Database Constraints - In-Class Practice Task #5 (15-20 minutes)

Scenario: You're building a restaurant order management system during class.

Task 1: Restaurant Tables (3 minutes)

Create a table called `restaurant_tables` with:

- `table_id` (integer, PRIMARY KEY)
- `table_number` (integer, NOT NULL, UNIQUE)
- `seating_capacity` (integer, NOT NULL) - must be between 2 and 12
- `location` (text, NOT NULL) - must be one of: 'indoor', 'outdoor', 'patio', 'private'
- `is_available` (boolean, NOT NULL, DEFAULT true)
- `notes` (text)

Quick Test:

- Insert 4 tables with different capacities and locations
- Try to insert a table with seating_capacity = 15 (should fail)
- Try to insert a table with location = 'rooftop' (should fail)
- Try to insert a duplicate table_number (should fail)

Task 2: Menu Items with Price Constraints (4 minutes)

Create a table called `menu_items` with:

- `item_id` (integer, PRIMARY KEY)
- `item_name` (text, NOT NULL)
- `category` (text, NOT NULL) - must be one of: 'appetizer', 'main', 'dessert', 'beverage'
- `base_price` (numeric, NOT NULL) - must be between 1 and 200
- `special_price` (numeric) - if not NULL, must be less than base_price
- `is_available` (boolean, NOT NULL, DEFAULT true)
- `preparation_time` (integer) - minutes, must be between 5 and 120
- `calories` (integer) - must be between 0 and 5000
- UNIQUE constraint on (item_name, category)

Why UNIQUE on (item_name, category)? You might have "Caesar Salad" as both an appetizer and a main course!

Quick Test:

- Insert 5 menu items across different categories
- Try to insert an item with special_price > base_price (should fail)
- Try to insert duplicate item_name in the same category (should fail)

- Insert the same item_name in a different category (should succeed)

Task 3: Customers with Contact Validation (3 minutes)

Create a table called `customers` with:

- `customer_id` (integer, PRIMARY KEY)
- `first_name` (text, NOT NULL)
- `last_name` (text, NOT NULL)
- `email` (text, UNIQUE)
- `phone` (text, NOT NULL)
- `loyalty_points` (integer, NOT NULL, DEFAULT 0) - must be >= 0
- `registration_date` (date, NOT NULL, DEFAULT CURRENT_DATE)
- `date_of_birth` (date)
- CHECK: at least one of email or phone must be provided (this is tricky!)

Quick Test:

- Insert 3 customers with valid data
- Try to insert a customer with negative loyalty_points (should fail)
- Try to insert a customer with duplicate email (should fail)

Task 4: Orders with Composite Relationships (5 minutes)

Create a table called `orders` with:

- `order_id` (integer, PRIMARY KEY)
- `table_id` (integer, NOT NULL, REFERENCES restaurant_tables)
- `customer_id` (integer, REFERENCES customers ON DELETE SET NULL)
- `order_datetime` (timestamp, NOT NULL, DEFAULT CURRENT_TIMESTAMP)
- `num_guests` (integer, NOT NULL) - must be between 1 and 20
- `status` (text, NOT NULL, DEFAULT 'pending') - must be one of: 'pending', 'preparing', 'ready', 'served', 'paid', 'cancelled'
- `subtotal` (numeric, NOT NULL, DEFAULT 0) - must be >= 0
- `tax` (numeric, NOT NULL, DEFAULT 0) - must be >= 0
- `tip` (numeric, DEFAULT 0) - must be >= 0
- `total_amount` (numeric, NOT NULL) - must be >= 0
- CHECK: total_amount = subtotal + tax + tip

Why SET NULL for customer_id? We want to keep order history even if a customer is deleted!

Quick Test:

- Insert 3 orders for existing tables and customers
- Try to insert an order with num_guests = 25 (should fail)

- Try to insert an order with status = 'completed' (should fail)
- Try to insert an order where total_amount doesn't equal subtotal + tax + tip (should fail)