TP1 — Library Database (PostgreSQL)

Full Correction Script (Q1–Q49)

```
-- ============================================================
-- TP1 (Library) — PostgreSQL Correction Script
-- Covers:
--   Core TP1: Q1–Q16
--   Additional tasks: Q17–Q37
--   Extended/Bonus tasks: Q38–Q49 (transactions, upsert, views, triggers), if time is
enough
-- Notes:
--   - Some statements are meant to demonstrate constraint errors. They are
--     handled with DO/EXCEPTION blocks to keep the script running.
--   - Dates use ISO format (YYYY-MM-DD).
-- ============================================================



-- ============================================================
-- Q1) Create the six tables
-- ============================================================

DROP TABLE IF EXISTS reservation CASCADE;
DROP TABLE IF EXISTS loan CASCADE;
DROP TABLE IF EXISTS librarian CASCADE;
DROP TABLE IF EXISTS member CASCADE;
DROP TABLE IF EXISTS book CASCADE;
DROP TABLE IF EXISTS author CASCADE;

CREATE TABLE author (
  author_id     INTEGER PRIMARY KEY,
  last_name     VARCHAR(50) NOT NULL,
  first_name    VARCHAR(50),
  nationality   VARCHAR(30),
  birth_year    INTEGER
);

CREATE TABLE book (
  book_id           INTEGER PRIMARY KEY,
  title             VARCHAR(100) NOT NULL,
  author_id         INTEGER NOT NULL,
  publication_year  INTEGER,              -- CHECK added in Q11
  category          VARCHAR(30),
  available         BOOLEAN NOT NULL DEFAULT TRUE,
  CONSTRAINT fk_book_author
    FOREIGN KEY (author_id) REFERENCES author(author_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
);

CREATE TABLE member (
  member_id          INTEGER PRIMARY KEY,
  last_name          VARCHAR(30) NOT NULL,
  first_name         VARCHAR(30) NOT NULL,
  email              TEXT,                -- UNIQUE added in Q12
  phone              VARCHAR(10),
  registration_date  DATE NOT NULL DEFAULT CURRENT_DATE
```

```
);

CREATE TABLE librarian (
  librarian_id  INTEGER PRIMARY KEY,
  last_name     VARCHAR(30) NOT NULL,
  first_name    VARCHAR(30) NOT NULL,
  username      VARCHAR(15),              -- NOT NULL added in Q13
  password      TEXT
);

CREATE TABLE loan (
  loan_id      INTEGER PRIMARY KEY,
  book_id      INTEGER NOT NULL,
  member_id    INTEGER NOT NULL,
  loan_date    DATE NOT NULL,
  return_date  DATE,
  CONSTRAINT fk_loan_book
    FOREIGN KEY (book_id) REFERENCES book(book_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT,
  CONSTRAINT fk_loan_member
    FOREIGN KEY (member_id) REFERENCES member(member_id)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);

CREATE TABLE reservation (
  reservation_id    INTEGER PRIMARY KEY,
  book_id           INTEGER NOT NULL,
  member_id         INTEGER NOT NULL,
  reservation_date  DATE NOT NULL,
  CONSTRAINT fk_res_book
    FOREIGN KEY (book_id) REFERENCES book(book_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT,
  CONSTRAINT fk_res_member
    FOREIGN KEY (member_id) REFERENCES member(member_id)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);



-- ============================================================
-- Q2) Verify creation (PostgreSQL)
-- ============================================================
-- In psql: \d author  \d book  \d member  \d librarian  \d loan  \d reservation

SELECT table_name
FROM information_schema.tables
WHERE table_schema='public'
  AND table_name IN ('author','book','member','librarian','loan','reservation')
ORDER BY table_name;



-- ============================================================
-- Q3) Add page_count to Book
-- ============================================================
ALTER TABLE book
ADD COLUMN page_count INTEGER;
```

```
-- ============================================================
-- Q4) Modify Book.category to accept 50 characters
-- ============================================================
ALTER TABLE book
ALTER COLUMN category TYPE VARCHAR(50);



-- ============================================================
-- Q5) Remove page_count from Book
-- ============================================================
ALTER TABLE book
DROP COLUMN page_count;



-- ============================================================
-- Q6) Insert seed population
-- ============================================================

INSERT INTO author (author_id, last_name, first_name, nationality, birth_year) VALUES
(1, 'Ibn Salih',    'Mohammed',     'Saudi',     1925),
(2, 'Ibn Khaldoun', 'Abdelrahman',  'Tunisian',  1332),
(3, 'Hugo',         'Victor',       'French',    1802),
(4, 'Tolstoï',      'Léon',         'Russian',   1828),
(5, 'Tanenbaum',    'Andrew',       'American',  1944),
(6, 'Pujolle',      'Guy',          'French',    1949),
(7, 'Rousseau',     'Jean-Jacques', 'French',    1712),
(8, 'Bennabi',      'Malek',        'Algerian',  1905);

INSERT INTO book (book_id, title, author_id, publication_year, category, available)
VALUES
(1,  'Ouloum Al-Hadith',                    1, 2001, 'Religion',         TRUE),
(2,  'Moqaddima Ibn Khaldoun',              2, 1377, 'History',          TRUE),
(3,  'Notre-Dame de Paris',                3, 1831, 'Novel',            TRUE),
(4,  'Les Misérables',                     3, 1862, 'Novel',            TRUE),
(5,  'Guerre et Paix',                     4, 1869, 'Novel',            TRUE),
(6,  'Systèmes d''exploitation modernes', 5, 2014, 'Computer Science', TRUE),
(7,  'Réseaux informatiques',              5, 2021, 'Computer Science', TRUE),
(8,  'Réseaux : des bases à l''Internet', 6, 2018, 'Computer Science', TRUE),
(9,  'Du Contrat Social',                  7, 1762, 'Politics',         TRUE),
(10, 'Les Conditions de la Renaissance',   8, 1948, 'Philosophy',       TRUE);

INSERT INTO member (member_id, last_name, first_name, email, phone, registration_date)
VALUES
(1, 'Ben Youssef', 'Ibrahim', 'ibrahim.benyoussef@mail.com', '0555123456', '2024-01-
10'),
(2, 'Naimi',       'Khaled',  'khaled.naimi@mail.com',       '0666789012', '2024-02-
15'),
(3, 'Sherif',      'Aicha',   'aicha.sherif@mail.com',       '0777345678', '2024-03-
05'),
(4, 'Arabi',       'Youssef', 'youssef.arabi@mail.com',      '0544987654', '2024-04-
20'),
(5, 'Zaoui',       'Fatima',  'fatima.zaoui@mail.com',       '0566098765', '2024-05-
30');

INSERT INTO librarian (librarian_id, last_name, first_name, username, password) VALUES
(1, 'Eddine',   'Mohamed Amine', 'm.amine',    'hashed_password_1'),
(2, 'Zeitouni', 'Hasna',         'h.zeitouni', 'hashed_password_2');
```

```
INSERT INTO loan (loan_id, book_id, member_id, loan_date, return_date) VALUES
(1, 3, 1, '2024-06-01', '2024-06-16'),
(2, 1, 3, '2024-06-05', NULL),
(3, 5, 2, '2024-06-10', NULL);

INSERT INTO reservation (reservation_id, book_id, member_id, reservation_date) VALUES
(1, 5, 4, '2024-06-12'),
(2, 1, 5, '2024-06-14');



-- ============================================================
-- Q7) Confirm inserted rows
-- ============================================================
SELECT * FROM author ORDER BY author_id;
SELECT * FROM book ORDER BY book_id;
SELECT * FROM member ORDER BY member_id;
SELECT * FROM librarian ORDER BY librarian_id;
SELECT * FROM loan ORDER BY loan_id;
SELECT * FROM reservation ORDER BY reservation_id;



-- ============================================================
-- Q8) Set available=FALSE for books currently on loan (return_date IS NULL)
-- ============================================================
UPDATE book b
SET available = FALSE
WHERE b.book_id IN (
  SELECT l.book_id
  FROM loan l
  WHERE l.return_date IS NULL
);



-- ============================================================
-- Q9) Update return_date for the book borrowed by member (Sherif, Aicha)
--     and set available=TRUE for that book
-- ============================================================
UPDATE loan l
SET return_date = '2024-06-20'
WHERE l.member_id = (
  SELECT m.member_id
  FROM member m
  WHERE m.last_name='Sherif' AND m.first_name='Aicha'
)
AND l.return_date IS NULL;

UPDATE book b
SET available = TRUE
WHERE b.book_id IN (
  SELECT l.book_id
  FROM loan l
  WHERE l.member_id = (
    SELECT m.member_id
    FROM member m
    WHERE m.last_name='Sherif' AND m.first_name='Aicha'
  )
  AND l.return_date = '2024-06-20'
);
```

```sql
-- ============================================================
-- Q10) Delete a member and observe ON DELETE CASCADE on loans/reservations
--     Example: delete member_id=4 (Arabi Youssef)
-- ============================================================
DELETE FROM member
WHERE member_id = 4;

SELECT * FROM loan ORDER BY loan_id;
SELECT * FROM reservation ORDER BY reservation_id;



-- ============================================================
-- Q11) Add CHECK: publication_year > 1800
-- IMPORTANT: the seed data contains publication_year 1377 and 1762.
-- A normal CHECK would FAIL immediately. Use NOT VALID to continue.
-- ============================================================

-- (Run separately to observe the failure; then revert and use NOT VALID.)
-- ALTER TABLE book
-- ADD CONSTRAINT chk_book_pub_year_gt_1800
-- CHECK (publication_year > 1800);

ALTER TABLE book
ADD CONSTRAINT chk_book_pub_year_gt_1800
CHECK (publication_year > 1800) NOT VALID;

-- Optional: validate later AFTER cleaning the violating rows (will fail now):
-- ALTER TABLE book VALIDATE CONSTRAINT chk_book_pub_year_gt_1800;



-- ============================================================
-- Q12) UNIQUE on member.email
-- ============================================================
ALTER TABLE member
ADD CONSTRAINT uq_member_email UNIQUE (email);



-- ============================================================
-- Q13) NOT NULL on librarian.username
-- ============================================================
ALTER TABLE librarian
ALTER COLUMN username SET NOT NULL;



-- ============================================================
-- Q14) Insert duplicate email (observe UNIQUE violation)
-- Handled with exception to keep script running.
-- ============================================================
DO $$
BEGIN
  BEGIN
    INSERT INTO member (member_id, last_name, first_name, email, phone,
registration_date)
    VALUES (99, 'Test', 'Duplicate', 'khaled.naimi@mail.com', '0500000000',
CURRENT_DATE);
  EXCEPTION WHEN unique_violation THEN
```

```
    RAISE NOTICE 'Q14: duplicate email rejected (UNIQUE works).';
  END;
END $$;



-- ============================================================
-- Q15) Insert a book with publication_year=1700 (observe CHECK violation)
-- Note: NOT VALID still enforces NEW rows, so this is rejected.
-- ============================================================
DO $$
BEGIN
  BEGIN
    INSERT INTO book (book_id, title, author_id, publication_year, category, available)
    VALUES (99, 'Test Old Book', 3, 1700, 'History', TRUE);
  EXCEPTION WHEN check_violation THEN
    RAISE NOTICE 'Q15: publication_year=1700 rejected (CHECK works for new rows).';
  END;
END $$;



-- ============================================================
-- Q16) Insert a loan with return_date < loan_date (observe)
-- At this stage (before Q24), no CHECK prevents it.
-- Use a transaction + ROLLBACK so the database is not polluted.
-- ============================================================
BEGIN;
  INSERT INTO loan (loan_id, book_id, member_id, loan_date, return_date)
  VALUES (999, 4, 1, '2024-06-20', '2024-06-10');

  -- Observe it exists inside the transaction:
  SELECT * FROM loan WHERE loan_id = 999;
ROLLBACK;



-- ============================================================
-- ================ ADDITIONAL TASKS (Q17-Q37) ============
-- ============================================================

-- ============================================================
-- Q17) Rename table member -> library_member
-- ============================================================
ALTER TABLE member RENAME TO library_member;



-- ============================================================
-- Q18) Rename column book.title -> book.book_title
-- ============================================================
ALTER TABLE book RENAME COLUMN title TO book_title;



-- ============================================================
-- Q19) Change library_member.phone to VARCHAR(20)
-- ============================================================
ALTER TABLE library_member
ALTER COLUMN phone TYPE VARCHAR(20);



-- ============================================================
```

```
-- Q20) Add created_at TIMESTAMP DEFAULT now() to all tables
-- ============================================================
ALTER TABLE author         ADD COLUMN IF NOT EXISTS created_at TIMESTAMP DEFAULT now();
ALTER TABLE book           ADD COLUMN IF NOT EXISTS created_at TIMESTAMP DEFAULT now();
ALTER TABLE library_member ADD COLUMN IF NOT EXISTS created_at TIMESTAMP DEFAULT now();
ALTER TABLE librarian      ADD COLUMN IF NOT EXISTS created_at TIMESTAMP DEFAULT now();
ALTER TABLE loan           ADD COLUMN IF NOT EXISTS created_at TIMESTAMP DEFAULT now();
ALTER TABLE reservation    ADD COLUMN IF NOT EXISTS created_at TIMESTAMP DEFAULT now();



-- ============================================================
-- Q21) Add updated_at TIMESTAMP to all tables
-- ============================================================
ALTER TABLE author         ADD COLUMN IF NOT EXISTS updated_at TIMESTAMP;
ALTER TABLE book           ADD COLUMN IF NOT EXISTS updated_at TIMESTAMP;
ALTER TABLE library_member ADD COLUMN IF NOT EXISTS updated_at TIMESTAMP;
ALTER TABLE librarian      ADD COLUMN IF NOT EXISTS updated_at TIMESTAMP;
ALTER TABLE loan           ADD COLUMN IF NOT EXISTS updated_at TIMESTAMP;
ALTER TABLE reservation    ADD COLUMN IF NOT EXISTS updated_at TIMESTAMP;



-- ============================================================
-- Q22) Enforce book.available NOT NULL
-- ============================================================
ALTER TABLE book
ALTER COLUMN available SET NOT NULL;



-- ============================================================
-- Q23) CHECK on library_member.phone (digits and optional +)
-- ============================================================
ALTER TABLE library_member
ADD CONSTRAINT chk_library_member_phone_format
CHECK (
  phone IS NULL OR phone ~ '^\+?[0-9]{6,20}$'
);



-- ============================================================
-- Q24) CHECK on loan dates: return_date IS NULL OR return_date >= loan_date
-- ============================================================
ALTER TABLE loan
ADD CONSTRAINT chk_loan_dates
CHECK (
  return_date IS NULL OR return_date >= loan_date
);



-- ============================================================
-- Q25) reservation_date >= registration_date of the corresponding member
-- Cross-table rule => trigger is required.
-- ============================================================
CREATE OR REPLACE FUNCTION trg_reservation_date_vs_registration()
RETURNS trigger
LANGUAGE plpgsql
AS $$
DECLARE
  reg_date date;
BEGIN
```

```
  SELECT lm.registration_date
  INTO reg_date
  FROM library_member lm
  WHERE lm.member_id = NEW.member_id;

  IF reg_date IS NULL THEN
    RAISE EXCEPTION 'Member % does not exist (cannot validate reservation).',
NEW.member_id;
  END IF;

  IF NEW.reservation_date < reg_date THEN
    RAISE EXCEPTION
      'Invalid reservation_date (%): must be >= member registration_date (%).',
      NEW.reservation_date, reg_date;
  END IF;

  RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS reservation_date_check ON reservation;

CREATE TRIGGER reservation_date_check
BEFORE INSERT OR UPDATE OF reservation_date, member_id
ON reservation
FOR EACH ROW
EXECUTE FUNCTION trg_reservation_date_vs_registration();



-- ============================================================
-- Q26) UNIQUE author identity: (last_name, first_name, birth_year)
-- ============================================================
ALTER TABLE author
ALTER COLUMN birth_year SET NOT NULL;

ALTER TABLE author
ADD CONSTRAINT uq_author_identity
UNIQUE (last_name, first_name, birth_year);



-- ============================================================
-- Q27) UNIQUE on librarian.username (if not already enforced)
-- ============================================================
DO $$
BEGIN
  IF NOT EXISTS (
    SELECT 1 FROM pg_constraint WHERE conname = 'uq_librarian_username'
  ) THEN
    ALTER TABLE librarian
    ADD CONSTRAINT uq_librarian_username UNIQUE (username);
  END IF;
END $$;



-- ============================================================
-- Q29) Ensure FK loan(book_id) -> book(book_id) uses ON DELETE RESTRICT
-- (Drop and recreate to make it explicit.)
-- ============================================================
```

```
ALTER TABLE loan DROP CONSTRAINT IF EXISTS fk_loan_book;

ALTER TABLE loan
ADD CONSTRAINT fk_loan_book
FOREIGN KEY (book_id) REFERENCES book(book_id)
ON UPDATE CASCADE
ON DELETE RESTRICT;



-- ============================================================
-- Q30) Create index on book(author_id)
-- ============================================================
CREATE INDEX IF NOT EXISTS idx_book_author_id ON book(author_id);



-- ============================================================
-- Q31) Create composite index on loan(member_id, return_date)
-- ============================================================
CREATE INDEX IF NOT EXISTS idx_loan_member_return ON loan(member_id, return_date);



-- ============================================================
-- Q32) Create UNIQUE index on library_member(email) instead of UNIQUE constraint
-- (Drop constraint then create unique index.)
-- ============================================================
ALTER TABLE library_member DROP CONSTRAINT IF EXISTS uq_member_email;
ALTER TABLE library_member DROP CONSTRAINT IF EXISTS uq_library_member_email;

CREATE UNIQUE INDEX IF NOT EXISTS ux_library_member_email
ON library_member(email);



-- ============================================================
-- Q33) Insert 3 new books and 2 new members (ensure constraints)
-- ============================================================
INSERT INTO library_member (member_id, last_name, first_name, email, phone,
registration_date) VALUES
(6, 'Kaci',   'Samir', 'samir.kaci@mail.com',   '+213551112233', '2024-06-01'),
(7, 'Meziane','Nadia', 'nadia.meziane@mail.com', '0555333444',    '2024-06-02');

INSERT INTO book (book_id, book_title, author_id, publication_year, category,
available) VALUES
(11, 'Le Dernier Jour d''un Condamné', 3, 1829, 'Novel',            TRUE),
(12, 'Computer Networks (4th ed.)',     5, 2003, 'Computer Science', TRUE),
(13, 'Réseaux et Télécoms',             6, 2016, 'Computer Science', TRUE);



-- ============================================================
-- Q34) Update category Computer Science -> CS
-- ============================================================
UPDATE book
SET category = 'CS'
WHERE category = 'Computer Science';



-- ============================================================
-- Q35) Update loan.return_date from NULL to CURRENT_DATE
--     only if loan_date is older than 30 days
```

```sql
-- ============================================================
UPDATE loan
SET return_date = CURRENT_DATE
WHERE return_date IS NULL
  AND (CURRENT_DATE - loan_date) > 30;



-- ============================================================
-- Q36) Delete reservations older than 1 year
-- ============================================================
DELETE FROM reservation
WHERE reservation_date < (CURRENT_DATE - INTERVAL '1 year');



-- ============================================================
-- Q37) Delete a book referenced by a loan (observe FK error)
-- Handled with exception to keep script running.
-- ============================================================
DO $$
BEGIN
  BEGIN
    DELETE FROM book WHERE book_id = 1;
  EXCEPTION WHEN foreign_key_violation THEN
    RAISE NOTICE 'Q37: cannot delete book_id=1 because it is referenced by loan (FK
RESTRICT works).';
  END;
END $$;



-- ============================================================
-- ============== EXTENDED / BONUS TASKS (Q38–Q49) ==========
-- ============================================================

-- ============================================================
-- Q38) Transaction (ROLLBACK): create loan + set book unavailable
-- ============================================================
BEGIN;

  INSERT INTO loan (loan_id, book_id, member_id, loan_date, return_date)
  VALUES (200, 4, 1, CURRENT_DATE, NULL);

  UPDATE book
  SET available = FALSE
  WHERE book_id = 4;

  SELECT book_id, available FROM book WHERE book_id = 4;
  SELECT * FROM loan WHERE loan_id = 200;

ROLLBACK;

SELECT book_id, available FROM book WHERE book_id = 4;
SELECT * FROM loan WHERE loan_id = 200;



-- ============================================================
-- Q39) Transaction (COMMIT): create loan + set book unavailable
-- ============================================================
BEGIN;
```

```
  INSERT INTO loan (loan_id, book_id, member_id, loan_date, return_date)
  VALUES (201, 4, 1, CURRENT_DATE, NULL);

  UPDATE book
  SET available = FALSE
  WHERE book_id = 4;

COMMIT;

SELECT book_id, available FROM book WHERE book_id = 4;
SELECT * FROM loan WHERE loan_id = 201;



-- ============================================================
-- Q40) UPSERT library_member by email: insert else update phone
-- Requires unique index ux_library_member_email.
-- ============================================================
INSERT INTO library_member (member_id, last_name, first_name, email, phone,
registration_date)
VALUES (50, 'Test', 'Upsert', 'nadia.meziane@mail.com', '+213777000111', CURRENT_DATE)
ON CONFLICT (email) DO UPDATE
SET phone = EXCLUDED.phone;



-- ============================================================
-- Q41) UPSERT book by book_id: insert else update category
-- ============================================================
INSERT INTO book (book_id, book_title, author_id, publication_year, category,
available)
VALUES (12, 'Computer Networks (4th ed.)', 5, 2003, 'Networks', TRUE)
ON CONFLICT (book_id) DO UPDATE
SET category = EXCLUDED.category;



-- ============================================================
-- Q42) View v_active_loans (return_date IS NULL)
-- ============================================================
CREATE OR REPLACE VIEW v_active_loans AS
SELECT
  l.loan_id,
  l.loan_date,
  l.return_date,
  b.book_id,
  b.book_title,
  lm.member_id,
  (lm.first_name || ' ' || lm.last_name) AS member_full_name
FROM loan l
JOIN book b ON b.book_id = l.book_id
JOIN library_member lm ON lm.member_id = l.member_id
WHERE l.return_date IS NULL;



-- ============================================================
-- Q43) View v_book_catalog
-- ============================================================
CREATE OR REPLACE VIEW v_book_catalog AS
SELECT
```

```
  b.book_id,
  b.book_title,
  (a.first_name || ' ' || a.last_name) AS author_full_name,
  b.category,
  b.available,
  b.publication_year
FROM book b
JOIN author a ON a.author_id = b.author_id;



-- ============================================================
-- Q44) Drop and recreate a view using CREATE OR REPLACE VIEW
-- (Example: redefine v_book_catalog)
-- ============================================================
CREATE OR REPLACE VIEW v_book_catalog AS
SELECT
  b.book_id,
  b.book_title,
  (a.first_name || ' ' || a.last_name) AS author_full_name,
  b.category,
  b.available
FROM book b
JOIN author a ON a.author_id = b.author_id;



-- ============================================================
-- Q45) Members with > 1 active loan
-- ============================================================
SELECT
  lm.member_id,
  lm.last_name,
  lm.first_name,
  COUNT(*) AS active_loans
FROM loan l
JOIN library_member lm ON lm.member_id = l.member_id
WHERE l.return_date IS NULL
GROUP BY lm.member_id, lm.last_name, lm.first_name
HAVING COUNT(*) > 1
ORDER BY active_loans DESC;



-- ============================================================
-- Q46) Inconsistency: book.available = TRUE but still in active loan
-- ============================================================
SELECT
  b.book_id,
  b.book_title,
  b.available,
  l.loan_id,
  l.loan_date
FROM book b
JOIN loan l ON l.book_id = b.book_id
WHERE b.available = TRUE
  AND l.return_date IS NULL;



-- ============================================================
-- Q47) Reservations for books currently on loan (active)
-- ============================================================
```

```
SELECT
  r.reservation_id,
  r.reservation_date,
  r.book_id,
  b.book_title,
  r.member_id,
  (lm.first_name || ' ' || lm.last_name) AS member_full_name
FROM reservation r
JOIN book b ON b.book_id = r.book_id
JOIN library_member lm ON lm.member_id = r.member_id
WHERE EXISTS (
  SELECT 1
  FROM loan l
  WHERE l.book_id = r.book_id
    AND l.return_date IS NULL
);



-- ============================================================
-- Q48) Trigger to sync book.available with loan activity
--   - AFTER INSERT on loan => available=FALSE
--   - AFTER UPDATE return_date => available=TRUE only if no other active loans remain
-- ============================================================
CREATE OR REPLACE FUNCTION trg_sync_book_availability()
RETURNS trigger
LANGUAGE plpgsql
AS $$
DECLARE
  other_active_loans int;
BEGIN
  IF TG_OP = 'INSERT' THEN
    UPDATE book
    SET available = FALSE
    WHERE book_id = NEW.book_id;
    RETURN NEW;
  END IF;

  IF TG_OP = 'UPDATE' THEN
    IF NEW.return_date IS DISTINCT FROM OLD.return_date THEN
      IF NEW.return_date IS NULL THEN
        UPDATE book
        SET available = FALSE
        WHERE book_id = NEW.book_id;
      ELSE
        SELECT COUNT(*)
        INTO other_active_loans
        FROM loan
        WHERE book_id = NEW.book_id
          AND return_date IS NULL;

        IF other_active_loans = 0 THEN
          UPDATE book
          SET available = TRUE
          WHERE book_id = NEW.book_id;
        END IF;
      END IF;
    END IF;

    RETURN NEW;
  END IF;
```

```
  RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS loan_set_unavailable ON loan;
DROP TRIGGER IF EXISTS loan_set_available_on_return ON loan;

CREATE TRIGGER loan_set_unavailable
AFTER INSERT ON loan
FOR EACH ROW
EXECUTE FUNCTION trg_sync_book_availability();

CREATE TRIGGER loan_set_available_on_return
AFTER UPDATE OF return_date ON loan
FOR EACH ROW
EXECUTE FUNCTION trg_sync_book_availability();



-- ============================================================
-- Q49) Trigger to auto-update updated_at on UPDATE (all tables)
-- ============================================================
CREATE OR REPLACE FUNCTION trg_set_updated_at()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
  NEW.updated_at = now();
  RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS set_updated_at_author ON author;
CREATE TRIGGER set_updated_at_author
BEFORE UPDATE ON author
FOR EACH ROW
EXECUTE FUNCTION trg_set_updated_at();

DROP TRIGGER IF EXISTS set_updated_at_book ON book;
CREATE TRIGGER set_updated_at_book
BEFORE UPDATE ON book
FOR EACH ROW
EXECUTE FUNCTION trg_set_updated_at();

DROP TRIGGER IF EXISTS set_updated_at_library_member ON library_member;
CREATE TRIGGER set_updated_at_library_member
BEFORE UPDATE ON library_member
FOR EACH ROW
EXECUTE FUNCTION trg_set_updated_at();

DROP TRIGGER IF EXISTS set_updated_at_librarian ON librarian;
CREATE TRIGGER set_updated_at_librarian
BEFORE UPDATE ON librarian
FOR EACH ROW
EXECUTE FUNCTION trg_set_updated_at();

DROP TRIGGER IF EXISTS set_updated_at_loan ON loan;
CREATE TRIGGER set_updated_at_loan
```

```
BEFORE UPDATE ON loan
FOR EACH ROW
EXECUTE FUNCTION trg_set_updated_at();

DROP TRIGGER IF EXISTS set_updated_at_reservation ON reservation;
CREATE TRIGGER set_updated_at_reservation
BEFORE UPDATE ON reservation
FOR EACH ROW
EXECUTE FUNCTION trg_set_updated_at();
```