARIA o ON o.prenotazione = p.codice
        JOIN FASCIA_ORARIA f ON f.oraria = o.prenotazione
        WHERE p.annullata = FALSE
        AND p.tipo = FALSE
        AND p.giorno = '2023-02-04'
        AND (
            (f.orario_inizio <= '<orario_inizio>' AND '<orario_inizio>' < f.orario_fine)
            ↪  OR -- [ 10:00 ( 11:00] )
            (f.orario_inizio < '<orario_fine>' AND '<orario_fine>' <= f.orario_fine) OR
            ↪     -- ( [ 10:00 ) 11:00]
```

```
            ('<orario_inizio>' <= f.orario_inizio AND f.orario_fine <= '<orario_fine>')
            ↪     -- ( [ 10:00 11:00] )
        )
    )
)
EXCEPT
(
    SELECT sala_numero, sala_piano
    FROM PRENOTAZIONE
    WHERE annullata = FALSE AND tipo = TRUE AND giorno = '<giorno>'
);

/* DATI di PROVA
 *  giorno '2023-02-04'
 *  orari_inizi['20:00:00', '18:00:00', '10:00:00', '12:00:00', '12:00:00', '11:00:00']
 *  orarifine['23:00:00', '23:00:00', '12:00:00', '13:00:00', '14:00:00', '13:00:00']
 */
```

6 Indci

6.1 Introduzione agli Indici

Abbiamo condotto uno studio sui tempi di esecuzione delle query, incrementando drasticamente il numero di record per valutare l'efficienza delle operazioni. Attraverso questa analisi, abbiamo identificato diversi indici che possono essere inseriti all'interno del database per migliorare le prestazioni delle query.

6.2 Implementazione e Motivazione degli Indici

6.2.1 Indice sull'attributo Artista dell'entità Ordine

```
-- Definizione dell'Indice
CREATE INDEX artista_ordine_index ON ordine (artista);
```

L'indice *artista_ordine* è stato creato per ottimizzare le query che coinvolgono la colonna *artista* nella tabella *ordine*. Quest'indice è utile quando si effettua una ricerca basata sull'artista negli ordini, riducendo il tempo di ricerca.

6.2.1.1 Studio della Query O7

```
/*
 * 07) ELENCARE GLI ORDINI CHE NON SONO ANCORA STATI PAGATI
 */
CREATE OR REPLACE VIEW ordini_non_pagati_gruppo AS
SELECT p.ordine, sub.nome_arte, sub.numero, sub.timestamp
FROM PAGAMENTO AS p
JOIN (
    SELECT a.nome_arte, o.codice, te.numero, o.timestamp
    FROM ORDINE AS o, ARTISTA AS a, GRUPPO AS g, TELEFONO_A AS te
    WHERE o.artista = a.nome_arte
    AND g.artista = a.nome_arte
    AND te.artista = a.nome_arte
) as sub ON p.ordine = sub.codice WHERE p.stato = 'Da pagare';

-- Analisi della query interessata
EXPLAIN (ANALYSE, BUFFERS, VERBOSE)
    select * from ordini_non_pagati_gruppo;
```

L'uso dell'indice *artista_ordine* ottimizza questa vista poiché l'operazione di join tra *ORDINE* e *ARTISTA* viene velocizzata.

Tabella Interessata: Numero di record creati:	Artista 100k
Tabella Interessata: Numero di record creati:	Ordine 100k
Tempo di esecuzione senza indice: Tempo di esecuzione con indice:	20ms 0.1ms

1	Nested Loop (cost=14.28..1304.01 rows=1 width=73) (actual time=5.707..20.483 rows=2 loops=1)
2	-> Nested Loop (cost=13.99..1303.66 rows=1 width=73) (actual time=0.046..20.451 rows=7 loops=1)
3	-> Nested Loop (cost=13.57..553.07 rows=1 width=1093) (actual time=0.035..0.163 rows=7 loops=1)
4	-> Hash Join (cost=13.15..24.80 rows=130 width=1080) (actual time=0.024..0.049 rows=7 loops=1)
5	Hash Cond: ((te.artista)::text = (g.artista)::text)
6	-> Seq Scan on telefono_a te (cost=0.00..11.30 rows=130 width=564) (actual time=0.009..0.015 rows=10 loops=1)
7	-> Hash (cost=11.40..11.40 rows=140 width=516) (actual time=0.009..0.010 rows=7 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on gruppo g (cost=0.00..11.40 rows=140 width=516) (actual time=0.005..0.005 rows=7 loops=1)
10	-> Index Only Scan using artista_pkey on artista a (cost=0.42..4.06 rows=1 width=13) (actual time=0.012..0.012 rows=1 loops=7)
11	Index Cond: (nome_arte = (g.artista)::text)
12	Heap Fetches: 0
13	-> Index Scan using unique_ordine on ordine o (cost=0.42..750.58 rows=1 width=25) (actual time=0.007..2.897 rows=1 loops=7)
14	Index Cond: ((artista)::text = (a.nome_arte)::text)
15	-> Index Scan using pagamento_pkey on pagamento p (cost=0.29..0.35 rows=1 width=4) (actual time=0.004..0.004 rows=0 loops=7)
16	Index Cond: (ordine = o.codice)
17	Filter: ((stato)::text = 'Da pagare'::text)
18	Rows Removed by Filter: 1
19	Planning Time: 1.940 ms
20	Execution Time: 20.544 ms

Senza indice

1	Nested Loop (cost=14.28..553.91 rows=1 width=73) (actual time=0.067..0.111 rows=2 loops=1)
2	-> Nested Loop (cost=13.99..553.56 rows=1 width=73) (actual time=0.038..0.098 rows=7 loops=1)
3	-> Nested Loop (cost=13.57..553.07 rows=1 width=1093) (actual time=0.030..0.061 rows=7 loops=1)
4	-> Hash Join (cost=13.15..24.80 rows=130 width=1080) (actual time=0.020..0.023 rows=7 loops=1)
5	Hash Cond: ((te.artista)::text = (g.artista)::text)
6	-> Seq Scan on telefono_a te (cost=0.00..11.30 rows=130 width=564) (actual time=0.007..0.008 rows=10 loops=1)
7	-> Hash (cost=11.40..11.40 rows=140 width=516) (actual time=0.008..0.008 rows=7 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on gruppo g (cost=0.00..11.40 rows=140 width=516) (actual time=0.004..0.005 rows=7 loops=1)
10	-> Index Only Scan using artista_pkey on artista a (cost=0.42..4.06 rows=1 width=13) (actual time=0.005..0.005 rows=1 loops=7)
11	Index Cond: (nome_arte = (g.artista)::text)
12	Heap Fetches: 0
13	-> Index Scan using ordine_artista_indx on ordine o (cost=0.42..0.49 rows=1 width=25) (actual time=0.005..0.005 rows=1 loops=7)
14	Index Cond: ((artista)::text = (a.nome_arte)::text)
15	-> Index Scan using pagamento_pkey on pagamento p (cost=0.29..0.35 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=7)
16	Index Cond: (ordine = o.codice)
17	Filter: ((stato)::text = 'Da pagare'::text)
18	Rows Removed by Filter: 1
19	Planning Time: 2.304 ms
20	Execution Time: 0.149 ms

Con indice

6.2.1.2 Studio della nuova Query N1

```
/*
 * N1) CONTARE L'AMMONTARE SPESO DA UN ARTISTA E IL NUMERO DI ORDINI EFFETTUATI
 */
EXPLAIN (ANALYSE, BUFFERS, VERBOSE)
    SELECT SUM(p.costo_totale) AS costo_totale, count(*) AS numero_ordini
    FROM PAGAMENTO AS p
    JOIN ORDINE AS o ON p.ordine = o.codice
    WHERE o.artista = 'SoloXZ 25000';
```

Anche in questo secondo caso l'uso dell'indice *artista_ordine* ottimizza questa query poiché l'operazione di join tra *ORDINE* e *ARTISTA* viene velocizzata.

Tabella Interessata: Numero di record creati:	Artista 30k
Tabella Interessata: Numero di record creati:	Ordine 30k
Tempo di esecuzione senza indice: Tempo di esecuzione con indice:	3ms 0.05ms

```
1  Aggregate  (cost=472.08..472.09 rows=1 width=40) (actual time=2.985..2.986 rows=1 loops=1)
2    -> Nested Loop  (cost=0.28..472.02 rows=10 width=16) (actual time=2.584..2.979 rows=1 loops=1)
3          -> Seq Scan on ordine o  (cost=0.00..326.66 rows=18 width=4) (actual time=2.569..2.964 rows=1 loops=1)
4                Filter: ((artista)::text = 'SoloXZ 25000'::text)
5                Rows Removed by Filter: 30010
6          -> Index Scan using pagamento_pkey on pagamento p  (cost=0.28..8.08 rows=1 width=20) (actual time=0.012..0.012 rows=1 loops=1)
7                Index Cond: (ordine = o.codice)
8  Planning Time: 0.115 ms
9  Execution Time: 3.009 ms
```

Senza indice

```
1  Aggregate  (cost=19.94..19.95 rows=1 width=40) (actual time=0.026..0.026 rows=1 loops=1)
2    -> Nested Loop  (cost=0.14..19.93 rows=1 width=16) (actual time=0.014..0.014 rows=0 loops=1)
3          -> Seq Scan on ordine o  (cost=0.00..11.62 rows=1 width=4) (actual time=0.013..0.013 rows=0 loops=1)
4                Filter: ((artista)::text = 'SoloXZ 25000'::text)
5                Rows Removed by Filter: 10
6          -> Index Scan using pagamento_pkey on pagamento p  (cost=0.14..8.16 rows=1 width=20) (never executed)
7                Index Cond: (ordine = o.codice)
8  Planning Time: 2.406 ms
9  Execution Time: 0.052 ms
```

Con indice

6.2.2 Indice sull'attributo Data-Inizio dell'entità Produzione

```
-- Definizione dell'Indice
CREATE INDEX data_inizio_produzione_index ON produzione (data_inizio);
```

L'indice *data_inizio_produzione* è stato creato per ottimizzare le query che coinvolgono la colonna *data_inizio* nella tabella *produzione*. Quest'indice è utile quando si effettua una ricerca basata sulla data di inizio registrazioni nella tabella produzione, riducendo il tempo di ricerca.

6.2.2.1 Studio della nuova Query N2

```
/*
 * N2) VISUALIZZA INFORMAZIONI PRODUZIONI AVVENUTE DOPO DATA FORNITA
 */
EXPLAIN (ANALYSE, BUFFERS, VERBOSE)
    SELECT * FROM PRODUZIONE
    WHERE data_inizio >= '2023-01-01';
```

L'uso dell'indice *data_inizio_produzione* ottimizza questa query poiché l'operazione di selezione di una produzione la cui data di inizio è successiva ad una determinata data viene velocizzata.

Tabella Interessata: Numero di record creati:	Artista 30k
Tabella Interessata: Numero di record creati:	Produzione 31k
Tempo di esecuzione senza indice: Tempo di esecuzione con indice:	2.5ms 0.2ms

1	Seq Scan on public.produzione (cost=0.00..771.89 rows=1498 width=60) (actual time=2.313..2.430 rows=1500 loops=1)
2	Output: codice, titolo, artista, data_inizio, data_fine, stato, tipo_produzione, genere
3	Filter: (produzione.data_inizio >= '2023-01-01'::date)
4	Rows Removed by Filter: 30011
5	Buffers: shared hit=378
6	Planning Time: 0.065 ms
7	Execution Time: 2.474 ms

Senza indice

1	Index Scan using data_inizio_produzione_indx on public.produzione (cost=0.29..57.14 rows=1499 width=60) (actual time=0.006..0.132 rows=1500 loops=1)
2	Output: codice, titolo, artista, data_inizio, data_fine, stato, tipo_produzione, genere
3	Index Cond: (produzione.data_inizio >= '2023-01-01'::date)
4	Buffers: shared hit=21
5	Planning:
6	Buffers: shared hit=3
7	Planning Time: 0.074 ms
8	Execution Time: 0.175 ms

Con indice

6.2.3 Indice sull'attributo Giorno dell'entità Prenotazione

```
-- Definizione dell'Indice
CREATE INDEX giorno_prenotazione_index ON prenotazione (giorno);
```

L'indice *giorno_prenotazione* è stato creato per ottimizzare le query che coinvolgono la colonna *giorno* nella tabella *prenotazione*. Quest'indice è utile quando si effettua una ricerca basata sul giorno delle prenotazioni, riducendo il tempo di ricerca.

6.2.3.1 Studio della nuova Query N3

```
/*
 * N3) CONTARE IL NUMERO DI PRENOTAZIONI AVVENUTE DOPO UNA DATA
 */
EXPLAIN (ANALYSE, BUFFERS, VERBOSE)
    SELECT count(*) AS numero_prenotazioni
    FROM PRENOTAZIONE AS p
    WHERE p.giorno >= '2023-01-01' ;
```

L'uso dell'indice *giorno_prenotazione* ottimizza questa query poiché l'operazione di selezione di una prenotazione il cui giorno di effettuazione è successivo ad un determinato giorno viene velocizzata.

Tabella Interessata: Numero di record creati:	Artista 31k
Tabella Interessata: Numero di record creati:	Prenotazione 31k
Tempo di esecuzione senza indice: Tempo di esecuzione con indice:	2ms 0.2ms

1	Aggregate (cost=598.77..598.78 rows=1 width=8) (actual time=2.307..2.308 rows=1 loops=1)
2	-> Seq Scan on prenotazione p (cost=0.00..594.89 rows=1552 width=0) (actual time=0.025..2.262 rows=1543 loops=1)
3	Filter: (giorno >= '2023-01-01'::date)
4	Rows Removed by Filter: 29968
5	Planning Time: 0.778 ms
6	Execution Time: 2.326 ms

Senza indice

1	Aggregate (cost=51.33..51.34 rows=1 width=8) (actual time=0.156..0.157 rows=1 loops=1)
2	-> Index Only Scan using data_inizio_prenotazione_idx on prenotazione p (cost=0.29..47.45 rows=1552 width=0) (actual time=0.013..0.112 rows=1543 loops=1)
3	Index Cond: (giorno >= '2023-01-01'::date)
4	Heap Fetches: 0
5	Planning Time: 0.089 ms
6	Execution Time: 0.174 ms

Con indice

6.2.4 Indice sull'attributo Data di Registrazione dell'entità Canzone

```
-- Definizione dell'Indice
CREATE INDEX data_di_registrazione_canzone_index ON canzone (data_di_registrazione);
```

L'indice *data_di_registrazione_canzone* è stato creato per ottimizzare le query che coinvolgono la colonna *data_di_registrazione* nella tabella *canzone*. Quest'indice è utile quando si effettua una ricerca basata sul giorno delle canzoni, riducendo il tempo di ricerca.

6.2.4.1 Studio della nuova Query N4

```
/*
 * N4) SELEZIONA LE CANZONI PIU' VECCHIE DI UNA CERTA DATA
 */
EXPLAIN (ANALYSE, BUFFERS, VERBOSE)
    SELECT * FROM CANZONE
    WHERE data_di_registrazione < '2020-04-10' ;
```

L'uso dell'indice *data_di_registrazione_canzone* ottimizza questa query poiché l'operazione di selezione di una canzone la cui data di registrazione è antecedente ad una determinata data viene velocizzata.

Tabella Interessata: Numero di record creati:	Canzone 40k
Tempo di esecuzione senza indice: Tempo di esecuzione con indice:	3ms 0.02ms

1	Seq Scan on public.canzone (cost=0.00..994.14 rows=301 width=65) (actual time=0.009..3.084 rows=67 loops=1)
2	Output: codice, titolo, produzione, testo, data_di_registrazione, lunghezza_in_secondi, nome_del_file, percorso_di_sistema, estensione
3	Filter: (canzone.data_di_registrazione < '2020-04-10'::date)
4	Rows Removed by Filter: 39944
5	Buffers: shared hit=494
6	Planning Time: 0.070 ms
7	Execution Time: 3.112 ms

Senza indice

1	Index Scan using data_registrazione_canzone_idx on public.canzone (cost=0.29..16.56 rows=301 width=65) (actual time=0.005..0.009 rows=67 loops=1)
2	Output: codice, titolo, produzione, testo, data_di_registrazione, lunghezza_in_secondi, nome_del_file, percorso_di_sistema, estensione
3	Index Cond: (canzone.data_di_registrazione < '2020-04-10'::date)
4	Buffers: shared hit=3
5	Planning:
6	Buffers: shared hit=3
7	Planning Time: 0.080 ms
8	Execution Time: 0.022 ms

Con indice

6.2.5 Indice sull'attributo Data di Registrazione dell'entità artista

```
-- Definizione dell'Indice
CREATE INDEX data_di_registrazione_artista_index ON artista (data_di_registrazione);
```

L'indice *data_di_registrazione_artista* è stato creato per ottimizzare le query che coinvolgono la colonna *data_di_registrazione* nella tabella *artista*. Quest'indice è utile quando si effettua una ricerca basata sulla data di registrazione degli artisti, riducendo il tempo di ricerca.

6.2.5.1 Studio della nuova Query N5

```
/*
 * N5) SELEZIONE TUTTI I DATI DI UN ARTISTA CHE SI E' REGISTRATO NEL SISTEMA PRIMA DI UNA
 ↪  CERTA DATA
 */
EXPLAIN (ANALYSE, BUFFERS, VERBOSE)
    SELECT * FROM ARTISTA AS a
    WHERE a.data_di_registrazione <= '2020-01-01';
```

L'uso dell'indice *data_di_registrazione_artista* ottimizza questa query poiché l'operazione di selezione di un artista la cui data di registrazione è antecedente ad una determinata data viene velocizzata.

Tabella Interessata: Numero di record creati:	Artista 30k
Tempo di esecuzione senza indice: Tempo di esecuzione con indice:	3ms 0.03ms

1	Seq Scan on artista a (cost=0.00..213.85 rows=849 width=520) (actual time=0.016..2.972 rows=8 loops=1)
2	Filter: (data_di_registrazione <= '2020-01-01'::date)
3	Rows Removed by Filter: 30007
4	Planning Time: 0.044 ms
5	Execution Time: 2.983 ms

Senza indice

1	Index Scan using data_registrazione_artista_idx on artista a (cost=0.29..8.43 rows=8 width=16) (actual time=0.018..0.019 rows=8 loops=1)
2	Index Cond: (data_di_registrazione <= '2020-01-01'::date)
3	Planning Time: 1.343 ms
4	Execution Time: 0.027 ms

Con indice