

## sql\install.sql

```
1  -- Drop and recreate the database
2  DROP DATABASE IF EXISTS pulse_university;
3  CREATE DATABASE IF NOT EXISTS pulse_university;
4  USE pulse_university;
5
6  -- Drop all tables
7  DROP TABLE IF EXISTS Continent;
8  DROP TABLE IF EXISTS Staff_Role;
9  DROP TABLE IF EXISTS Experience_Level;
10 DROP TABLE IF EXISTS Performance_Type;
11 DROP TABLE IF EXISTS Ticket_Type;
12 DROP TABLE IF EXISTS Payment_Method;
13 DROP TABLE IF EXISTS Ticket_Status;
14 DROP TABLE IF EXISTS Genre;
15 DROP TABLE IF EXISTS SubGenre;
16 DROP TABLE IF EXISTS Location;
17 DROP TABLE IF EXISTS Festival;
18 DROP TABLE IF EXISTS Stage;
19 DROP TABLE IF EXISTS Equipment;
20 DROP TABLE IF EXISTS Stage_Equipment;
21 DROP TABLE IF EXISTS Event;
22 DROP TABLE IF EXISTS Staff;
23 DROP TABLE IF EXISTS Works_On;
24 DROP TABLE IF EXISTS Performance;
25 DROP TABLE IF EXISTS Artist;
26 DROP TABLE IF EXISTS Artist_Genre;
27 DROP TABLE IF EXISTS Artist_SubGenre;
28 DROP TABLE IF EXISTS Band;
29 DROP TABLE IF EXISTS Band_Genre;
30 DROP TABLE IF EXISTS Band_SubGenre;
31 DROP TABLE IF EXISTS Band_Member;
32 DROP TABLE IF EXISTS Performance_Band;
33 DROP TABLE IF EXISTS Performance_Artist;
34 DROP TABLE IF EXISTS Attendee;
35 DROP TABLE IF EXISTS Ticket;
36 DROP TABLE IF EXISTS Review;
37 DROP TABLE IF EXISTS Resale_Offer;
38 DROP TABLE IF EXISTS Resale_Interest;
39 DROP TABLE IF EXISTS Resale_Interest_Type;
40 DROP TABLE IF EXISTS Resale_Match_Log;
41
42 -- Lookup Tables
43 CREATE TABLE Continent (
44     continent_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
45     name VARCHAR(50) NOT NULL UNIQUE,
46     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
47 );
48
49 INSERT INTO Continent (name) VALUES
50 ('Africa'),
51 ('Asia'),
```

```
52 ('Europe'),
53 ('North America'),
54 ('South America'),
55 ('Oceania');
56
57 CREATE TABLE Staff_Role (
58     role_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
59     name VARCHAR(50) NOT NULL UNIQUE,
60     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
61 );
62
63 INSERT INTO Staff_Role (name) VALUES
64 ('security'),
65 ('support'),
66 ('sound engineer'),
67 ('light technician'),
68 ('stagehand'),
69 ('medic'),
70 ('cleaning'),
71 ('backstage assistant');
72
73 CREATE TABLE Experience_Level (
74     level_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
75     name VARCHAR(50) NOT NULL UNIQUE,
76     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
77 );
78
79 INSERT INTO Experience_Level (name) VALUES
80 ('intern'),
81 ('beginner'),
82 ('intermediate'),
83 ('experienced'),
84 ('expert');
85
86 CREATE TABLE Performance_Type (
87     type_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
88     name VARCHAR(50) NOT NULL UNIQUE,
89     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
90 );
91
92 INSERT INTO Performance_Type (name) VALUES
93 ('warm up'),
94 ('headline'),
95 ('special guest'),
96 ('encore'),
97 ('other');
98
99 CREATE TABLE Ticket_Type (
100     type_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
101     name VARCHAR(50) NOT NULL UNIQUE,
102     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
103 );
104
105 INSERT INTO Ticket_Type (name) VALUES
```

```
106 ('general'),
107 ('VIP'),
108 ('backstage'),
109 ('early bird'),
110 ('student');
111
112 CREATE TABLE Payment_Method (
113     method_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
114     name VARCHAR(50) NOT NULL UNIQUE,
115     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
116 );
117
118 INSERT INTO Payment_Method (name) VALUES
119 ('credit card'),
120 ('debit card'),
121 ('bank transfer');
122
123 CREATE TABLE Ticket_Status (
124     status_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
125     name VARCHAR(20) NOT NULL UNIQUE,
126     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
127 );
128
129 INSERT INTO Ticket_Status (name) VALUES
130 ('active'),
131 ('used'),
132 ('on offer'),
133 ('unused');
134
135 CREATE TABLE Genre (
136     genre_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
137     name VARCHAR(100) NOT NULL UNIQUE
138 );
139
140 INSERT INTO Genre (name) VALUES
141 ('Rock'),
142 ('Pop'),
143 ('Jazz'),
144 ('Hip Hop'),
145 ('Electronic'),
146 ('Classical'),
147 ('Reggae'),
148 ('Latin'),
149 ('Metal'),
150 ('Funk');
151
152 CREATE TABLE SubGenre (
153     sub_genre_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
154     name VARCHAR(100) NOT NULL UNIQUE,
155     genre_id INT UNSIGNED NOT NULL,
156     FOREIGN KEY (genre_id) REFERENCES Genre(genre_id) ON DELETE CASCADE
157 );
158
159 INSERT INTO SubGenre (name, genre_id) VALUES
```

```

160 ('Hard Rock', (SELECT genre_id FROM Genre WHERE name='Rock')),
161 ('Progressive Rock', (SELECT genre_id FROM Genre WHERE name='Rock')),
162 ('Punk Rock', (SELECT genre_id FROM Genre WHERE name='Rock')),
163 ('Synthpop', (SELECT genre_id FROM Genre WHERE name='Pop')),
164 ('Electropop', (SELECT genre_id FROM Genre WHERE name='Pop')),
165 ('Dance Pop', (SELECT genre_id FROM Genre WHERE name='Pop')),
166 ('Bebop', (SELECT genre_id FROM Genre WHERE name='Jazz')),
167 ('Smooth Jazz', (SELECT genre_id FROM Genre WHERE name='Jazz')),
168 ('Free Jazz', (SELECT genre_id FROM Genre WHERE name='Jazz')),
169 ('Trap', (SELECT genre_id FROM Genre WHERE name='Hip Hop')),
170 ('Boom Bap', (SELECT genre_id FROM Genre WHERE name='Hip Hop')),
171 ('Lo-fi Hip Hop', (SELECT genre_id FROM Genre WHERE name='Hip Hop')),
172 ('Techno', (SELECT genre_id FROM Genre WHERE name='Electronic')),
173 ('House', (SELECT genre_id FROM Genre WHERE name='Electronic')),
174 ('Trance', (SELECT genre_id FROM Genre WHERE name='Electronic')),
175 ('Baroque', (SELECT genre_id FROM Genre WHERE name='Classical')),
176 ('Romantic', (SELECT genre_id FROM Genre WHERE name='Classical')),
177 ('Contemporary Classical', (SELECT genre_id FROM Genre WHERE name='Classical')),
178 ('Dub', (SELECT genre_id FROM Genre WHERE name='Reggae')),
179 ('Dancehall', (SELECT genre_id FROM Genre WHERE name='Reggae')),
180 ('Roots Reggae', (SELECT genre_id FROM Genre WHERE name='Reggae')),
181 ('Salsa', (SELECT genre_id FROM Genre WHERE name='Latin')),
182 ('Reggaeton', (SELECT genre_id FROM Genre WHERE name='Latin')),
183 ('Bachata', (SELECT genre_id FROM Genre WHERE name='Latin')),
184 ('Death Metal', (SELECT genre_id FROM Genre WHERE name='Metal')),
185 ('Black Metal', (SELECT genre_id FROM Genre WHERE name='Metal')),
186 ('Thrash Metal', (SELECT genre_id FROM Genre WHERE name='Metal')),
187 ('Afrofunk', (SELECT genre_id FROM Genre WHERE name='Funk')),
188 ('P-Funk', (SELECT genre_id FROM Genre WHERE name='Funk')),
189 ('Jazz-Funk', (SELECT genre_id FROM Genre WHERE name='Funk'));
190
191 -- Main Tables
192 CREATE TABLE Location (
193     loc_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
194     street_name VARCHAR(255) NOT NULL,
195     street_number VARCHAR(20) NOT NULL,
196     zip_code VARCHAR(10) NOT NULL,
197     city VARCHAR(100) NOT NULL,
198     country VARCHAR(100) NOT NULL,
199     continent_id INT UNSIGNED NOT NULL,
200     latitude DECIMAL(9,6) NOT NULL,
201     longitude DECIMAL(9,6) NOT NULL,
202     image VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
203     caption VARCHAR(100) NOT NULL,
204     FOREIGN KEY (continent_id) REFERENCES Continent(continent_id)
205         ON DELETE RESTRICT ON UPDATE CASCADE,
206     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
207 );
208
209 CREATE TABLE Festival (
210     fest_year INT UNSIGNED PRIMARY KEY,
211     name VARCHAR(255) NOT NULL,
212     start_date DATE NOT NULL,
213     end_date DATE NOT NULL,

```

```

214     image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
215     caption VARCHAR(100) NOT NULL,
216     loc_id  INT UNSIGNED NOT NULL,
217     FOREIGN KEY (loc_id) REFERENCES Location(loc_id)
218         ON DELETE RESTRICT ON UPDATE CASCADE,
219     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
220 );
221
222 CREATE TABLE Stage (
223     stage_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
224     name     VARCHAR(255) NOT NULL,
225     capacity INT NOT NULL CHECK (capacity > 0),
226     image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
227     caption VARCHAR(100) NOT NULL,
228     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
229 );
230
231 CREATE TABLE Equipment (
232     equip_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
233     name     VARCHAR(255) NOT NULL,
234     image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
235     caption VARCHAR(100) NOT NULL,
236     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
237 );
238
239 CREATE TABLE Stage_Equipment (
240     stage_id INT UNSIGNED,
241     equip_id INT UNSIGNED,
242     PRIMARY KEY(stage_id, equip_id),
243     FOREIGN KEY(stage_id) REFERENCES Stage(stage_id)
244         ON DELETE CASCADE ON UPDATE CASCADE,
245     FOREIGN KEY(equip_id) REFERENCES Equipment(equip_id)
246         ON DELETE CASCADE ON UPDATE CASCADE,
247     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
248 );
249
250 CREATE TABLE Event (
251     event_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
252     title    VARCHAR(255) NOT NULL,
253     is_full  BOOLEAN NOT NULL DEFAULT FALSE,
254     start_dt DATETIME NOT NULL,
255     end_dt   DATETIME NOT NULL,
256     image   VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
257     caption VARCHAR(100) NOT NULL,
258     fest_year INT UNSIGNED NOT NULL,
259     stage_id INT UNSIGNED NOT NULL,
260     generated_date DATE NOT NULL,                -- updated via trigger
261     UNIQUE KEY uq_event_stage (event_id, stage_id),
262     UNIQUE KEY uq_start_date (generated_date),
263     FOREIGN KEY (fest_year) REFERENCES Festival(fest_year)
264         ON DELETE RESTRICT ON UPDATE CASCADE,
265     FOREIGN KEY (stage_id) REFERENCES Stage(stage_id)
266         ON DELETE RESTRICT ON UPDATE CASCADE,
267     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

```

```

268 );
269
270 CREATE TABLE Staff (
271     staff_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
272     first_name VARCHAR(100) NOT NULL,
273     last_name VARCHAR(100) NOT NULL,
274     date_of_birth DATE NOT NULL,
275     role_id INT UNSIGNED NOT NULL,
276     experience_id INT UNSIGNED NOT NULL,
277     image VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
278     caption VARCHAR(100) NOT NULL,
279     FOREIGN KEY (role_id) REFERENCES Staff_Role(role_id)
280         ON DELETE RESTRICT ON UPDATE CASCADE,
281     FOREIGN KEY (experience_id) REFERENCES Experience_Level(level_id)
282         ON DELETE RESTRICT ON UPDATE CASCADE,
283     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
284 );
285
286 CREATE TABLE Works_On (
287     staff_id INT UNSIGNED,
288     event_id INT UNSIGNED,
289     PRIMARY KEY(staff_id, event_id),
290     FOREIGN KEY(staff_id) REFERENCES Staff(staff_id)
291         ON DELETE CASCADE ON UPDATE CASCADE,
292     FOREIGN KEY(event_id) REFERENCES Event(event_id)
293         ON DELETE RESTRICT ON UPDATE CASCADE,
294     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
295 );
296
297 CREATE TABLE Performance (
298     perf_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
299     type_id INT UNSIGNED NOT NULL,
300     datetime DATETIME NOT NULL,
301     duration TINYINT UNSIGNED NOT NULL CHECK (duration BETWEEN 1 AND 180),
302     break_duration TINYINT CHECK (break_duration BETWEEN 5 AND 30),
303     stage_id INT UNSIGNED NOT NULL,
304     event_id INT UNSIGNED NOT NULL,
305     sequence_number TINYINT UNSIGNED NOT NULL CHECK (sequence_number > 0),
306     UNIQUE KEY uq_event_seq (event_id, sequence_number),
307     CONSTRAINT fk_perf_event_stage
308         FOREIGN KEY (event_id, stage_id)
309             REFERENCES Event(event_id, stage_id)
310             ON DELETE RESTRICT ON UPDATE CASCADE,
311     FOREIGN KEY(stage_id) REFERENCES Stage(stage_id)
312         ON DELETE RESTRICT ON UPDATE CASCADE,
313     FOREIGN KEY(type_id) REFERENCES Performance_Type(type_id)
314         ON DELETE RESTRICT ON UPDATE CASCADE,
315     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
316 );
317
318 CREATE TABLE Artist (
319     artist_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
320     first_name VARCHAR(100) NOT NULL,
321     last_name VARCHAR(100) NOT NULL,

```

```
322     nickname    VARCHAR(100),
323     date_of_birth DATE NOT NULL,
324     webpage     VARCHAR(100) CHECK (webpage LIKE 'https://%'),
325     instagram   VARCHAR(100) CHECK (instagram LIKE '@%'),
326     image       VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
327     caption     VARCHAR(100) NOT NULL,
328     updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
329 );
330
331 CREATE TABLE Artist_Genre (
332     artist_id INT UNSIGNED,
333     genre_id  INT UNSIGNED,
334     PRIMARY KEY (artist_id, genre_id),
335     FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE CASCADE,
336     FOREIGN KEY (genre_id)  REFERENCES Genre(genre_id)  ON DELETE CASCADE
337 );
338
339 CREATE TABLE Artist_SubGenre (
340     artist_id INT UNSIGNED,
341     sub_genre_id INT UNSIGNED,
342     PRIMARY KEY (artist_id, sub_genre_id),
343     FOREIGN KEY (artist_id) REFERENCES Artist(artist_id) ON DELETE CASCADE,
344     FOREIGN KEY (sub_genre_id) REFERENCES SubGenre(sub_genre_id) ON DELETE CASCADE
345 );
346
347 CREATE TABLE Band (
348     band_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
349     name    VARCHAR(255) NOT NULL,
350     formation_date DATE,
351     webpage   VARCHAR(100) CHECK (webpage LIKE 'https://%'),
352     instagram VARCHAR(100) CHECK (instagram LIKE '@%'),
353     image     VARCHAR(100) NOT NULL CHECK (image LIKE 'https://%'),
354     caption   VARCHAR(100) NOT NULL,
355     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
356 );
357
358 CREATE TABLE Band_Genre (
359     band_id INT UNSIGNED,
360     genre_id INT UNSIGNED,
361     PRIMARY KEY (band_id, genre_id),
362     FOREIGN KEY (band_id) REFERENCES Band(band_id) ON DELETE CASCADE,
363     FOREIGN KEY (genre_id) REFERENCES Genre(genre_id) ON DELETE CASCADE
364 );
365
366 CREATE TABLE Band_SubGenre (
367     band_id INT UNSIGNED,
368     sub_genre_id INT UNSIGNED,
369     PRIMARY KEY (band_id, sub_genre_id),
370     FOREIGN KEY (band_id) REFERENCES Band(band_id) ON DELETE CASCADE,
371     FOREIGN KEY (sub_genre_id) REFERENCES SubGenre(sub_genre_id) ON DELETE CASCADE
372 );
373
374 CREATE TABLE Band_Member (
375     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
```

```
484 );
485
486 CREATE TABLE Resale_Interest_Type (
487     request_id INT UNSIGNED NOT NULL,
488     type_id INT UNSIGNED NOT NULL,
489     PRIMARY KEY(request_id, type_id),
490     FOREIGN KEY(request_id) REFERENCES Resale_Interest(request_id)
491         ON DELETE CASCADE ON UPDATE CASCADE,
492     FOREIGN KEY(type_id) REFERENCES Ticket_Type(type_id)
493         ON DELETE RESTRICT ON UPDATE CASCADE,
494     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
495 );
496
497 CREATE TABLE Resale_Match_Log (
498     match_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
499     match_type ENUM('offer', 'interest') NOT NULL,
500     ticket_id INT UNSIGNED NOT NULL,
501     offered_type_id INT UNSIGNED NOT NULL,
502     requested_type_id INT UNSIGNED NOT NULL,
503     buyer_id INT UNSIGNED NOT NULL,
504     seller_id INT UNSIGNED NOT NULL,
505     match_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
506     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
507     FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id)
508         ON DELETE RESTRICT ON UPDATE CASCADE,
509     FOREIGN KEY (offered_type_id) REFERENCES Ticket_Type(type_id)
510         ON DELETE RESTRICT ON UPDATE CASCADE,
511     FOREIGN KEY (requested_type_id) REFERENCES Ticket_Type(type_id)
512         ON DELETE RESTRICT ON UPDATE CASCADE,
513     FOREIGN KEY (buyer_id) REFERENCES Attendee(attendee_id)
514         ON DELETE RESTRICT ON UPDATE CASCADE,
515     FOREIGN KEY (seller_id) REFERENCES Attendee(attendee_id)
516         ON DELETE RESTRICT ON UPDATE CASCADE
517 );
518
```

## sql\indexing.sql

```

1  -- -----
2  -- - Indexing ---
3  -- -----
4
5  USE pulse_university;
6
7  -- Helper: drop procedure if it exists
8  -- MYSQL does not support DROP INDEX IF EXISTS!
9  DROP PROCEDURE IF EXISTS DropIndexIfExists;
10
11 CREATE PROCEDURE DropIndexIfExists(tbl VARCHAR(64), idx VARCHAR(64))
12     SQL SECURITY INVOKER
13     COMMENT 'Drops index only if it exists'
14 BEGIN
15     DECLARE count INT;
16     SELECT COUNT(*) INTO count
17     FROM information_schema.statistics
18     WHERE table_schema = DATABASE()
19         AND table_name = tbl
20         AND index_name = idx;
21
22     IF count > 0 THEN
23         SET @stmt = CONCAT('DROP INDEX `', idx, '` ON `', tbl, '`');
24         PREPARE s FROM @stmt;
25         EXECUTE s;
26         DEALLOCATE PREPARE s;
27     END IF;
28 END;
29
30 /* -----
31  * 1. Ticket-centric queries    (Q 1, 8, 9)
32  * -----*/
33 CALL DropIndexIfExists('Ticket', 'idx_ticket_event_date_payment');
34 CALL DropIndexIfExists('Ticket', 'idx_ticket_attendee_event');
35 CALL DropIndexIfExists('Ticket', 'idx_ticket_attendee_year_event');
36
37 CREATE INDEX idx_ticket_event_date_payment ON Ticket (event_id, purchase_date,
method_id);
38 CREATE INDEX idx_ticket_attendee_event ON Ticket (attendee_id, event_id);
39 CREATE INDEX idx_ticket_attendee_year_event ON Ticket (attendee_id, purchase_date,
event_id);
40
41 /* -----
42  * 2. Event / festival helpers  (Q 1, 2, 3, 7, 8, 10, 13, 14)
43  * -----*/
44 CALL DropIndexIfExists('Event', 'idx_event_year');
45 CALL DropIndexIfExists('Event', 'idx_event_start');
46
47 CREATE INDEX idx_event_year ON Event (fest_year);           -- yearly roll-ups
48 CREATE INDEX idx_event_start ON Event(start_dt);
49
50 /* -----

```

```

51 * 3. Genres & sub-genres (Q 2, 10, 14)
52 * -----*/
53 CALL DropIndexIfExists('Artist_Genre', 'idx_artist_genre');
54 CALL DropIndexIfExists('Artist_Genre', 'idx_artist_genre_by_artist');
55
56 CREATE INDEX idx_artist_genre ON Artist_Genre (genre_id, artist_id); --
filter-first
57 CREATE INDEX idx_artist_genre_by_artist ON Artist_Genre (artist_id, genre_id); -- self-
join path
58
59 /* -----
60 * 4. Artist attributes (Q 5)
61 * -----*/
62 CALL DropIndexIfExists('Artist', 'idx_artist_dob');
63 CREATE INDEX idx_artist_dob ON Artist (date_of_birth); -- "young artists" filter
64
65 /* -----
66 * 5. Performance look-ups (Q 2, 3, 5, 10, 11, 13, 14, 15)
67 * -----*/
68 CALL DropIndexIfExists('Performance', 'idx_perf_event_type');
69 CALL DropIndexIfExists('Performance', 'idx_perf_type');
70 CALL DropIndexIfExists('Performance', 'idx_perf_datetime');
71 CALL DropIndexIfExists('Performance_Artist', 'idx_perf_artist');
72
73 CREATE INDEX idx_perf_event_type ON Performance (event_id, type_id);
74 CREATE INDEX idx_perf_type ON Performance (type_id); -- quick "warm-up"
filter
75 CREATE INDEX idx_perf_datetime ON Performance (datetime);
76 CREATE INDEX idx_perf_artist ON Performance_Artist (artist_id, perf_id);
77
78 /* -----
79 * 6. Staff & staffing ratios (Q 7, 8)
80 * -----*/
81 CALL DropIndexIfExists('Works_On', 'idx_workson_event_staff');
82 CALL DropIndexIfExists('Staff', 'idx_staff_role_staff');
83 CALL DropIndexIfExists('Staff', 'idx_staff_experience');
84
85 CREATE INDEX idx_workson_event_staff ON Works_On (event_id, staff_id); -- event-centric
scans
86 CREATE INDEX idx_staff_role_staff ON Staff (role_id, staff_id); -- role filter +
join
87 CREATE INDEX idx_staff_experience ON Staff (experience_id);
88
89 /* -----
90 * 7. Reviews (Q 6, 11, 15)
91 * -----*/
92 CALL DropIndexIfExists('Review', 'idx_review_perf_io');
93 CALL DropIndexIfExists('Review', 'idx_review_attendee_overall');
94 CALL DropIndexIfExists('Review', 'idx_review_perf_attendee_overall');
95
96 CREATE INDEX idx_review_perf_io ON Review (perf_id, interpretation,
overall); -- covering index
97 CREATE INDEX idx_review_attendee_overall ON Review (attendee_id, overall);
98 CREATE INDEX idx_review_perf_attendee_overall ON Review (perf_id, attendee_id, overall);

```

```
99
100 /* -----
101  * 8. Geography (Q 13)
102  * -----*/
103 CALL DropIndexIfExists('Location', 'idx_location_continent');
104 CREATE INDEX idx_location_continent ON Location (continent_id);
105
106 -- Clean up: drop helper procedure
107 DROP PROCEDURE IF EXISTS DropIndexIfExists;
108
```

## sql\triggers.sql

```

1  -- -----
2  -- - Triggers ---
3  -- -----
4
5  USE pulse_university;
6
7  -- Drop all triggers
8  DROP TRIGGER IF EXISTS trg_band_validate_before_ins;
9  DROP TRIGGER IF EXISTS trg_band_sync_members_after_ins;
10 DROP TRIGGER IF EXISTS trg_artist_validate_before_ins;
11 DROP TRIGGER IF EXISTS trg_auto_assign_band_after_artist_ins;
12 DROP TRIGGER IF EXISTS trg_no_double_stage_artist;
13 DROP TRIGGER IF EXISTS trg_no_double_stage_band;
14 DROP TRIGGER IF EXISTS trg_no_stage_overlap;
15 DROP TRIGGER IF EXISTS trg_max_consecutive_years_artist;
16 DROP TRIGGER IF EXISTS trg_max_consecutive_years_band;
17 DROP TRIGGER IF EXISTS trg_event_within_festival_dates;
18 DROP TRIGGER IF EXISTS trg_performance_inside_event;
19 DROP TRIGGER IF EXISTS trg_safe_festival_date_update;
20 DROP TRIGGER IF EXISTS trg_safe_event_date_update;
21 DROP TRIGGER IF EXISTS trg_delete_band_if_no_members;
22 DROP TRIGGER IF EXISTS trg_staff_ratio_after_delete;
23 DROP TRIGGER IF EXISTS trg_staff_ratio_after_update;
24 DROP TRIGGER IF EXISTS trg_ticket_capacity_check;
25 DROP TRIGGER IF EXISTS trg_check_vip_ticket_limit;
26 DROP TRIGGER IF EXISTS trg_validate_ticket_ean;
27 DROP TRIGGER IF EXISTS trg_validate_ticket_ean_upd;
28 DROP TRIGGER IF EXISTS trg_resale_offer_only_active;
29 DROP TRIGGER IF EXISTS trg_review_only_with_used_ticket;
30 DROP TRIGGER IF EXISTS trg_resale_offer_timestamp;
31 DROP TRIGGER IF EXISTS trg_resale_interest_timestamp;
32 DROP TRIGGER IF EXISTS trg_stage_capacity_update;
33 DROP TRIGGER IF EXISTS trg_block_purchase_date_update;
34 DROP TRIGGER IF EXISTS trg_validate_ticket_purchase_date;
35 DROP TRIGGER IF EXISTS trg_artist_subgenre_consistency;
36 DROP TRIGGER IF EXISTS trg_band_subgenre_consistency;
37 DROP TRIGGER IF EXISTS trg_delete_attendee_cleanup;
38 DROP TRIGGER IF EXISTS trg_match_resale_interest;
39 DROP TRIGGER IF EXISTS trg_match_resale_offer;
40 DROP TRIGGER IF EXISTS trg_set_event_date;
41
42 -- =====
43 -- 1. Performance-assignment rules (solo / band)
44 -- =====
45
46 -- 1. BEFORE INSERT on Performance_Band - validate band insertion
47 CREATE TRIGGER trg_band_validate_before_ins
48 BEFORE INSERT ON Performance_Band
49 FOR EACH ROW
50 BEGIN
51     -- 1.a allow **only one** band per performance

```

```
52 IF (SELECT COUNT(*) FROM Performance_Band WHERE perf_id = NEW.perf_id) > 0 THEN
53     SIGNAL SQLSTATE '45000'
54     SET MESSAGE_TEXT = 'A band is already assigned to this performance.';
55 END IF;
56
57 -- 1.b if artists already exist, they **must** belong to that band
58 IF EXISTS (
59     SELECT 1
60     FROM Performance_Artist pa
61     WHERE pa.perf_id = NEW.perf_id
62           AND pa.artist_id NOT IN
63             (SELECT artist_id FROM Band_Member WHERE band_id = NEW.band_id)
64 ) THEN
65     SIGNAL SQLSTATE '45000'
66     SET MESSAGE_TEXT = 'Existing artist is not a member of this band.';
67 END IF;
68 END;
69
70 -- 2. AFTER INSERT on Performance_Band - auto-insert every band member into
Performance_Artist
71 CREATE TRIGGER trg_band_sync_members_after_ins
72 AFTER INSERT ON Performance_Band
73 FOR EACH ROW
74 BEGIN
75     -- INSERT IGNORE avoids duplicates thanks to (perf_id,artist_id) PK
76     INSERT IGNORE INTO Performance_Artist (perf_id, artist_id)
77     SELECT NEW.perf_id, bm.artist_id
78     FROM Band_Member bm
79     WHERE bm.band_id = NEW.band_id;
80 END;
81
82 -- 3. BEFORE INSERT on Performance_Artist - validate artist insertion
83 CREATE TRIGGER trg_artist_validate_before_ins
84 BEFORE INSERT ON Performance_Artist
85 FOR EACH ROW
86 BEGIN
87     -- 3.a If a band is already set, artist must be a member
88     DECLARE v_band INT;
89     SELECT band_id INTO v_band
90     FROM Performance_Band
91     WHERE perf_id = NEW.perf_id
92     LIMIT 1;
93
94     IF v_band IS NOT NULL
95         AND NOT EXISTS (
96             SELECT 1
97             FROM Band_Member
98             WHERE band_id = v_band
99                   AND artist_id = NEW.artist_id)
100     THEN
101         SIGNAL SQLSTATE '45000'
102         SET MESSAGE_TEXT = 'Artist is not a member of the assigned band.';
103     END IF;
104
```



```

105      -- 3.b If NO band yet, but other artists exist ⇒ they must all share at least one
common band with the newcomer
106      IF v_band IS NULL
107          AND (SELECT COUNT(*) FROM Performance_Artist WHERE perf_id = NEW.perf_id) > 0
108          AND NOT EXISTS (
109              SELECT 1
110              FROM    Band_Member bm_new                -- bands of the new artist
111              WHERE   bm_new.artist_id = NEW.artist_id
112                  -- every existing artist must also be member of bm_new.band_id
113              AND NOT EXISTS (
114                  SELECT 1
115                  FROM    Performance_Artist pa
116                  WHERE   pa.perf_id = NEW.perf_id        -- existing artists
117                      AND NOT EXISTS (
118                          SELECT 1
119                          FROM    Band_Member bm_old
120                          WHERE   bm_old.band_id = bm_new.band_id
121                              AND bm_old.artist_id = pa.artist_id))
122              )
123      THEN
124          SIGNAL SQLSTATE '45000'
125          SET MESSAGE_TEXT = 'Artists do not share a common band.';
126      END IF;
127  END;
128
129  -- 4. AFTER INSERT on Performance_Artist - when every member of some band is now present
130  -- insert that band into Performance_Band (rule 5)
131  CREATE TRIGGER trg_auto_assign_band_after_artist_ins
132  AFTER INSERT ON Performance_Artist
133  FOR EACH ROW
134  BEGIN
135      -- all DECLAREs first
136      DECLARE target_band INT;
137
138      -- rest of the logic
139      IF (SELECT COUNT(*) FROM Performance_Band WHERE perf_id = NEW.perf_id) = 0 THEN
140          /* find a band whose every member is now present */
141          SELECT bm.band_id INTO target_band
142          FROM    Band_Member bm
143          WHERE   bm.artist_id = NEW.artist_id
144          GROUP BY bm.band_id
145          HAVING COUNT(*) = (
146              SELECT COUNT(*)                -- #members of that band
147              FROM Band_Member
148              WHERE band_id = bm.band_id)
149          AND COUNT(*) = (
150              SELECT COUNT(DISTINCT artist_id)
151              FROM    Performance_Artist
152              WHERE   perf_id = NEW.perf_id)
153          LIMIT 1;
154
155      IF target_band IS NOT NULL THEN
156          INSERT IGNORE INTO Performance_Band (perf_id, band_id)
157          VALUES (NEW.perf_id, target_band);

```

```

158         END IF;
159     END IF;
160 END;
161
162 -- =====
163 -- 2. Scheduling & participation (time / dates)
164 -- =====
165
166 -- 5. Same artist cannot play two stages at the same time
167 CREATE TRIGGER trg_no_double_stage_artist
168 BEFORE INSERT ON Performance_Artist
169 FOR EACH ROW
170 BEGIN
171     DECLARE d DATETIME; DECLARE s INT;
172     SELECT datetime, stage_id INTO d, s FROM Performance WHERE perf_id = NEW.perf_id;
173     IF EXISTS (
174         SELECT 1
175         FROM Performance p
176             JOIN Performance_Artist pa ON p.perf_id = pa.perf_id
177         WHERE pa.artist_id = NEW.artist_id
178             AND p.datetime = d
179             AND p.stage_id <> s )
180     THEN
181         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Artist already on another stage at
that time.';
182     END IF;
183 END;
184
185 -- 6. Same band cannot play two stages at the same time
186 CREATE TRIGGER trg_no_double_stage_band
187 BEFORE INSERT ON Performance_Band
188 FOR EACH ROW
189 BEGIN
190     DECLARE d DATETIME; DECLARE s INT;
191     SELECT datetime, stage_id INTO d, s FROM Performance WHERE perf_id = NEW.perf_id;
192     IF EXISTS (
193         SELECT 1
194         FROM Performance p
195             JOIN Performance_Band pb ON p.perf_id = pb.perf_id
196         WHERE pb.band_id = NEW.band_id
197             AND p.datetime = d
198             AND p.stage_id <> s )
199     THEN
200         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Band already on another stage at that
time.';
201     END IF;
202 END;
203
204 -- 7. One performance per stage at any moment on insert
205 CREATE TRIGGER trg_no_stage_overlap
206 BEFORE INSERT ON Performance
207 FOR EACH ROW
208 BEGIN
209     IF EXISTS (

```

```

210     SELECT 1
211     FROM    Performance
212     WHERE   stage_id = NEW.stage_id
213           AND perf_id <> NEW.perf_id
214           AND NEW.datetime < ADDDATE(datetime, INTERVAL duration MINUTE)
215           AND ADDDATE(NEW.datetime, INTERVAL NEW.duration MINUTE) > datetime
216   ) THEN
217       SIGNAL SQLSTATE '45000'
218       SET MESSAGE_TEXT = 'Stage already booked for that time-slot.';
219   END IF;
220 END;
221
222 -- 8. Check if an artist has performed for more than 3 consecutive years
223 CREATE TRIGGER trg_max_consecutive_years_artist
224 BEFORE INSERT ON Performance_Artist
225 FOR EACH ROW
226 BEGIN
227     DECLARE perf_year INT;
228
229     -- Year of the performance we're inserting
230     SELECT f.fest_year
231     INTO perf_year
232     FROM Performance p
233     JOIN Event      e ON p.event_id = e.event_id
234     JOIN Festival   f ON e.fest_year = f.fest_year
235     WHERE p.perf_id = NEW.perf_id;
236
237     -- Past years + the pending year for this artist
238     IF EXISTS (
239         WITH years AS (
240             SELECT DISTINCT f.fest_year AS y
241             FROM Performance      p
242             JOIN Event            e ON e.event_id   = p.event_id
243             JOIN Festival         f ON f.fest_year  = e.fest_year
244             WHERE EXISTS (
245                 SELECT 1 FROM Performance_Artist pa
246                 WHERE pa.perf_id = p.perf_id
247                 AND pa.artist_id = NEW.artist_id
248             )
249             UNION ALL
250             SELECT perf_year
251         ),
252         seq AS (
253             SELECT y,
254                    ROW_NUMBER() OVER (ORDER BY y) AS rn
255             FROM years
256         ),
257         runs AS (
258             SELECT COUNT(*) AS run_len
259             FROM seq
260             GROUP BY y - rn      -- islands-and-gaps trick
261         )
262         SELECT 1 FROM runs WHERE run_len >= 4
263     ) 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
271 -- years => don't insert the band
272 CREATE TRIGGER trg_max_consecutive_years_band
273 BEFORE INSERT ON Performance_Band
274 FOR EACH ROW
275 BEGIN
276     DECLARE perf_year INT;
277
278     -- Year of the performance we are inserting
279     SELECT f.fest_year
280     INTO perf_year
281     FROM Performance p
282     JOIN Event e ON p.event_id = e.event_id
283     JOIN Festival f ON e.fest_year = f.fest_year
284     WHERE p.perf_id = NEW.perf_id;
285
286     -- Build a set of {artist_id, year} rows:
287     -- every past year each band member has performed
288     -- plus the year we are about to add for each artist look for a run of 4
289     consecutive years.
290     IF EXISTS (
291         WITH member_years AS (
292             SELECT bm.artist_id,
293                    f.fest_year AS y
294             FROM Band_Member bm
295             JOIN Performance_Artist pa ON pa.artist_id = bm.artist_id
296             JOIN Performance p ON p.perf_id = pa.perf_id
297             JOIN Event e ON e.event_id = p.event_id
298             JOIN Festival f ON f.fest_year = e.fest_year
299             WHERE bm.band_id = NEW.band_id
300
301             UNION ALL
302
303             SELECT bm.artist_id, perf_year
304             FROM Band_Member bm
305             WHERE bm.band_id = NEW.band_id
306         ),
307         seq AS (
308             SELECT artist_id,
309                    y,
310                    ROW_NUMBER() OVER (PARTITION BY artist_id ORDER BY y) AS rn
311             FROM (SELECT DISTINCT artist_id, y FROM member_years) x
312         ),
313         runs AS (
314             SELECT artist_id,
315                    COUNT(*) AS run_len
316             FROM seq
317             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
353 window.';
354     END IF;
355 END;
356
357 -- 12. Allow change of dates of festival, only if events are still in bounds
358 CREATE TRIGGER trg_safe_festival_date_update
359 BEFORE UPDATE ON Festival
360 FOR EACH ROW
361 BEGIN
362     -- Only run check if dates are changing
363     IF NEW.start_date <> OLD.start_date OR NEW.end_date <> OLD.end_date THEN
364         -- Block if any event for this festival falls outside new bounds
365         IF EXISTS (
366             SELECT 1
367             FROM Event
368             WHERE fest_year = NEW.fest_year
369                 AND (DATE(start_dt) < NEW.start_date OR DATE(start_dt) > NEW.end_date)

```

```

369         ) THEN
370             SIGNAL SQLSTATE '45000'
371             SET MESSAGE_TEXT = 'Cannot change festival dates: some events would be out of
bounds.';
372         END IF;
373     END IF;
374 END;
375
376 -- 13. Allow change of dates of event, only if performances are still in bounds
377 CREATE TRIGGER trg_safe_event_date_update
378 BEFORE UPDATE ON Event
379 FOR EACH ROW
380 BEGIN
381     -- Only proceed if event window changes
382     IF NEW.start_dt <> OLD.start_dt OR NEW.end_dt <> OLD.end_dt THEN
383         -- Block update if any performance starts before or ends after the new window
384         IF EXISTS (
385             SELECT 1
386             FROM Performance
387             WHERE event_id = NEW.event_id
388                 AND (
389                     datetime < NEW.start_dt
390                     OR ADDTIME(datetime, SEC_TO_TIME(duration * 60)) > NEW.end_dt
391                 )
392         ) THEN
393             SIGNAL SQLSTATE '45000'
394             SET MESSAGE_TEXT = 'Cannot update event dates: some performances would fall
outside the new window.';
395         END IF;
396     END IF;
397 END;
398
399 -- =====
400 -- 3. Band membership clean-up
401 -- =====
402
403 -- 14. Delete band when it has no members
404 CREATE TRIGGER trg_delete_band_if_no_members
405 AFTER DELETE ON Band_Member
406 FOR EACH ROW
407 BEGIN
408     IF (SELECT COUNT(*) FROM Band_Member WHERE band_id = OLD.band_id) = 0 THEN
409         DELETE FROM Band WHERE band_id = OLD.band_id;
410     END IF;
411 END;
412
413 -- =====
414 -- 4. Staffing ratio (≥5 % security, ≥2 % support)
415 -- =====
416
417 -- 15. Check ratios after DELETE on Works_On
418 CREATE TRIGGER trg_staff_ratio_after_delete
419 AFTER DELETE ON Works_On
420 FOR EACH ROW

```

```
421 BEGIN
422     CALL check_staff_ratio(OLD.event_id);
423 END;
424
425 -- 16. check ratios after UPDATE on Works_On
426 CREATE TRIGGER trg_staff_ratio_after_update
427 AFTER UPDATE ON Works_On
428 FOR EACH ROW
429 BEGIN
430     IF OLD.event_id <> NEW.event_id THEN
431         CALL check_staff_ratio(OLD.event_id);
432         CALL check_staff_ratio(NEW.event_id);
433     ELSE
434         CALL check_staff_ratio(NEW.event_id);
435     END IF;
436 END;
437
438 -- =====
439 -- 5. Ticket capacity / selling
440 -- =====
441
442 -- 17. Block overselling and mark event full (all statuses)
443 CREATE TRIGGER trg_ticket_capacity_check
444 BEFORE INSERT ON Ticket
445 FOR EACH ROW
446 BEGIN
447     DECLARE cap INT;
448     DECLARE sold INT;
449
450     -- Capacity of event's stage
451     SELECT s.capacity INTO cap
452     FROM   Event e
453     JOIN   Stage s ON e.stage_id = s.stage_id
454     WHERE  e.event_id = NEW.event_id;
455
456     -- Count all tickets for this event, regardless of status
457     SELECT COUNT(*) INTO sold
458     FROM   Ticket
459     WHERE  event_id = NEW.event_id;
460
461     -- Block if full
462     IF sold >= cap THEN
463         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Capacity reached for this event.';
464     END IF;
465
466     -- If this insert fills the event exactly, mark it as full
467     IF sold + 1 = cap THEN
468         UPDATE Event SET is_full = TRUE WHERE event_id = NEW.event_id;
469     END IF;
470 END;
471
472 -- 18. VIP tickets ≤10 % of capacity
473 CREATE TRIGGER trg_check_vip_ticket_limit
474 BEFORE INSERT ON Ticket
```



```
475 FOR EACH ROW
476 BEGIN
477     DECLARE cap INT; DECLARE vip INT; DECLARE vipType INT;
478
479     SELECT type_id INTO vipType FROM Ticket_Type WHERE name='VIP' LIMIT 1;
480
481     SELECT s.capacity INTO cap
482     FROM   Event e
483           JOIN Stage s ON e.stage_id = s.stage_id
484     WHERE  e.event_id = NEW.event_id;
485
486     SELECT COUNT(*) INTO vip
487     FROM   Ticket
488     WHERE  event_id = NEW.event_id
489           AND type_id = vipType;
490
491     IF NEW.type_id = vipType AND vip >= CEIL(cap * 0.10) THEN
492         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'VIP ticket cap exceeded.';
493     END IF;
494 END;
495
496 -- 19. Confirm EAN format on insert
497 CREATE TRIGGER trg_validate_ticket_ean
498 BEFORE INSERT ON Ticket
499 FOR EACH ROW
500 BEGIN
501     DECLARE ean_str CHAR(13);
502     DECLARE sum INT DEFAULT 0;
503     DECLARE i INT DEFAULT 1;
504     DECLARE digit INT;
505     DECLARE checksum INT;
506
507     -- Convert to 13-digit string
508     SET ean_str = LPAD(NEW.ean_number, 13, '0');
509
510     -- Must be 13 numeric characters
511     IF ean_str NOT REGEXP '^[0-9]{13}$' THEN
512         SIGNAL SQLSTATE '45000'
513         SET MESSAGE_TEXT = 'EAN must be a 13-digit number.';
514     END IF;
515
516     -- Calculate EAN-13 checksum: use digits 1-12
517     WHILE i <= 12 DO
518         SET digit = CAST(SUBSTRING(ean_str, i, 1) AS UNSIGNED);
519         SET sum = sum + digit * IF(MOD(i, 2) = 0, 3, 1);
520         SET i = i + 1;
521     END WHILE;
522
523     SET checksum = (10 - (sum MOD 10)) MOD 10;
524
525     -- Validate against 13th digit
526     IF checksum <> CAST(SUBSTRING(ean_str, 13, 1) AS UNSIGNED) THEN
527         SIGNAL SQLSTATE '45000'
528         SET MESSAGE_TEXT = 'Invalid EAN-13 checksum.';
```

```

529     END IF;
530 END;
531
532 -- 20. Confirm EAN format on update
533 CREATE TRIGGER trg_validate_ticket_ean_upd
534 BEFORE UPDATE ON Ticket
535 FOR EACH ROW
536 BEGIN
537     DECLARE ean_str CHAR(13);
538     DECLARE sum INT DEFAULT 0;
539     DECLARE i INT DEFAULT 1;
540     DECLARE digit INT;
541     DECLARE checksum INT;
542
543     -- Convert to 13-digit string
544     SET ean_str = LPAD(NEW.ean_number, 13, '0');
545
546     -- Must be 13 numeric characters
547     IF ean_str NOT REGEXP '^([0-9]){13}$' THEN
548         SIGNAL SQLSTATE '45000'
549         SET MESSAGE_TEXT = 'EAN must be a 13-digit number.';
550     END IF;
551
552     -- Calculate EAN-13 checksum: use digits 1-12
553     WHILE i <= 12 DO
554         SET digit = CAST(SUBSTRING(ean_str, i, 1) AS UNSIGNED);
555         SET sum = sum + digit * IF(MOD(i, 2) = 0, 3, 1);
556         SET i = i + 1;
557     END WHILE;
558
559     SET checksum = (10 - (sum MOD 10)) MOD 10;
560
561     -- Validate against 13th digit
562     IF checksum <> CAST(SUBSTRING(ean_str, 13, 1) AS UNSIGNED) THEN
563         SIGNAL SQLSTATE '45000'
564         SET MESSAGE_TEXT = 'Invalid EAN-13 checksum.';
565     END IF;
566 END;
567
568 -- =====
569 -- 6. Resale & ticket status
570 -- =====
571
572 -- 21. Only ACTIVE tickets can be offered for resale
573 CREATE TRIGGER trg_resale_offer_only_active
574 BEFORE INSERT ON Resale_Offer
575 FOR EACH ROW
576 BEGIN
577     IF (SELECT status_id FROM Ticket WHERE ticket_id = NEW.ticket_id) <>
578         (SELECT status_id FROM Ticket_Status WHERE name='active' LIMIT 1) THEN
579         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Ticket not eligible for resale.';
580     END IF;
581 END;
582

```

```

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
```

```

690 BEFORE INSERT ON Band_SubGenre
691 FOR EACH ROW
692 BEGIN
693     IF NOT EXISTS (
694         SELECT 1
695         FROM SubGenre sg
696             JOIN Band_Genre bg ON bg.genre_id = sg.genre_id
697         WHERE sg.sub_genre_id = NEW.sub_genre_id
698             AND bg.band_id = NEW.band_id
699     ) THEN
700         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Band sub-genre inconsistent.';
701     END IF;
702 END;
703
704 -- =====
705 -- 8. Cascade clean-ups
706 -- =====
707
708 -- 30. Delete attendee => purge tickets / resale / reviews
709 CREATE TRIGGER trg_delete_attendee_cleanup
710 AFTER DELETE ON Attendee
711 FOR EACH ROW
712 BEGIN
713     DELETE FROM Ticket WHERE attendee_id = OLD.attendee_id;
714     DELETE FROM Resale_Offer WHERE seller_id = OLD.attendee_id;
715     DELETE FROM Resale_Interest WHERE buyer_id = OLD.attendee_id;
716     DELETE FROM Review WHERE attendee_id = OLD.attendee_id;
717 END;
718
719 -- =====
720 -- 9. Auto-match offers
721 -- =====
722
723 -- 31. Match one resale interest with the FIFO offer queue (auto-buy ticket if seller
724 -- exists)
725 CREATE TRIGGER trg_match_resale_interest
726 AFTER INSERT ON Resale_Interest_Type
727 FOR EACH ROW
728 BEGIN
729     DECLARE v_event INT;
730     DECLARE v_type INT;
731     DECLARE v_ticket INT;
732     DECLARE v_offer INT;
733     DECLARE v_buyer INT;
734     DECLARE v_seller INT;
735     DECLARE sActive INT;
736
737     -- lookup status id for 'active'
738     SELECT status_id INTO sActive
739     FROM Ticket_Status WHERE name = 'active' LIMIT 1;
740
741     -- lookup event and buyer
742     SELECT event_id, buyer_id
743     INTO v_event, v_buyer

```

```
743 FROM Resale_Interest
744 WHERE request_id = NEW.request_id;
745
746 -- type requested
747 SET v_type = NEW.type_id;
748
749 -- find first matching offer
750 SELECT ro.offer_id, ro.ticket_id, ro.seller_id
751 INTO v_offer, v_ticket, v_seller
752 FROM Resale_Offer ro
753 JOIN Ticket t ON t.ticket_id = ro.ticket_id
754 WHERE ro.event_id = v_event AND t.type_id = v_type
755 ORDER BY ro.offer_timestamp
756 LIMIT 1;
757
758 -- if match found, reassign ticket + log match
759 IF v_offer IS NOT NULL THEN
760     UPDATE Ticket
761     SET attendee_id = v_buyer,
762         status_id = sActive
763     WHERE ticket_id = v_ticket;
764
765     DELETE FROM Resale_Offer
766     WHERE offer_id = v_offer;
767
768     DELETE FROM Resale_Interest
769     WHERE request_id = NEW.request_id;
770
771     INSERT INTO Resale_Match_Log (
772         match_type, ticket_id, offered_type_id, requested_type_id,
773         buyer_id, seller_id
774     )
775     VALUES ('interest', v_ticket, v_type, v_type, v_buyer, v_seller);
776 END IF;
777 END;
778
779 -- 32. Match one resale offer with the FIFO interest queue (auto-sell ticket if buyer
exists)
780 CREATE TRIGGER trg_match_resale_offer
781 AFTER INSERT ON Resale_Offer
782 FOR EACH ROW
783 BEGIN
784     DECLARE v_event INT;
785     DECLARE v_type INT;
786     DECLARE v_req INT;
787     DECLARE v_buyer INT;
788     DECLARE sActive INT;
789
790     -- lookup status id for 'active'
791     SELECT status_id INTO sActive
792     FROM Ticket_Status WHERE name = 'active' LIMIT 1;
793
794     -- get event id from offer
795     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

```

1  -- -----
2  -- Procedures --
3  -- -----
4
5  USE pulse_university;
6
7  -- Drop all procedures
8  DROP PROCEDURE IF EXISTS UpdateExpiredTickets;
9  DROP PROCEDURE IF EXISTS ExpireResaleOffers;
10 DROP PROCEDURE IF EXISTS ExpireResaleInterests;
11 DROP PROCEDURE IF EXISTS ScanTicket;
12 DROP PROCEDURE IF EXISTS RunMaintenance;
13 DROP PROCEDURE IF EXISTS sp_rename_self;
14 DROP PROCEDURE IF EXISTS check_staff_ratio;
15
16 -- =====
17 -- Procedure 1: Update expired tickets (active / on offer → unused)
18 -- =====
19 CREATE PROCEDURE UpdateExpiredTickets()
20 BEGIN
21     DECLARE sActive INT;
22     DECLARE sOffer INT;
23     DECLARE sUnused INT;
24
25     SELECT status_id INTO sActive FROM Ticket_Status WHERE name = 'active' LIMIT 1;
26     SELECT status_id INTO sOffer FROM Ticket_Status WHERE name = 'on offer' LIMIT 1;
27     SELECT status_id INTO sUnused FROM Ticket_Status WHERE name = 'unused' LIMIT 1;
28
29     UPDATE Ticket t
30         JOIN Event e ON t.event_id = e.event_id
31     SET t.status_id = sUnused
32     WHERE CURDATE() > e.end_dt
33         AND t.status_id IN (sActive, sOffer);
34 END;
35
36 -- =====
37 -- Procedure 2: Expire resale offers after event end
38 -- =====
39 CREATE PROCEDURE ExpireResaleOffers()
40 BEGIN
41     DELETE ro
42     FROM Resale_Offer ro
43         JOIN Event e ON ro.event_id = e.event_id
44     WHERE CURDATE() > e.end_dt;
45 END;
46
47 -- =====
48 -- Procedure 3: Expire resale interests after event end
49 -- =====
50 CREATE PROCEDURE ExpireResaleInterests()
51 BEGIN

```

```

52     DELETE ri
53     FROM   Resale_Interest ri
54           JOIN Event e ON ri.event_id = e.event_id
55     WHERE  CURDATE() > e.end_dt;
56 END;
57
58 -- =====
59 -- Procedure 4: Scan ticket at entrance (active → used)
60 -- =====
61 CREATE PROCEDURE ScanTicket(IN p_ean BIGINT)
62 BEGIN
63     DECLARE v_status INT;
64     DECLARE sActive INT;
65     DECLARE sUsed INT;
66
67     SELECT status_id INTO sActive FROM Ticket_Status WHERE name = 'active' LIMIT 1;
68     SELECT status_id INTO sUsed   FROM Ticket_Status WHERE name = 'used'   LIMIT 1;
69
70     SELECT status_id INTO v_status
71     FROM   Ticket
72     WHERE  ean_number = p_ean;
73
74     IF v_status IS NULL THEN
75         SIGNAL SQLSTATE '45000'
76         SET MESSAGE_TEXT = 'Ticket not found.';
77     ELSEIF v_status <> sActive THEN
78         SIGNAL SQLSTATE '45000'
79         SET MESSAGE_TEXT = 'Ticket already used or inactive.';
80     ELSE
81         UPDATE Ticket
82         SET     status_id = sUsed
83         WHERE  ean_number = p_ean;
84     END IF;
85 END;
86
87 -- =====
88 -- Procedure 5: Run maintenance (expire tickets/offers/interests + ratios)
89 -- =====
90 CREATE PROCEDURE RunMaintenance()
91 BEGIN
92     DECLARE done INT DEFAULT 0;
93     DECLARE ev_id INT;
94
95     -- cursor must be before handler
96     DECLARE cur CURSOR FOR
97         SELECT DISTINCT event_id FROM Works_On;
98
99     -- handler after cursor
100    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
101
102    -- 1. Run expiration subprocedures
103    CALL UpdateExpiredTickets();
104    CALL ExpireResaleOffers();
105    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

```

```
160     IF cap IS NULL THEN
161         SIGNAL SQLSTATE '45000'
162         SET MESSAGE_TEXT = 'Invalid event ID: no matching event found.';
163     END IF;
164
165     -- Count security staff
166     SELECT COUNT(*) INTO sec
167     FROM   Works_On wo
168     JOIN   Staff st ON wo.staff_id = st.staff_id
169     WHERE  wo.event_id = ev
170           AND st.role_id = (
171         SELECT role_id FROM Staff_Role WHERE name = 'security' LIMIT 1
172     );
173
174     -- Count support staff
175     SELECT COUNT(*) INTO sup
176     FROM   Works_On wo
177     JOIN   Staff st ON wo.staff_id = st.staff_id
178     WHERE  wo.event_id = ev
179           AND st.role_id = (
180         SELECT role_id FROM Staff_Role WHERE name = 'support' LIMIT 1
181     );
182
183     -- Check if staffing ratios are met
184     IF sec < CEIL(cap * 0.05) OR sup < CEIL(cap * 0.02) THEN
185         SIGNAL SQLSTATE '45000'
186         SET MESSAGE_TEXT = 'Staffing below required ratio.';
187     END IF;
188 END;
189
```

## sql\views.sql

```

1  -- -----
2  -- --- Views ----
3  -- -----
4
5  USE pulse_university;
6
7  /* ===== drop old versions if they exist ===== */
8  DROP VIEW IF EXISTS View_Yearly_Revenue_By_Method;
9  DROP VIEW IF EXISTS View_Artist_Year_Participation;
10 DROP VIEW IF EXISTS View_Performance_Detail;
11 DROP VIEW IF EXISTS View_Artist_Performance_Rating;
12 DROP VIEW IF EXISTS View_Attendee_Event_Rating;
13 DROP VIEW IF EXISTS View_Attendee_Yearly_Visits;
14 DROP VIEW IF EXISTS View_Genre_Pairs;
15 DROP VIEW IF EXISTS View_Artist_Continents;
16 DROP VIEW IF EXISTS View_Genre_Year_Counts;
17 DROP VIEW IF EXISTS View_Attendee_Artist_Review;
18
19 /* -----
20  * View 1 - yearly ticket revenues by payment method (Q 1)
21  * -----*/
22 CREATE VIEW View_Yearly_Revenue_By_Method AS
23 SELECT
24     e.fest_year,
25     t.method_id,
26     SUM(t.cost) AS total_revenue
27 FROM Ticket t
28 JOIN Event e ON t.event_id = e.event_id
29 GROUP BY e.fest_year, t.method_id;
30
31 /* -----
32  * View 2 - artist participation by year (Q 2)
33  * -----*/
34 CREATE VIEW View_Artist_Year_Participation AS
35 SELECT DISTINCT
36     pa.artist_id,
37     e.fest_year
38 FROM Performance_Artist pa
39 JOIN Performance p ON pa.perf_id = p.perf_id
40 JOIN Event e ON p.event_id = e.event_id;
41
42 /* -----
43  * View 3 - handy performance detail (Q 3)
44  *           one row per performer-performance, with fest year & type
45  * -----*/
46 CREATE VIEW View_Performance_Detail AS
47 SELECT p.perf_id,
48     p.event_id,
49     e.fest_year,
50     pt.name AS perf_type,
51     p.datetime,

```

```

52         pa.artist_id,
53         pb.band_id
54 FROM Performance p
55 JOIN      Event          e  ON p.event_id = e.event_id
56 JOIN      Performance_Type pt ON p.type_id = pt.type_id
57 LEFT JOIN Performance_Artist pa ON pa.perf_id = p.perf_id
58 LEFT JOIN Performance_Band pb ON pb.perf_id = p.perf_id;
59
60 /* -----
61  * View 4 - artist workload & average ratings (Q 4, 11)
62  * -----*/
63 CREATE VIEW View_Artist_Performance_Rating AS
64 SELECT
65     a.artist_id,
66     CONCAT(a.first_name, ' ', a.last_name) AS artist_name,
67     (
68         SELECT COUNT(DISTINCT pa.perf_id)
69         FROM Performance_Artist pa
70         WHERE pa.artist_id = a.artist_id
71     ) AS performance_count,
72     (
73         SELECT AVG(r.interpretation)
74         FROM Performance_Artist pa
75         JOIN Review r ON pa.perf_id = r.perf_id
76         WHERE pa.artist_id = a.artist_id
77     ) AS avg_interpretation,
78     (
79         SELECT AVG(r.overall)
80         FROM Performance_Artist pa
81         JOIN Review r ON pa.perf_id = r.perf_id
82         WHERE pa.artist_id = a.artist_id
83     ) AS avg_overall
84 FROM Artist a;
85
86 /* -----
87  * View 5 - attendee's watched performances & own rating (Q 6)
88  * -----*/
89 CREATE VIEW View_Attendee_Event_Rating AS
90 SELECT
91     att.attendee_id,
92     CONCAT(att.first_name, ' ', att.last_name) AS attendee_name,
93     e.event_id,
94     e.title AS event_title,
95     AVG(r.overall) AS avg_event_rating
96 FROM Ticket t
97 JOIN Event e ON t.event_id = e.event_id
98 JOIN Attendee att ON att.attendee_id = t.attendee_id
99 JOIN Performance p ON p.event_id = e.event_id
100 LEFT JOIN Review r ON r.perf_id = p.perf_id
101                  AND r.attendee_id = att.attendee_id
102 GROUP BY att.attendee_id, attendee_name, e.event_id, e.title;
103
104 /* -----
105  * View 6 - attendee visit count per calendar year (Q 9)

```

```

106  * -----*/
107  CREATE VIEW View_Attendee_Yearly_Visits AS
108  SELECT  t.attendee_id,
109          YEAR(t.purchase_date)      AS festival_year,
110          COUNT(DISTINCT t.event_id) AS events_attended
111  FROM    Ticket t
112  GROUP BY t.attendee_id, YEAR(t.purchase_date);
113
114  /* -----
115  * View 7 - unique artist-genre pairs & artist count (Q 10)
116  *          only for artist that actually have performed
117  * -----*/
118  CREATE VIEW View_Genre_Pairs AS
119  SELECT
120      LEAST(ag1.genre_id, ag2.genre_id) AS genre_id1,
121      GREATEST(ag1.genre_id, ag2.genre_id) AS genre_id2,
122      COUNT(DISTINCT ag1.artist_id)      AS artist_count
123  FROM Artist_Genre ag1
124  JOIN Artist_Genre ag2
125      ON ag1.artist_id = ag2.artist_id AND ag1.genre_id < ag2.genre_id
126  WHERE ag1.artist_id IN (
127      SELECT DISTINCT pa.artist_id
128      FROM Performance_Artist pa
129      JOIN Performance p ON pa.perf_id = p.perf_id
130      JOIN Event       e ON p.event_id = e.event_id
131      JOIN Festival    f ON e.fest_year = f.fest_year
132      WHERE f.fest_year < YEAR(CURDATE())
133  )
134  GROUP BY genre_id1, genre_id2;
135
136  /* -----
137  * View 8 - artists & number of continents performed in (Q 13)
138  * -----*/
139  CREATE VIEW View_Artist_Continents AS
140  SELECT  a.artist_id,
141          CONCAT(a.first_name, ' ', a.last_name) AS artist_name,
142          COUNT(DISTINCT l.continent_id)      AS
continents_performed
143  FROM    Artist a
144  JOIN    Performance_Artist pa ON a.artist_id = pa.artist_id
145  JOIN    Performance      p  ON pa.perf_id = p.perf_id
146  JOIN    Event            e  ON p.event_id = e.event_id
147  JOIN    Festival         f  ON e.fest_year = f.fest_year
148  JOIN    Location         l  ON f.loc_id   = l.loc_id
149  GROUP BY a.artist_id, artist_name;
150
151  /* -----
152  * View 9 - yearly appearances per genre (Q 14)
153  * -----*/
154  CREATE VIEW View_Genre_Year_Counts AS
155  SELECT  g.genre_id,
156          g.name AS genre_name,
157          d.fest_year,
158          COUNT(*) AS perf_count

```



```
159 FROM View_Performance_Detail d
160 JOIN Artist_Genre ag ON ag.artist_id = d.artist_id
161 JOIN Genre g ON g.genre_id = ag.genre_id
162 GROUP BY g.genre_id, g.name, d.fest_year;
163
164 /* -----
165  * View 10 - aggregates per attendee-artist pair (Q 15)
166  * -----*/
167 CREATE VIEW View_Attendee_Artist_Review AS
168 SELECT
169     r.attendee_id,
170     pa.artist_id,
171     SUM(r.interpretation + r.overall) AS total_score
172 FROM Review r
173 JOIN Performance_Artist pa ON r.perf_id = pa.perf_id
174 GROUP BY r.attendee_id, pa.artist_id;
175
```