sql\install.sql

```sql
-- Drop and recreate the database
DROP DATABASE IF EXISTS pulse_university;
CREATE DATABASE IF NOT EXISTS pulse_university;
USE pulse_university;

-- Drop all tables
DROP TABLE IF EXISTS Continent;
DROP TABLE IF EXISTS Staff_Role;
DROP TABLE IF EXISTS Experience_Level;
DROP TABLE IF EXISTS Performance_Type;
DROP TABLE IF EXISTS Ticket_Type;
DROP TABLE IF EXISTS Payment_Method;
DROP TABLE IF EXISTS Ticket_Status;
DROP TABLE IF EXISTS Genre;
DROP TABLE IF EXISTS SubGenre;
DROP TABLE IF EXISTS Location;
DROP TABLE IF EXISTS Festival;
DROP TABLE IF EXISTS Stage;
DROP TABLE IF EXISTS Equipment;
DROP TABLE IF EXISTS Stage_Equipment;
DROP TABLE IF EXISTS Event;
DROP TABLE IF EXISTS Staff;
DROP TABLE IF EXISTS Works_On;
DROP TABLE IF EXISTS Performance;
DROP TABLE IF EXISTS Artist;
DROP TABLE IF EXISTS Artist_Genre;
DROP TABLE IF EXISTS Artist_SubGenre;
DROP TABLE IF EXISTS Band;
DROP TABLE IF EXISTS Band_Genre;
DROP TABLE IF EXISTS Band_SubGenre;
DROP TABLE IF EXISTS Band_Member;
DROP TABLE IF EXISTS Performance_Band;
DROP TABLE IF EXISTS Performance_Artist;
DROP TABLE IF EXISTS Attendee;
DROP TABLE IF EXISTS Ticket;
DROP TABLE IF EXISTS Review;
DROP TABLE IF EXISTS Resale_Offer;
DROP TABLE IF EXISTS Resale_Interest;
DROP TABLE IF EXISTS Resale_Interest_Type;
DROP TABLE IF EXISTS Resale_Match_Log;

-- Lookup Tables
CREATE TABLE Continent (
    continent_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Continent (name) VALUES
('Africa'),
('Asia'),
```

```
('Europe'),
('North America'),
('South America'),
('Oceania');

CREATE TABLE Staff_Role (
    role_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Staff_Role (name) VALUES
('security'),
('support'),
('sound engineer'),
('light technician'),
('stagehand'),
('medic'),
('cleaning'),
('backstage assistant');

CREATE TABLE Experience_Level (
    level_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Experience_Level (name) VALUES
('intern'),
('beginner'),
('intermediate'),
('experienced'),
('expert');

CREATE TABLE Performance_Type (
    type_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Performance_Type (name) VALUES
('warm up'),
('headline'),
('special guest'),
('encore'),
('other');

CREATE TABLE Ticket_Type (
    type_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Ticket_Type (name) VALUES
```

```
('general'),
('VIP'),
('backstage'),
('early bird'),
('student');

CREATE TABLE Payment_Method (
    method_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Payment_Method (name) VALUES
('credit card'),
('debit card'),
('bank transfer');

CREATE TABLE Ticket_Status (
    status_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(20) NOT NULL UNIQUE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO Ticket_Status (name) VALUES
('active'),
('used'),
('on offer'),
('unused');

CREATE TABLE Genre (
    genre_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);

INSERT INTO Genre (name) VALUES
('Rock'),
('Pop'),
('Jazz'),
('Hip Hop'),
('Electronic'),
('Classical'),
('Reggae'),
('Latin'),
('Metal'),
('Funk');

CREATE TABLE SubGenre (
    sub_genre_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    genre_id INT UNSIGNED NOT NULL,
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id) ON DELETE CASCADE
);

INSERT INTO SubGenre (name, genre_id) VALUES
```

```
('Hard Rock',         (SELECT genre_id FROM Genre WHERE name='Rock')),
('Progressive Rock',  (SELECT genre_id FROM Genre WHERE name='Rock')),
('Punk Rock',         (SELECT genre_id FROM Genre WHERE name='Rock')),
('Synthpop',          (SELECT genre_id FROM Genre WHERE name='Pop')),
('Electropop',        (SELECT genre_id FROM Genre WHERE name='Pop')),
('Dance Pop',         (SELECT genre_id FROM Genre WHERE name='Pop')),
('Bebop',             (SELECT genre_id FROM Genre WHERE name='Jazz')),
('Smooth Jazz',       (SELECT genre_id FROM Genre WHERE name='Jazz')),
('Free Jazz',         (SELECT genre_id FROM Genre WHERE name='Jazz')),
('Trap',              (SELECT genre_id FROM Genre WHERE name='Hip Hop')),
('Boom Bap',          (SELECT genre_id FROM Genre WHERE name='Hip Hop')),
('Lo-fi Hip Hop',     (SELECT genre_id FROM Genre WHERE name='Hip Hop')),
('Techno',            (SELECT genre_id FROM Genre WHERE name='Electronic')),
('House',             (SELECT genre_id FROM Genre WHERE name='Electronic')),
('Trance',            (SELECT genre_id FROM Genre WHERE name='Electronic')),
('Baroque',           (SELECT genre_id FROM Genre WHERE name='Classical')),
('Romantic',          (SELECT genre_id FROM Genre WHERE name='Classical')),
('Contemporary Classical',(SELECT genre_id FROM Genre WHERE name='Classical')),
('Dub',               (SELECT genre_id FROM Genre WHERE name='Reggae')),
('Dancehall',         (SELECT genre_id FROM Genre WHERE name='Reggae')),
('Roots Reggae',      (SELECT genre_id FROM Genre WHERE name='Reggae')),
('Salsa',             (SELECT genre_id FROM Genre WHERE name='Latin')),
('Reggaeton',         (SELECT genre_id FROM Genre WHERE name='Latin')),
('Bachata',           (SELECT genre_id FROM Genre WHERE name='Latin')),
('Death Metal',       (SELECT genre_id FROM Genre WHERE name='Metal')),
('Black Metal',       (SELECT genre_id FROM Genre WHERE name='Metal')),
('Thrash Metal',      (SELECT genre_id FROM Genre WHERE name='Metal')),
('Afrofunk',          (SELECT genre_id FROM Genre WHERE name='Funk')),
('P-Funk',            (SELECT genre_id FROM Genre WHERE name='Funk')),
('Jazz-Funk',         (SELECT genre_id FROM Genre WHERE name='Funk'));

-- Main Tables
CREATE TABLE Location (
    loc_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    street_name  VARCHAR(255) NOT NULL,
    street_number VARCHAR(20) NOT NULL,
    zip_code VARCHAR(10) NOT NULL,
    city VARCHAR(100) NOT NULL,
    country VARCHAR(100) NOT NULL,
    continent_id INT UNSIGNED NOT NULL,
    latitude  DECIMAL(9,6) NOT NULL,
    longitude DECIMAL(9,6) NOT NULL,
    image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    FOREIGN KEY (continent_id) REFERENCES Continent(continent_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Festival (
    fest_year INT UNSIGNED PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date   DATE NOT NULL,
```

```sql
    image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    loc_id INT UNSIGNED NOT NULL,
    FOREIGN KEY (loc_id) REFERENCES Location(loc_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Stage (
    stage_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name     VARCHAR(255) NOT NULL,
    capacity INT NOT NULL CHECK (capacity > 0),
    image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Equipment (
    equip_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name   VARCHAR(255) NOT NULL,
    image  VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Stage_Equipment (
    stage_id INT UNSIGNED,
    equip_id INT UNSIGNED,
    PRIMARY KEY(stage_id, equip_id),
    FOREIGN KEY(stage_id) REFERENCES Stage(stage_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(equip_id) REFERENCES Equipment(equip_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Event (
    event_id  INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title     VARCHAR(255) NOT NULL,
    is_full   BOOLEAN NOT NULL DEFAULT FALSE,
    start_dt  DATETIME NOT NULL,
    end_dt    DATETIME NOT NULL,
    image     VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption   VARCHAR(100) NOT NULL,
    fest_year INT UNSIGNED NOT NULL,
    stage_id  INT UNSIGNED NOT NULL,
    generated_date DATE NOT NULL,                       -- updated via trigger
    UNIQUE KEY uq_event_stage (event_id, stage_id),
    UNIQUE KEY uq_start_date (generated_date),
    FOREIGN KEY (fest_year) REFERENCES Festival(fest_year)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (stage_id) REFERENCES Stage(stage_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

```
);

CREATE TABLE Staff (
    staff_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name  VARCHAR(100) NOT NULL,
    date_of_birth DATE NOT NULL,
    role_id   INT UNSIGNED NOT NULL,
    experience_id INT UNSIGNED NOT NULL,
    image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    FOREIGN KEY (role_id)      REFERENCES Staff_Role(role_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (experience_id) REFERENCES Experience_Level(level_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Works_On (
    staff_id INT UNSIGNED,
    event_id INT UNSIGNED,
    PRIMARY KEY(staff_id, event_id),
    FOREIGN KEY(staff_id) REFERENCES Staff(staff_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(event_id) REFERENCES Event(event_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Performance (
    perf_id  INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_id  INT UNSIGNED NOT NULL,
    datetime DATETIME NOT NULL,
    duration TINYINT UNSIGNED NOT NULL CHECK (duration BETWEEN 1 AND 180),
    break_duration TINYINT CHECK (break_duration BETWEEN 5 AND 30),
    stage_id INT UNSIGNED NOT NULL,
    event_id INT UNSIGNED NOT NULL,
    sequence_number TINYINT UNSIGNED NOT NULL CHECK (sequence_number > 0),
    UNIQUE KEY uq_event_seq (event_id, sequence_number),
    CONSTRAINT fk_perf_event_stage
        FOREIGN KEY (event_id, stage_id)
        REFERENCES Event(event_id, stage_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY(stage_id) REFERENCES Stage(stage_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY(type_id)  REFERENCES Performance_Type(type_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Artist (
    artist_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name  VARCHAR(100) NOT NULL,
```

```
    nickname   VARCHAR(100),
    date_of_birth DATE NOT NULL,
    webpage   VARCHAR(100) CHECK (webpage LIKE 'https://%'),
    instagram VARCHAR(100) CHECK (instagram LIKE '@%'),
    image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Artist_Genre (
    artist_id INT UNSIGNED,
    genre_id  INT UNSIGNED,
    PRIMARY KEY (artist_id, genre_id),
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE CASCADE,
    FOREIGN KEY (genre_id)  REFERENCES Genre(genre_id)  ON DELETE CASCADE
);

CREATE TABLE Artist_SubGenre (
    artist_id INT UNSIGNED,
    sub_genre_id INT UNSIGNED,
    PRIMARY KEY (artist_id, sub_genre_id),
    FOREIGN KEY (artist_id)     REFERENCES Artist(artist_id) ON DELETE CASCADE,
    FOREIGN KEY (sub_genre_id)  REFERENCES SubGenre(sub_genre_id) ON DELETE CASCADE
);

CREATE TABLE Band (
    band_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    formation_date DATE,
    webpage   VARCHAR(100) CHECK (webpage LIKE 'https://%'),
    instagram VARCHAR(100) CHECK (instagram LIKE '@%'),
    image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
    caption VARCHAR(100) NOT NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Band_Genre (
    band_id INT UNSIGNED,
    genre_id INT UNSIGNED,
    PRIMARY KEY (band_id, genre_id),
    FOREIGN KEY (band_id) REFERENCES Band(band_id)   ON DELETE CASCADE,
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id) ON DELETE CASCADE
);

CREATE TABLE Band_SubGenre (
    band_id INT UNSIGNED,
    sub_genre_id INT UNSIGNED,
    PRIMARY KEY (band_id, sub_genre_id),
    FOREIGN KEY (band_id)      REFERENCES Band(band_id)      ON DELETE CASCADE,
    FOREIGN KEY (sub_genre_id) REFERENCES SubGenre(sub_genre_id) ON DELETE CASCADE
);

CREATE TABLE Band_Member (
    band_id INT UNSIGNED,
```

```
376     artist_id INT UNSIGNED,
377     PRIMARY KEY(band_id, artist_id),
378     FOREIGN KEY(band_id)  REFERENCES Band(band_id)
379         ON DELETE CASCADE ON UPDATE CASCADE,
380     FOREIGN KEY(artist_id) REFERENCES Artist(artist_id)
381         ON DELETE CASCADE ON UPDATE CASCADE,
382     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
383 );
384
385 CREATE TABLE Performance_Band (
386     perf_id INT UNSIGNED PRIMARY KEY,
387     band_id INT UNSIGNED NOT NULL,
388     FOREIGN KEY(perf_id) REFERENCES Performance(perf_id)
389         ON DELETE CASCADE ON UPDATE RESTRICT,
390     FOREIGN KEY(band_id) REFERENCES Band(band_id)
391         ON DELETE CASCADE ON UPDATE CASCADE,
392     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
393 );
394
395 CREATE TABLE Performance_Artist (
396     perf_id INT UNSIGNED,
397     artist_id INT UNSIGNED,
398     PRIMARY KEY(perf_id, artist_id),
399     FOREIGN KEY(perf_id) REFERENCES Performance(perf_id)
400         ON DELETE CASCADE ON UPDATE RESTRICT,
401     FOREIGN KEY(artist_id) REFERENCES Artist(artist_id)
402         ON DELETE CASCADE ON UPDATE CASCADE,
403     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
404 );
405
406 CREATE TABLE Attendee (
407     attendee_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
408     first_name VARCHAR(100) NOT NULL,
409     last_name  VARCHAR(100) NOT NULL,
410     date_of_birth DATE NOT NULL,
411     phone_number VARCHAR(20),
412     email VARCHAR(255),
413     CHECK (phone_number IS NOT NULL OR email IS NOT NULL),
414     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
415 );
416
417 CREATE TABLE Ticket (
418     ticket_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
419     type_id INT UNSIGNED NOT NULL,
420     purchase_date DATE NOT NULL,
421     cost DECIMAL(7,2) NOT NULL,
422     method_id INT UNSIGNED NOT NULL,
423     ean_number BIGINT NOT NULL UNIQUE,
424     status_id INT UNSIGNED NOT NULL,
425     attendee_id INT UNSIGNED NOT NULL,
426     event_id INT UNSIGNED NOT NULL,
427     UNIQUE(attendee_id, event_id),
428     FOREIGN KEY(type_id) REFERENCES Ticket_Type(type_id)
429         ON DELETE RESTRICT ON UPDATE CASCADE,
```

```
430     FOREIGN KEY(status_id) REFERENCES Ticket_Status(status_id)
431         ON DELETE RESTRICT ON UPDATE CASCADE,
432     FOREIGN KEY(method_id) REFERENCES Payment_Method(method_id)
433         ON DELETE RESTRICT ON UPDATE CASCADE,
434     FOREIGN KEY(attendee_id) REFERENCES Attendee(attendee_id)
435         ON DELETE RESTRICT ON UPDATE CASCADE,
436     FOREIGN KEY(event_id) REFERENCES Event(event_id)
437         ON DELETE RESTRICT  ON UPDATE CASCADE,
438     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
439 );
440
441 CREATE TABLE Review (
442     review_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
443     interpretation     TINYINT NOT NULL CHECK (interpretation BETWEEN 1 AND 5),
444     sound_and_visuals  TINYINT NOT NULL CHECK (sound_and_visuals BETWEEN 1 AND 5),
445     stage_presence     TINYINT NOT NULL CHECK (stage_presence  BETWEEN 1 AND 5),
446     organization       TINYINT NOT NULL CHECK (organization    BETWEEN 1 AND 5),
447     overall            TINYINT NOT NULL CHECK (overall         BETWEEN 1 AND 5),
448     attendee_id INT UNSIGNED NOT NULL,
449     perf_id     INT UNSIGNED NOT NULL,
450     UNIQUE(perf_id, attendee_id),
451     FOREIGN KEY(attendee_id) REFERENCES Attendee(attendee_id)
452         ON DELETE RESTRICT ON UPDATE CASCADE,
453     FOREIGN KEY(perf_id) REFERENCES Performance(perf_id)
454         ON DELETE CASCADE  ON UPDATE RESTRICT,
455     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
456 );
457
458 CREATE TABLE Resale_Offer (
459     offer_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
460     ticket_id INT UNSIGNED NOT NULL UNIQUE,
461     event_id  INT UNSIGNED NOT NULL,
462     seller_id INT UNSIGNED NOT NULL,
463     offer_timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
464     FOREIGN KEY(ticket_id) REFERENCES Ticket(ticket_id)
465         ON DELETE CASCADE ON UPDATE CASCADE,
466     FOREIGN KEY(event_id) REFERENCES Event(event_id)
467         ON DELETE RESTRICT ON UPDATE CASCADE,
468     FOREIGN KEY(seller_id) REFERENCES Attendee(attendee_id)
469         ON DELETE CASCADE ON UPDATE CASCADE,
470     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
471 );
472
473 CREATE TABLE Resale_Interest (
474     request_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
475     buyer_id INT UNSIGNED NOT NULL,
476     event_id INT UNSIGNED NOT NULL,
477     interest_timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
478     FOREIGN KEY(buyer_id) REFERENCES Attendee(attendee_id)
479         ON DELETE CASCADE ON UPDATE CASCADE,
480     FOREIGN KEY(event_id) REFERENCES Event(event_id)
481         ON DELETE RESTRICT ON UPDATE CASCADE,
482     UNIQUE(buyer_id, event_id),
483     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

```
);

CREATE TABLE Resale_Interest_Type (
    request_id INT UNSIGNED NOT NULL,
    type_id INT UNSIGNED NOT NULL,
    PRIMARY KEY(request_id, type_id),
    FOREIGN KEY(request_id) REFERENCES Resale_Interest(request_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(type_id) REFERENCES Ticket_Type(type_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE Resale_Match_Log (
    match_id           INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    match_type         ENUM('offer', 'interest') NOT NULL,
    ticket_id          INT UNSIGNED NOT NULL,
    offered_type_id    INT UNSIGNED NOT NULL,
    requested_type_id  INT UNSIGNED NOT NULL,
    buyer_id           INT UNSIGNED NOT NULL,
    seller_id          INT UNSIGNED NOT NULL,
    match_time         DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (offered_type_id) REFERENCES Ticket_Type(type_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (requested_type_id) REFERENCES Ticket_Type(type_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (buyer_id) REFERENCES Attendee(attendee_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (seller_id) REFERENCES Attendee(attendee_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

```

sql\indexing.sql

```sql
-- --------------
-- - Indexing ---
-- --------------

USE pulse_university;

-- Helper: drop procedure if it exists
-- MYSQL does not support DROP INDEX IF EXISTS!
DROP PROCEDURE IF EXISTS DropIndexIfExists;

CREATE PROCEDURE DropIndexIfExists(tbl VARCHAR(64), idx VARCHAR(64))
    SQL SECURITY INVOKER
    COMMENT 'Drops index only if it exists'
BEGIN
    DECLARE count INT;
    SELECT COUNT(*) INTO count
    FROM information_schema.statistics
    WHERE table_schema = DATABASE()
        AND table_name = tbl
        AND index_name = idx;

    IF count > 0 THEN
        SET @stmt = CONCAT('DROP INDEX `', idx, '` ON `', tbl, '`');
        PREPARE s FROM @stmt;
        EXECUTE s;
        DEALLOCATE PREPARE s;
    END IF;
END;

/* ------------------------------------------------------------
 * 1.  Ticket-centric queries   (Q 1, 8, 9)
 * ------------------------------------------------------------*/
CALL DropIndexIfExists('Ticket', 'idx_ticket_event_date_payment');
CALL DropIndexIfExists('Ticket', 'idx_ticket_attendee_event');
CALL DropIndexIfExists('Ticket', 'idx_ticket_attendee_year_event');

CREATE INDEX idx_ticket_event_date_payment  ON Ticket (event_id, purchase_date,
method_id);
CREATE INDEX idx_ticket_attendee_event      ON Ticket (attendee_id, event_id);
CREATE INDEX idx_ticket_attendee_year_event ON Ticket (attendee_id, purchase_date,
event_id);

/* ------------------------------------------------------------
 * 2.  Event / festival helpers   (Q 1, 2, 3, 7, 8, 10, 13, 14)
 * ------------------------------------------------------------*/
CALL DropIndexIfExists('Event', 'idx_event_year');
CALL DropIndexIfExists('Event', 'idx_event_start');

CREATE INDEX idx_event_year  ON Event (fest_year);      -- yearly roll-ups
CREATE INDEX idx_event_start ON Event(start_dt);

/* ------------------------------------------------------------
```

```
 * 3.  Genres & sub-genres   (Q 2, 10, 14)
 * ----------------------------------------------------------*/
CALL DropIndexIfExists('Artist_Genre', 'idx_artist_genre');
CALL DropIndexIfExists('Artist_Genre', 'idx_artist_genre_by_artist');

CREATE INDEX idx_artist_genre           ON Artist_Genre (genre_id, artist_id);   -- filter-first
CREATE INDEX idx_artist_genre_by_artist ON Artist_Genre (artist_id, genre_id);   -- self-join path

/* ----------------------------------------------------------
 * 4.  Artist attributes   (Q 5)
 * ----------------------------------------------------------*/
CALL DropIndexIfExists('Artist', 'idx_artist_dob');
CREATE INDEX idx_artist_dob ON Artist (date_of_birth);  -- “young artists” filter

/* ----------------------------------------------------------
 * 5.  Performance look-ups   (Q 2, 3, 5, 10, 11, 13, 14, 15)
 * ----------------------------------------------------------*/
CALL DropIndexIfExists('Performance',        'idx_perf_event_type');
CALL DropIndexIfExists('Performance',        'idx_perf_type');
CALL DropIndexIfExists('Performance',        'idx_perf_datetime');
CALL DropIndexIfExists('Performance_Artist', 'idx_perf_artist');

CREATE INDEX idx_perf_event_type ON Performance (event_id, type_id);
CREATE INDEX idx_perf_type       ON Performance (type_id);        -- quick “warm-up” filter
CREATE INDEX idx_perf_datetime   ON Performance (datetime);
CREATE INDEX idx_perf_artist     ON Performance_Artist (artist_id, perf_id);

/* ----------------------------------------------------------
 * 6.  Staff & staffing ratios   (Q 7, 8)
 * ----------------------------------------------------------*/
CALL DropIndexIfExists('Works_On', 'idx_workson_event_staff');
CALL DropIndexIfExists('Staff',    'idx_staff_role_staff');
CALL DropIndexIfExists('Staff',    'idx_staff_experience');

CREATE INDEX idx_workson_event_staff ON Works_On (event_id, staff_id);   -- event-centric scans
CREATE INDEX idx_staff_role_staff    ON Staff    (role_id,  staff_id);   -- role filter + join
CREATE INDEX idx_staff_experience    ON Staff    (experience_id);

/* ----------------------------------------------------------
 * 7.  Reviews   (Q 6, 11, 15)
 * ----------------------------------------------------------*/
CALL DropIndexIfExists('Review', 'idx_review_perf_io');
CALL DropIndexIfExists('Review', 'idx_review_attendee_overall');
CALL DropIndexIfExists('Review', 'idx_review_perf_attendee_overall');

CREATE INDEX idx_review_perf_io               ON Review (perf_id, interpretation, overall); -- covering index
CREATE INDEX idx_review_attendee_overall      ON Review (attendee_id, overall);
CREATE INDEX idx_review_perf_attendee_overall ON Review (perf_id, attendee_id, overall);
```

```sql

/* ------------------------------------------------------------
 * 8.  Geography   (Q 13)
 * ------------------------------------------------------------*/
CALL DropIndexIfExists('Location', 'idx_location_continent');
CREATE INDEX idx_location_continent ON Location (continent_id);

-- Clean up: drop helper procedure
DROP PROCEDURE IF EXISTS DropIndexIfExists;

```

sql\triggers.sql

```sql
-- --------------
-- - Triggers ---
-- --------------

USE pulse_university;

-- Drop all triggers
DROP TRIGGER IF EXISTS trg_band_validate_before_ins;
DROP TRIGGER IF EXISTS trg_band_sync_members_after_ins;
DROP TRIGGER IF EXISTS trg_artist_validate_before_ins;
DROP TRIGGER IF EXISTS trg_auto_assign_band_after_artist_ins;
DROP TRIGGER IF EXISTS trg_no_double_stage_artist;
DROP TRIGGER IF EXISTS trg_no_double_stage_band;
DROP TRIGGER IF EXISTS trg_no_stage_overlap;
DROP TRIGGER IF EXISTS trg_max_consecutive_years_artist;
DROP TRIGGER IF EXISTS trg_max_consecutive_years_band;
DROP TRIGGER IF EXISTS trg_event_within_festival_dates;
DROP TRIGGER IF EXISTS trg_performance_inside_event;
DROP TRIGGER IF EXISTS trg_safe_festival_date_update;
DROP TRIGGER IF EXISTS trg_safe_event_date_update;
DROP TRIGGER IF EXISTS trg_delete_band_if_no_members;
DROP TRIGGER IF EXISTS trg_staff_ratio_after_delete;
DROP TRIGGER IF EXISTS trg_staff_ratio_after_update;
DROP TRIGGER IF EXISTS trg_ticket_capacity_check;
DROP TRIGGER IF EXISTS trg_check_vip_ticket_limit;
DROP TRIGGER IF EXISTS trg_validate_ticket_ean;
DROP TRIGGER IF EXISTS trg_validate_ticket_ean_upd;
DROP TRIGGER IF EXISTS trg_resale_offer_only_active;
DROP TRIGGER IF EXISTS trg_review_only_with_used_ticket;
DROP TRIGGER IF EXISTS trg_resale_offer_timestamp;
DROP TRIGGER IF EXISTS trg_resale_interest_timestamp;
DROP TRIGGER IF EXISTS trg_stage_capacity_update;
DROP TRIGGER IF EXISTS trg_block_purchase_date_update;
DROP TRIGGER IF EXISTS trg_validate_ticket_purchase_date;
DROP TRIGGER IF EXISTS trg_artist_subgenre_consistency;
DROP TRIGGER IF EXISTS trg_band_subgenre_consistency;
DROP TRIGGER IF EXISTS trg_delete_attendee_cleanup;
DROP TRIGGER IF EXISTS trg_match_resale_interest;
DROP TRIGGER IF EXISTS trg_match_resale_offer;
DROP TRIGGER IF EXISTS trg_set_event_date;

-- ============================================================
-- 1. Performance-assignment rules (solo / band)
-- ============================================================

-- 1. BEFORE INSERT on Performance_Band – validate band insertion
CREATE TRIGGER trg_band_validate_before_ins
BEFORE INSERT ON Performance_Band
FOR EACH ROW
BEGIN
    -- 1.a  allow **only one** band per performance
```

```
    IF (SELECT COUNT(*) FROM Performance_Band WHERE perf_id = NEW.perf_id) > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A band is already assigned to this performance.';
    END IF;

    -- 1.b  if artists already exist, they **must** belong to that band
    IF EXISTS (
        SELECT 1
        FROM   Performance_Artist pa
        WHERE  pa.perf_id = NEW.perf_id
            AND  pa.artist_id NOT IN
                (SELECT artist_id FROM Band_Member WHERE band_id = NEW.band_id)
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Existing artist is not a member of this band.';
    END IF;
END;

-- 2. AFTER INSERT on Performance_Band – auto-insert every band member into
Performance_Artist
CREATE TRIGGER trg_band_sync_members_after_ins
AFTER INSERT ON Performance_Band
FOR EACH ROW
BEGIN
    -- INSERT IGNORE avoids duplicates thanks to (perf_id,artist_id) PK
    INSERT IGNORE INTO Performance_Artist (perf_id, artist_id)
    SELECT NEW.perf_id, bm.artist_id
    FROM   Band_Member bm
    WHERE  bm.band_id = NEW.band_id;
END;

-- 3. BEFORE INSERT on Performance_Artist – validate artist insertion
CREATE TRIGGER trg_artist_validate_before_ins
BEFORE INSERT ON Performance_Artist
FOR EACH ROW
BEGIN
    -- 3.a  If a band is already set, artist must be a member
    DECLARE v_band INT;
    SELECT band_id INTO v_band
    FROM   Performance_Band
    WHERE  perf_id = NEW.perf_id
    LIMIT  1;

    IF v_band IS NOT NULL
        AND NOT EXISTS (
            SELECT 1
            FROM   Band_Member
            WHERE  band_id = v_band
                AND  artist_id = NEW.artist_id)
    THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artist is not a member of the assigned band.';
    END IF;

```

```
    -- 3.b  If NO band yet, but other artists exist ⇒ they must all share at least one common band with the newcomer
    IF v_band IS NULL
        AND (SELECT COUNT(*) FROM Performance_Artist WHERE perf_id = NEW.perf_id) > 0
        AND NOT EXISTS (
            SELECT 1
            FROM   Band_Member bm_new                -- bands of the new artist
            WHERE  bm_new.artist_id = NEW.artist_id
                -- every existing artist must also be member of bm_new.band_id
                AND NOT EXISTS (
                    SELECT 1
                    FROM   Performance_Artist pa
                    WHERE  pa.perf_id = NEW.perf_id         -- existing artists
                        AND NOT EXISTS (
                            SELECT 1
                            FROM   Band_Member bm_old
                            WHERE  bm_old.band_id  = bm_new.band_id
                                AND  bm_old.artist_id = pa.artist_id))
        )
    THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Artists do not share a common band.';
    END IF;
END;

-- 4. AFTER INSERT on Performance_Artist – when every member of some band is now present
-- insert that band into Performance_Band (rule 5)
CREATE TRIGGER trg_auto_assign_band_after_artist_ins
AFTER INSERT ON Performance_Artist
FOR EACH ROW
BEGIN
    -- all DECLAREs first
    DECLARE target_band INT;

    -- rest of the logic
    IF (SELECT COUNT(*) FROM Performance_Band WHERE perf_id = NEW.perf_id) = 0 THEN
        /* find a band whose every member is now present */
        SELECT bm.band_id INTO target_band
        FROM   Band_Member bm
        WHERE  bm.artist_id = NEW.artist_id
        GROUP BY bm.band_id
        HAVING COUNT(*) = (
                    SELECT COUNT(*)                -- #members of that band
                    FROM Band_Member
                    WHERE band_id = bm.band_id)
            AND COUNT(*) = (
                    SELECT COUNT(DISTINCT artist_id)
                    FROM   Performance_Artist
                    WHERE  perf_id = NEW.perf_id)
        LIMIT 1;

        IF target_band IS NOT NULL THEN
            INSERT IGNORE INTO Performance_Band (perf_id, band_id)
            VALUES (NEW.perf_id, target_band);
```

```
        END IF;
    END IF;
END;

-- ============================================================
-- 2. Scheduling & participation (time / dates)
-- ============================================================

-- 5. Same artist cannot play two stages at the same time
CREATE TRIGGER trg_no_double_stage_artist
BEFORE INSERT ON Performance_Artist
FOR EACH ROW
BEGIN
    DECLARE d DATETIME; DECLARE s INT;
    SELECT datetime, stage_id INTO d, s FROM Performance WHERE perf_id = NEW.perf_id;
    IF EXISTS (
        SELECT 1
        FROM   Performance p
            JOIN Performance_Artist pa ON p.perf_id = pa.perf_id
        WHERE  pa.artist_id = NEW.artist_id
            AND  p.datetime  = d
            AND  p.stage_id <> s )
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Artist already on another stage at
that time.';
    END IF;
END;

-- 6. Same band cannot play two stages at the same time
CREATE TRIGGER trg_no_double_stage_band
BEFORE INSERT ON Performance_Band
FOR EACH ROW
BEGIN
    DECLARE d DATETIME; DECLARE s INT;
    SELECT datetime, stage_id INTO d, s FROM Performance WHERE perf_id = NEW.perf_id;
    IF EXISTS (
        SELECT 1
        FROM   Performance p
            JOIN Performance_Band pb ON p.perf_id = pb.perf_id
        WHERE  pb.band_id = NEW.band_id
            AND  p.datetime = d
            AND  p.stage_id <> s )
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Band already on another stage at that
time.';
    END IF;
END;

-- 7. One performance per stage at any moment on insert
CREATE TRIGGER trg_no_stage_overlap
BEFORE INSERT ON Performance
FOR EACH ROW
BEGIN
    IF EXISTS (
```

```
        SELECT 1
        FROM   Performance
        WHERE  stage_id = NEW.stage_id
            AND  perf_id <> NEW.perf_id
            AND  NEW.datetime < ADDDATE(datetime, INTERVAL duration MINUTE)
            AND  ADDDATE(NEW.datetime, INTERVAL NEW.duration MINUTE) > datetime
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Stage already booked for that time-slot.';
    END IF;
END;

-- 8. Check if an artist has performed for more than 3 consecutive years
CREATE TRIGGER trg_max_consecutive_years_artist
BEFORE INSERT ON Performance_Artist
FOR EACH ROW
BEGIN
    DECLARE perf_year INT;

    -- Year of the performance we're inserting
    SELECT f.fest_year
        INTO perf_year
        FROM Performance p
        JOIN Event     e ON p.event_id = e.event_id
        JOIN Festival  f ON e.fest_year = f.fest_year
        WHERE p.perf_id = NEW.perf_id;

    -- Past years + the pending year for this artist
    IF EXISTS (
        WITH years AS (
            SELECT DISTINCT f.fest_year AS y
                FROM Performance        p
                JOIN Event              e ON e.event_id   = p.event_id
                JOIN Festival           f ON f.fest_year  = e.fest_year
                WHERE EXISTS (
                        SELECT 1 FROM Performance_Artist pa
                        WHERE pa.perf_id = p.perf_id
                        AND pa.artist_id = NEW.artist_id
                    )
            UNION ALL
            SELECT perf_year
        ),
        seq AS (
            SELECT y,
                    ROW_NUMBER() OVER (ORDER BY y) AS rn
                FROM years
        ),
        runs AS (
            SELECT COUNT(*) AS run_len
                FROM seq
                GROUP BY y - rn          -- islands-and-gaps trick
        )
        SELECT 1 FROM runs WHERE run_len >= 4
    ) THEN
```

```
264         SIGNAL SQLSTATE '45000'
265         SET MESSAGE_TEXT =
266             'Artist exceeds the 3-year consecutive performance limit';
267     END IF;
268 END;
269 
270 -- 9. Check if any of the members of the band has performed for more than 3 consecutive
    years => don't insert the band
271 CREATE TRIGGER trg_max_consecutive_years_band
272 BEFORE INSERT ON Performance_Band
273 FOR EACH ROW
274 BEGIN
275     DECLARE perf_year INT;
276 
277     -- Year of the performance we are inserting
278     SELECT f.fest_year
279         INTO perf_year
280         FROM Performance p
281         JOIN Event     e ON p.event_id = e.event_id
282         JOIN Festival  f ON e.fest_year = f.fest_year
283         WHERE p.perf_id = NEW.perf_id;
284 
285     -- Build a set of {artist_id, year} rows:
286         -- every past year each band member has performed
287         -- plus the year we are about to add for each artist look for a run of 4
    consecutive years.
288     IF EXISTS (
289         WITH member_years AS (
290             SELECT bm.artist_id,
291                     f.fest_year AS y
292                 FROM Band_Member bm
293                 JOIN Performance_Artist pa ON pa.artist_id = bm.artist_id
294                 JOIN Performance        p  ON p.perf_id    = pa.perf_id
295                 JOIN Event              e  ON e.event_id   = p.event_id
296                 JOIN Festival           f  ON f.fest_year  = e.fest_year
297                 WHERE bm.band_id = NEW.band_id
298 
299             UNION ALL
300 
301             SELECT bm.artist_id, perf_year
302                 FROM Band_Member bm
303                 WHERE bm.band_id = NEW.band_id
304         ),
305         seq AS (
306             SELECT artist_id,
307                     y,
308                     ROW_NUMBER() OVER (PARTITION BY artist_id ORDER BY y) AS rn
309                 FROM (SELECT DISTINCT artist_id, y FROM member_years) x
310         ),
311         runs AS (
312             SELECT artist_id,
313                     COUNT(*) AS run_len
314                 FROM seq
315                 GROUP BY artist_id, y - rn         -- islands-and-gaps trick
```

```
316         )
317         SELECT 1 FROM runs WHERE run_len >= 4
318     ) THEN
319         SIGNAL SQLSTATE '45000'
320         SET MESSAGE_TEXT =
321             'One or more band members exceed the 3-year consecutive limit';
322     END IF;
323 END;
324 
325 -- 10. An event must start only in the days of the festival
326 CREATE TRIGGER trg_event_within_festival_dates
327 BEFORE INSERT ON Event
328 FOR EACH ROW
329 BEGIN
330     DECLARE s DATE; DECLARE e DATE;
331     SELECT start_date, end_date INTO s, e
332     FROM   Festival
333     WHERE  fest_year = NEW.fest_year;
334 
335     IF DATE(NEW.start_dt) < s OR DATE(NEW.start_dt) > e THEN
336         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Event dates outside festival.';
337     END IF;
338 END;
339 
340 -- 11. Performance must fit inside its parent event window
341 CREATE TRIGGER trg_performance_inside_event
342 BEFORE INSERT ON Performance
343 FOR EACH ROW
344 BEGIN
345     DECLARE eStart DATETIME; DECLARE eEnd DATETIME;
346     SELECT start_dt, end_dt INTO eStart, eEnd
347     FROM   Event
348     WHERE  event_id = NEW.event_id;
349 
350     IF NEW.datetime < eStart
351         OR ADDDATE(NEW.datetime, INTERVAL NEW.duration MINUTE) > eEnd THEN
352         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Performance outside its event
    window.';
353     END IF;
354 END;
355 
356 -- 12. Allow change of dates of festival, only if events are still in bounds
357 CREATE TRIGGER trg_safe_festival_date_update
358 BEFORE UPDATE ON Festival
359 FOR EACH ROW
360 BEGIN
361     -- Only run check if dates are changing
362     IF NEW.start_date <> OLD.start_date OR NEW.end_date <> OLD.end_date THEN
363         -- Block if any event for this festival falls outside new bounds
364         IF EXISTS (
365             SELECT 1
366             FROM   Event
367             WHERE  fest_year = NEW.fest_year
368                 AND (DATE(start_dt) < NEW.start_date OR DATE(start_dt) > NEW.end_date)
```

```
        ) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Cannot change festival dates: some events would be out of bounds.';
        END IF;
    END IF;
END;

-- 13. Allow change of dates of event, only if performances are still in bounds
CREATE TRIGGER trg_safe_event_date_update
BEFORE UPDATE ON Event
FOR EACH ROW
BEGIN
    -- Only proceed if event window changes
    IF NEW.start_dt <> OLD.start_dt OR NEW.end_dt <> OLD.end_dt THEN
        -- Block update if any performance starts before or ends after the new window
        IF EXISTS (
            SELECT 1
            FROM   Performance
            WHERE  event_id = NEW.event_id
                AND (
                    datetime < NEW.start_dt
                OR ADDTIME(datetime, SEC_TO_TIME(duration * 60)) > NEW.end_dt
                )
        ) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Cannot update event dates: some performances would fall outside the new window.';
        END IF;
    END IF;
END;

-- ============================================================
-- 3. Band membership clean-up
-- ============================================================

-- 14. Delete band when it has no members
CREATE TRIGGER trg_delete_band_if_no_members
AFTER DELETE ON Band_Member
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM Band_Member WHERE band_id = OLD.band_id) = 0 THEN
        DELETE FROM Band WHERE band_id = OLD.band_id;
    END IF;
END;

-- ============================================================
-- 4. Staffing ratio (≥5 % security, ≥2 % support)
-- ============================================================

-- 15. Check ratios after DELETE on Works_On
CREATE TRIGGER trg_staff_ratio_after_delete
AFTER DELETE ON Works_On
FOR EACH ROW
```

```sql
BEGIN
    CALL check_staff_ratio(OLD.event_id);
END;

-- 16. check ratios after UPDATE on Works_On
CREATE TRIGGER trg_staff_ratio_after_update
AFTER UPDATE ON Works_On
FOR EACH ROW
BEGIN
    IF OLD.event_id <> NEW.event_id THEN
        CALL check_staff_ratio(OLD.event_id);
        CALL check_staff_ratio(NEW.event_id);
    ELSE
        CALL check_staff_ratio(NEW.event_id);
    END IF;
END;

-- ============================================================
-- 5. Ticket capacity / selling
-- ============================================================

-- 17. Block overselling and mark event full (all statuses)
CREATE TRIGGER trg_ticket_capacity_check
BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
    DECLARE cap INT;
    DECLARE sold INT;

    -- Capacity of event's stage
    SELECT s.capacity INTO cap
    FROM   Event e
    JOIN   Stage s ON e.stage_id = s.stage_id
    WHERE  e.event_id = NEW.event_id;

    -- Count all tickets for this event, regardless of status
    SELECT COUNT(*) INTO sold
    FROM   Ticket
    WHERE  event_id = NEW.event_id;

    -- Block if full
    IF sold >= cap THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Capacity reached for this event.';
    END IF;

    -- If this insert fills the event exactly, mark it as full
    IF sold + 1 = cap THEN
        UPDATE Event SET is_full = TRUE WHERE event_id = NEW.event_id;
    END IF;
END;

-- 18. VIP tickets ≤10 % of capacity
CREATE TRIGGER trg_check_vip_ticket_limit
BEFORE INSERT ON Ticket
```

```sql
FOR EACH ROW
BEGIN
    DECLARE cap INT; DECLARE vip INT; DECLARE vipType INT;

    SELECT type_id INTO vipType FROM Ticket_Type WHERE name='VIP' LIMIT 1;

    SELECT s.capacity INTO cap
    FROM   Event e
           JOIN Stage s ON e.stage_id = s.stage_id
    WHERE  e.event_id = NEW.event_id;

    SELECT COUNT(*) INTO vip
    FROM   Ticket
    WHERE  event_id = NEW.event_id
        AND type_id  = vipType;

    IF NEW.type_id = vipType AND vip >= CEIL(cap * 0.10) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'VIP ticket cap exceeded.';
    END IF;
END;

-- 19. Confirm EAN format on insert
CREATE TRIGGER trg_validate_ticket_ean
BEFORE INSERT ON Ticket
FOR EACH ROW
BEGIN
    DECLARE ean_str CHAR(13);
    DECLARE sum INT DEFAULT 0;
    DECLARE i INT DEFAULT 1;
    DECLARE digit INT;
    DECLARE checksum INT;

    -- Convert to 13-digit string
    SET ean_str = LPAD(NEW.ean_number, 13, '0');

    -- Must be 13 numeric characters
    IF ean_str NOT REGEXP '^[0-9]{13}$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'EAN must be a 13-digit number.';
    END IF;

    -- Calculate EAN-13 checksum: use digits 1–12
    WHILE i <= 12 DO
        SET digit = CAST(SUBSTRING(ean_str, i, 1) AS UNSIGNED);
        SET sum = sum + digit * IF(MOD(i, 2) = 0, 3, 1);
        SET i = i + 1;
    END WHILE;

    SET checksum = (10 - (sum MOD 10)) MOD 10;

    -- Validate against 13th digit
    IF checksum <> CAST(SUBSTRING(ean_str, 13, 1) AS UNSIGNED) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid EAN-13 checksum.';
```

```
    END IF;
END;

-- 20. Confirm EAN format on update
CREATE TRIGGER trg_validate_ticket_ean_upd
BEFORE UPDATE ON Ticket
FOR EACH ROW
BEGIN
    DECLARE ean_str CHAR(13);
    DECLARE sum INT DEFAULT 0;
    DECLARE i INT DEFAULT 1;
    DECLARE digit INT;
    DECLARE checksum INT;

    -- Convert to 13-digit string
    SET ean_str = LPAD(NEW.ean_number, 13, '0');

    -- Must be 13 numeric characters
    IF ean_str NOT REGEXP '^[0-9]{13}$' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'EAN must be a 13-digit number.';
    END IF;

    -- Calculate EAN-13 checksum: use digits 1–12
    WHILE i <= 12 DO
        SET digit = CAST(SUBSTRING(ean_str, i, 1) AS UNSIGNED);
        SET sum = sum + digit * IF(MOD(i, 2) = 0, 3, 1);
        SET i = i + 1;
    END WHILE;

    SET checksum = (10 - (sum MOD 10)) MOD 10;

    -- Validate against 13th digit
    IF checksum <> CAST(SUBSTRING(ean_str, 13, 1) AS UNSIGNED) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid EAN-13 checksum.';
    END IF;
END;

-- ============================================================
-- 6. Resale & ticket status
-- ============================================================

-- 21. Only ACTIVE tickets can be offered for resale
CREATE TRIGGER trg_resale_offer_only_active
BEFORE INSERT ON Resale_Offer
FOR EACH ROW
BEGIN
    IF (SELECT status_id FROM Ticket WHERE ticket_id = NEW.ticket_id) <>
        (SELECT status_id FROM Ticket_Status WHERE name='active' LIMIT 1) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Ticket not eligible for resale.';
    END IF;
END;

```

```
583 -- 22. Review allowed only with USED ticket
584 CREATE TRIGGER trg_review_only_with_used_ticket
585 BEFORE INSERT ON Review
586 FOR EACH ROW
587 BEGIN
588     IF NOT EXISTS (
589         SELECT 1
590         FROM   Ticket t
591                 JOIN Performance p ON p.event_id = t.event_id
592         WHERE  t.attendee_id = NEW.attendee_id
593             AND p.perf_id     = NEW.perf_id
594             AND t.status_id   = (SELECT status_id FROM Ticket_Status WHERE name='used'
    LIMIT 1)
595     ) THEN
596         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Must have USED ticket to review.';
597     END IF;
598 END;
599
600 -- 23. Protect resale-offer timestamp
601 CREATE TRIGGER trg_resale_offer_timestamp
602 BEFORE UPDATE ON Resale_Offer
603 FOR EACH ROW
604 BEGIN
605     IF NEW.offer_timestamp <> OLD.offer_timestamp THEN
606         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'offer_timestamp is immutable.';
607     END IF;
608 END;
609
610 --24. Protect resale-interest timestamp
611 CREATE TRIGGER trg_resale_interest_timestamp
612 BEFORE UPDATE ON Resale_Interest
613 FOR EACH ROW
614 BEGIN
615     IF NEW.interest_timestamp <> OLD.interest_timestamp THEN
616         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'interest_timestamp is immutable.';
617     END IF;
618 END;
619
620 -- 25. Allow capacity to only be increased and mark full events as not full.
621 CREATE TRIGGER trg_stage_capacity_update
622 BEFORE UPDATE ON Stage
623 FOR EACH ROW
624 BEGIN
625     -- 1. Block capacity decrease
626     IF NEW.capacity < OLD.capacity THEN
627         SIGNAL SQLSTATE '45000'
628         SET MESSAGE_TEXT = 'Cannot decrease stage capacity.';
629     END IF;
630
631     -- 2. If capacity is increased, unmark full events on this stage
632     IF NEW.capacity > OLD.capacity THEN
633         UPDATE Event
634         SET    is_full = FALSE
635         WHERE  stage_id = NEW.stage_id
```

```
636             AND  is_full = TRUE;
637     END IF;
638 END;
639 
640 -- 26. Prevent purchase data of ticket to change
641 CREATE TRIGGER trg_block_purchase_date_update
642 BEFORE UPDATE ON Ticket
643 FOR EACH ROW
644 BEGIN
645     IF NEW.purchase_date <> OLD.purchase_date THEN
646         SIGNAL SQLSTATE '45000'
647         SET MESSAGE_TEXT = 'purchase_date cannot be modified after insertion.';
648     END IF;
649 END;
650 
651 -- 27. Prevent ticket to be purchased after the starte date of the event
652 CREATE TRIGGER trg_validate_ticket_purchase_date
653 BEFORE INSERT ON Ticket
654 FOR EACH ROW
655 BEGIN
656     DECLARE ev_start DATETIME;
657 
658     SELECT start_dt INTO ev_start
659     FROM   Event
660     WHERE  event_id = NEW.event_id;
661 
662     IF NEW.purchase_date >= ev_start THEN
663         SIGNAL SQLSTATE '45000'
664         SET MESSAGE_TEXT = 'Ticket must be purchased before the event starts.';
665     END IF;
666 END;
667 
668 -- ============================================================
669 -- 7. Genre / sub-genre consistency
670 -- ============================================================
671 
672 -- 28. Artist sub-genre must match an artist genre
673 CREATE TRIGGER trg_artist_subgenre_consistency
674 BEFORE INSERT ON Artist_SubGenre
675 FOR EACH ROW
676 BEGIN
677     IF NOT EXISTS (
678         SELECT 1
679         FROM   SubGenre sg
680                JOIN Artist_Genre ag ON ag.genre_id = sg.genre_id
681         WHERE  sg.sub_genre_id = NEW.sub_genre_id
682             AND ag.artist_id    = NEW.artist_id
683     ) THEN
684         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Artist sub-genre inconsistent.';
685     END IF;
686 END;
687 
688 -- 29. Band sub-genre must match a band genre
689 CREATE TRIGGER trg_band_subgenre_consistency
```

```
BEFORE INSERT ON Band_SubGenre
FOR EACH ROW
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM   SubGenre sg
                JOIN Band_Genre bg ON bg.genre_id = sg.genre_id
        WHERE  sg.sub_genre_id = NEW.sub_genre_id
            AND bg.band_id      = NEW.band_id
    ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Band sub-genre inconsistent.';
    END IF;
END;

-- ============================================================
-- 8. Cascade clean-ups
-- ============================================================

-- 30. Delete attendee ⇒ purge tickets / resale / reviews
CREATE TRIGGER trg_delete_attendee_cleanup
AFTER DELETE ON Attendee
FOR EACH ROW
BEGIN
    DELETE FROM Ticket          WHERE attendee_id = OLD.attendee_id;
    DELETE FROM Resale_Offer    WHERE seller_id   = OLD.attendee_id;
    DELETE FROM Resale_Interest WHERE buyer_id    = OLD.attendee_id;
    DELETE FROM Review          WHERE attendee_id = OLD.attendee_id;
END;

-- ============================================================
-- 9. Auto-match offers
-- ============================================================

-- 31. Match one resale interest with the FIFO offer queue (auto-buy ticket if seller
exists)
CREATE TRIGGER trg_match_resale_interest
AFTER INSERT ON Resale_Interest_Type
FOR EACH ROW
BEGIN
    DECLARE v_event   INT;
    DECLARE v_type    INT;
    DECLARE v_ticket  INT;
    DECLARE v_offer   INT;
    DECLARE v_buyer   INT;
    DECLARE v_seller  INT;
    DECLARE sActive   INT;

    -- lookup status id for 'active'
    SELECT status_id INTO sActive
    FROM Ticket_Status WHERE name = 'active' LIMIT 1;

    -- lookup event and buyer
    SELECT event_id, buyer_id
    INTO v_event, v_buyer
```

```
    FROM Resale_Interest
    WHERE request_id = NEW.request_id;

    -- type requested
    SET v_type = NEW.type_id;

    -- find first matching offer
    SELECT ro.offer_id, ro.ticket_id, ro.seller_id
    INTO v_offer, v_ticket, v_seller
    FROM Resale_Offer ro
    JOIN Ticket t ON t.ticket_id = ro.ticket_id
    WHERE ro.event_id = v_event AND t.type_id = v_type
    ORDER BY ro.offer_timestamp
    LIMIT 1;

    -- if match found, reassign ticket + log match
    IF v_offer IS NOT NULL THEN
        UPDATE Ticket
        SET attendee_id = v_buyer,
            status_id   = sActive
        WHERE ticket_id = v_ticket;

        DELETE FROM Resale_Offer
        WHERE offer_id = v_offer;

        DELETE FROM Resale_Interest
        WHERE request_id = NEW.request_id;

        INSERT INTO Resale_Match_Log (
            match_type, ticket_id, offered_type_id, requested_type_id,
            buyer_id, seller_id
        )
        VALUES ('interest', v_ticket, v_type, v_type, v_buyer, v_seller);
    END IF;
END;

-- 32. Match one resale offer with the FIFO interest queue (auto-sell ticket if buyer
exists)
CREATE TRIGGER trg_match_resale_offer
AFTER INSERT ON Resale_Offer
FOR EACH ROW
BEGIN
    DECLARE v_event   INT;
    DECLARE v_type    INT;
    DECLARE v_req     INT;
    DECLARE v_buyer   INT;
    DECLARE sActive   INT;

    -- lookup status id for 'active'
    SELECT status_id INTO sActive
    FROM Ticket_Status WHERE name = 'active' LIMIT 1;

    -- get event id from offer
    SET v_event = NEW.event_id;
```

```
796
797     -- lookup type of the offered ticket
798     SELECT type_id INTO v_type
799     FROM Ticket WHERE ticket_id = NEW.ticket_id;
800
801     -- find first matching interest
802     SELECT ri.request_id, ri.buyer_id
803     INTO v_req, v_buyer
804     FROM Resale_Interest ri
805     JOIN Resale_Interest_Type rit ON rit.request_id = ri.request_id
806     WHERE ri.event_id = v_event AND rit.type_id = v_type
807     ORDER BY ri.interest_timestamp
808     LIMIT 1;
809
810     -- if match found, reassign ticket + log match
811     IF v_req IS NOT NULL THEN
812         UPDATE Ticket
813         SET attendee_id = v_buyer,
814             status_id   = sActive
815         WHERE ticket_id = NEW.ticket_id;
816
817         DELETE FROM Resale_Interest
818         WHERE request_id = v_req;
819
820         INSERT INTO Resale_Match_Log (
821             match_type, ticket_id, offered_type_id, requested_type_id,
822             buyer_id, seller_id
823         )
824         VALUES ('offer', NEW.ticket_id, v_type, v_type, v_buyer, NEW.seller_id);
825     END IF;
826 END;
827
828 -- 33. Event date
829 CREATE TRIGGER trg_set_event_date
830 BEFORE INSERT ON Event
831 FOR EACH ROW
832     SET NEW.generated_date = CAST(NEW.start_dt AS DATE);
833
```

sql\procedures.sql

```sql
-- --------------
-- Procedures --
-- --------------

USE pulse_university;

-- Drop all procedures
DROP PROCEDURE IF EXISTS UpdateExpiredTickets;
DROP PROCEDURE IF EXISTS ExpireResaleOffers;
DROP PROCEDURE IF EXISTS ExpireResaleInterests;
DROP PROCEDURE IF EXISTS ScanTicket;
DROP PROCEDURE IF EXISTS RunMaintenance;
DROP PROCEDURE IF EXISTS sp_rename_self;
DROP PROCEDURE IF EXISTS check_staff_ratio;

-- ============================================================
-- Procedure 1: Update expired tickets (active / on offer → unused)
-- ============================================================
CREATE PROCEDURE UpdateExpiredTickets()
BEGIN
    DECLARE sActive  INT;
    DECLARE sOffer   INT;
    DECLARE sUnused  INT;

    SELECT status_id INTO sActive  FROM Ticket_Status WHERE name = 'active'   LIMIT 1;
    SELECT status_id INTO sOffer   FROM Ticket_Status WHERE name = 'on offer' LIMIT 1;
    SELECT status_id INTO sUnused  FROM Ticket_Status WHERE name = 'unused'   LIMIT 1;

    UPDATE Ticket t
        JOIN Event e ON t.event_id = e.event_id
    SET t.status_id = sUnused
    WHERE CURDATE() > e.end_dt
        AND t.status_id IN (sActive, sOffer);
END;

-- ============================================================
-- Procedure 2: Expire resale offers after event end
-- ============================================================
CREATE PROCEDURE ExpireResaleOffers()
BEGIN
    DELETE ro
    FROM   Resale_Offer ro
            JOIN Event e ON ro.event_id = e.event_id
    WHERE  CURDATE() > e.end_dt;
END;

-- ============================================================
-- Procedure 3: Expire resale interests after event end
-- ============================================================
CREATE PROCEDURE ExpireResaleInterests()
BEGIN
```

```
    DELETE ri
    FROM   Resale_Interest ri
           JOIN Event e ON ri.event_id = e.event_id
    WHERE  CURDATE() > e.end_dt;
END;

-- ============================================================
-- Procedure 4: Scan ticket at entrance (active → used)
-- ============================================================
CREATE PROCEDURE ScanTicket(IN p_ean BIGINT)
BEGIN
    DECLARE v_status INT;
    DECLARE sActive  INT;
    DECLARE sUsed    INT;

    SELECT status_id INTO sActive FROM Ticket_Status WHERE name = 'active' LIMIT 1;
    SELECT status_id INTO sUsed   FROM Ticket_Status WHERE name = 'used'   LIMIT 1;

    SELECT status_id INTO v_status
    FROM   Ticket
    WHERE  ean_number = p_ean;

    IF v_status IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Ticket not found.';
    ELSEIF v_status <> sActive THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Ticket already used or inactive.';
    ELSE
        UPDATE Ticket
        SET    status_id = sUsed
        WHERE  ean_number = p_ean;
    END IF;
END;

-- ============================================================================
-- Procedure 5: Run maintenance (expire tickets/offers/interests + ratios)
-- ============================================================================
CREATE PROCEDURE RunMaintenance()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE ev_id INT;

    -- cursor must be before handler
    DECLARE cur CURSOR FOR
        SELECT DISTINCT event_id FROM Works_On;

    -- handler after cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- 1. Run expiration subprocedures
    CALL UpdateExpiredTickets();
    CALL ExpireResaleOffers();
    CALL ExpireResaleInterests();
```

```
106
107     -- 2. Loop through all events to validate staff ratios
108     OPEN cur;
109     read_loop: LOOP
110         FETCH cur INTO ev_id;
111         IF done THEN
112             LEAVE read_loop;
113         END IF;
114
115         BEGIN
116             DECLARE ratio_error CONDITION FOR SQLSTATE '45000';
117             DECLARE CONTINUE HANDLER FOR ratio_error
118                 SELECT CONCAT('⚠ Event ', ev_id, ' fails staff ratio check') AS warning;
119
120             CALL check_staff_ratio(ev_id);
121         END;
122     END LOOP;
123     CLOSE cur;
124 END;
125
126 -- ============================================================
127 -- Procedure 6: Rename the currently connected user (self)
128 -- ============================================================
129 CREATE PROCEDURE sp_rename_self(IN new_name VARCHAR(64))
130 SQL SECURITY DEFINER
131 BEGIN
132     DECLARE cur_user VARCHAR(64);
133     SET cur_user = SUBSTRING_INDEX(USER(), '@', 1);
134     SET @sql = CONCAT(
135         'ALTER USER `', cur_user, '`@''%'' RENAME TO `', new_name, '`@''%'';'
136     );
137     PREPARE stmt FROM @sql;
138     EXECUTE stmt;
139     DEALLOCATE PREPARE stmt;
140 END;
141
142 -- ============================================================
143 -- Procedure 7: Check staff ratios (≥5% security, ≥2% support)
144 -- ============================================================
145 DROP PROCEDURE IF EXISTS check_staff_ratio;
146 CREATE PROCEDURE check_staff_ratio(IN ev INT)
147 BEGIN
148     DECLARE cap INT DEFAULT NULL;
149     DECLARE sec INT;
150     DECLARE sup INT;
151
152     -- Get the capacity of the event's stage
153     SELECT s.capacity INTO cap
154     FROM   Event e
155     JOIN   Stage s ON e.stage_id = s.stage_id
156     WHERE  e.event_id = ev
157     LIMIT 1;
158
159     -- If event not found, raise error
```

```sql
    IF cap IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid event ID: no matching event found.';
    END IF;

    -- Count security staff
    SELECT COUNT(*) INTO sec
    FROM   Works_On wo
    JOIN   Staff st ON wo.staff_id = st.staff_id
    WHERE  wo.event_id = ev
        AND  st.role_id = (
        SELECT role_id FROM Staff_Role WHERE name = 'security' LIMIT 1
    );

    -- Count support staff
    SELECT COUNT(*) INTO sup
    FROM   Works_On wo
    JOIN   Staff st ON wo.staff_id = st.staff_id
    WHERE  wo.event_id = ev
        AND  st.role_id = (
        SELECT role_id FROM Staff_Role WHERE name = 'support' LIMIT 1
    );

    -- Check if staffing ratios are met
    IF sec < CEIL(cap * 0.05) OR sup < CEIL(cap * 0.02) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Staffing below required ratio.';
    END IF;
END;

```

sql\views.sql

```sql
-- --------------
-- --- Views ----
-- --------------

USE pulse_university;

/* ============  drop old versions if they exist  ============ */
DROP VIEW IF EXISTS View_Yearly_Revenue_By_Method;
DROP VIEW IF EXISTS View_Artist_Year_Participation;
DROP VIEW IF EXISTS View_Performance_Detail;
DROP VIEW IF EXISTS View_Artist_Performance_Rating;
DROP VIEW IF EXISTS View_Attendee_Event_Rating;
DROP VIEW IF EXISTS View_Attendee_Yearly_Visits;
DROP VIEW IF EXISTS View_Genre_Pairs;
DROP VIEW IF EXISTS View_Artist_Continents;
DROP VIEW IF EXISTS View_Genre_Year_Counts;
DROP VIEW IF EXISTS View_Attendee_Artist_Review;

/* ------------------------------------------------------------
 * View 1 – yearly ticket revenues by payment method (Q 1)
 * ------------------------------------------------------------*/
CREATE VIEW View_Yearly_Revenue_By_Method AS
SELECT
    e.fest_year,
    t.method_id,
    SUM(t.cost) AS total_revenue
FROM Ticket t
JOIN Event e ON t.event_id = e.event_id
GROUP BY e.fest_year, t.method_id;

/* ------------------------------------------------------------
 * View 2 –  artist participation by year (Q 2)
 * ------------------------------------------------------------*/
CREATE VIEW View_Artist_Year_Participation AS
SELECT DISTINCT
    pa.artist_id,
    e.fest_year
FROM Performance_Artist pa
JOIN Performance p ON pa.perf_id = p.perf_id
JOIN Event e ON p.event_id = e.event_id;

/* ------------------------------------------------------------
 * View 3 – handy performance detail (Q 3)
 *          one row per performer-performance, with fest year & type
 * ------------------------------------------------------------*/
CREATE VIEW View_Performance_Detail AS
SELECT p.perf_id,
        p.event_id,
        e.fest_year,
        pt.name AS perf_type,
        p.datetime,
```

```
        pa.artist_id,
        pb.band_id
FROM Performance p
JOIN      Event              e  ON p.event_id = e.event_id
JOIN      Performance_Type   pt ON p.type_id  = pt.type_id
LEFT JOIN Performance_Artist pa ON pa.perf_id = p.perf_id
LEFT JOIN Performance_Band   pb ON pb.perf_id = p.perf_id;

/* ------------------------------------------------------------
 * View 4 – artist workload & average ratings (Q 4, 11)
 * ------------------------------------------------------------*/
CREATE VIEW View_Artist_Performance_Rating AS
SELECT
    a.artist_id,
    CONCAT(a.first_name, ' ', a.last_name) AS artist_name,
    (
        SELECT COUNT(DISTINCT pa.perf_id)
        FROM Performance_Artist pa
        WHERE pa.artist_id = a.artist_id
    ) AS performance_count,
    (
        SELECT AVG(r.interpretation)
        FROM Performance_Artist pa
        JOIN Review r ON pa.perf_id = r.perf_id
        WHERE pa.artist_id = a.artist_id
    ) AS avg_interpretation,
    (
        SELECT AVG(r.overall)
        FROM Performance_Artist pa
        JOIN Review r ON pa.perf_id = r.perf_id
        WHERE pa.artist_id = a.artist_id
    ) AS avg_overall
FROM Artist a;

/* ------------------------------------------------------------
 * View 5 – attendee's watched performances & own rating (Q 6)
 * ------------------------------------------------------------*/
CREATE VIEW View_Attendee_Event_Rating AS
SELECT
    att.attendee_id,
    CONCAT(att.first_name, ' ', att.last_name) AS attendee_name,
    e.event_id,
    e.title AS event_title,
    AVG(r.overall) AS avg_event_rating
FROM Ticket t
JOIN Event       e   ON t.event_id      = e.event_id
JOIN Attendee    att ON att.attendee_id = t.attendee_id
JOIN Performance p   ON p.event_id      = e.event_id
LEFT JOIN Review r   ON r.perf_id       = p.perf_id
                    AND r.attendee_id   = att.attendee_id
GROUP BY att.attendee_id, attendee_name, e.event_id, e.title;

/* ------------------------------------------------------------
 * View 6 – attendee visit count per calendar year (Q 9)
```

```sql
 * ------------------------------------------------------------*/
CREATE VIEW View_Attendee_Yearly_Visits AS
SELECT  t.attendee_id,
        YEAR(t.purchase_date)      AS festival_year,
        COUNT(DISTINCT t.event_id) AS events_attended
FROM    Ticket t
GROUP BY t.attendee_id, YEAR(t.purchase_date);

/* ------------------------------------------------------------
 * View 7 – unique artist-genre pairs & artist count (Q 10)
 *          only for artist that actually have performed
 * ------------------------------------------------------------*/
CREATE VIEW View_Genre_Pairs AS
SELECT
    LEAST(ag1.genre_id, ag2.genre_id)    AS genre_id1,
    GREATEST(ag1.genre_id, ag2.genre_id) AS genre_id2,
    COUNT(DISTINCT pa.perf_id)           AS performance_count
FROM Performance_Artist pa
JOIN Artist_Genre ag1 ON pa.artist_id = ag1.artist_id
JOIN Artist_Genre ag2 ON pa.artist_id = ag2.artist_id AND ag1.genre_id < ag2.genre_id
JOIN Performance p    ON pa.perf_id = p.perf_id
JOIN Event e          ON p.event_id = e.event_id
JOIN Festival f       ON e.fest_year = f.fest_year
WHERE f.fest_year < YEAR(CURDATE())
GROUP BY genre_id1, genre_id2;

/* ------------------------------------------------------------
 * View 8 – artists & number of continents performed in (Q 13)
 * ------------------------------------------------------------*/
CREATE VIEW View_Artist_Continents AS
SELECT  a.artist_id,
        CONCAT(a.first_name,' ',a.last_name) AS artist_name,
        COUNT(DISTINCT l.continent_id)       AS continents_performed
FROM    Artist a
JOIN    Performance_Artist pa ON a.artist_id = pa.artist_id
JOIN    Performance        p  ON pa.perf_id  = p.perf_id
JOIN    Event              e  ON p.event_id  = e.event_id
JOIN    Festival           f  ON e.fest_year = f.fest_year
JOIN    Location           l  ON f.loc_id    = l.loc_id
GROUP BY a.artist_id, artist_name;

/* ------------------------------------------------------------
 * View 9 – yearly appearances per genre (Q 14)
 * ------------------------------------------------------------*/
CREATE VIEW View_Genre_Year_Counts AS
SELECT  g.genre_id,
        g.name   AS genre_name,
        d.fest_year,
        COUNT(*) AS perf_count
FROM    View_Performance_Detail d
JOIN    Artist_Genre ag ON ag.artist_id = d.artist_id
JOIN    Genre        g  ON g.genre_id   = ag.genre_id
GROUP BY g.genre_id, g.name, d.fest_year;

```

```sql
/* ------------------------------------------------------------
 * View 10 – aggregates per attendee-artist pair (Q 15)
 * ------------------------------------------------------------*/
CREATE VIEW View_Attendee_Artist_Review AS
SELECT
    r.attendee_id,
    pa.artist_id,
    SUM(r. interpretation + r.overall) AS total_score
FROM Review r
JOIN Performance_Artist pa ON r.perf_id = pa.perf_id
GROUP BY r.attendee_id, pa.artist_id;

```