```
-- ============================================================
-- OPERATIONAL QUERIES
-- ============================================================

-- 1. Find all reservations for a specific date
CREATE OR REPLACE VIEW daily_reservations AS
SELECT
    r.reservation_id,
    c.first_name,
    c.last_name,
    r.reservation_time,
    r.party_size,
    t.table_number,
    r.special_requests,
    r.status
FROM
    Reservations r
JOIN
    Customers c ON r.customer_id = c.customer_id
JOIN
    Tables t ON r.table_id = t.table_id
WHERE
    r.reservation_date = CURRENT_DATE  -- Can be parameterized
ORDER BY
    r.reservation_time;

-- 2. Track daily sales by category
CREATE OR REPLACE VIEW daily_sales_by_category AS
SELECT
    mc.name AS category,
    SUM(oi.price * oi.quantity) AS total_sales
FROM
    Order_Items oi
JOIN
    Menu m ON oi.dish_id = m.dish_id
JOIN
    Menu_Categories mc ON m.category_id = mc.category_id
JOIN
    Orders o ON oi.order_id = o.order_id
WHERE
    o.order_date = CURRENT_DATE  -- Can be parameterized
GROUP BY
    mc.name
ORDER BY
    total_sales DESC;

-- 3. Most popular menu items for current month
CREATE OR REPLACE VIEW popular_dishes AS
SELECT
    m.dish_name,
    COUNT(oi.order_item_id) AS times_ordered,
    SUM(oi.quantity) AS total_quantity,
    SUM(oi.price * oi.quantity) AS total_revenue
FROM
    Order_Items oi
JOIN
    Menu m ON oi.dish_id = m.dish_id
JOIN
```

```
    Orders o ON oi.order_id = o.order_id
WHERE
    EXTRACT(MONTH FROM o.order_date) = EXTRACT(MONTH FROM CURRENT_DATE)
    AND EXTRACT(YEAR FROM o.order_date) = EXTRACT(YEAR FROM
CURRENT_DATE)
GROUP BY
    m.dish_id, m.dish_name
ORDER BY
    times_ordered DESC
LIMIT 10;

-- 4. Active tables and their status
CREATE OR REPLACE VIEW active_tables AS
SELECT
    t.table_id,
    t.table_number,
    t.capacity,
    t.section,
    t.status,
    CASE
        WHEN o.order_id IS NOT NULL THEN o.order_id
        ELSE NULL
    END AS active_order_id,
    CASE
        WHEN o.order_id IS NOT NULL THEN o.order_time
        ELSE NULL
    END AS order_start_time,
    CASE
        WHEN r.reservation_id IS NOT NULL AND t.status = 'Reserved'
THEN r.reservation_time
        ELSE NULL
    END AS upcoming_reservation
FROM
    Tables t
LEFT JOIN
    Orders o ON t.table_id = o.table_id AND o.status IN ('Placed',
'Preparing', 'Served')
LEFT JOIN
    Reservations r ON t.table_id = r.table_id
                   AND r.reservation_date = CURRENT_DATE
                   AND r.status = 'Confirmed'
                   AND t.status = 'Reserved'
ORDER BY
    t.section, t.table_number;

-- 5. Check availability for a reservation
-- This is a parameterized query, example usage:
/*
SELECT * FROM check_table_availability('2023-10-20', '19:00:00', 4);
*/
CREATE OR REPLACE FUNCTION check_table_availability(
    p_date DATE,
    p_time TIME,
    p_party_size INTEGER
) RETURNS TABLE (
    table_id INTEGER,
    table_number VARCHAR(10),
    capacity INTEGER,
```

```
    section VARCHAR(20)
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        t.table_id,
        t.table_number,
        t.capacity,
        t.section
    FROM
        Tables t
    WHERE
        t.capacity >= p_party_size
        AND t.status = 'Available'
        AND NOT EXISTS (
            SELECT 1 FROM Reservations r
            WHERE r.table_id = t.table_id
            AND r.reservation_date = p_date
            AND r.status = 'Confirmed'
            AND (
                -- Check if the requested time conflicts with existing
reservations
                -- Assuming an average dining time of 2 hours
                (r.reservation_time <= p_time AND r.reservation_time +
INTERVAL '2 hours' > p_time)
                OR (p_time <= r.reservation_time AND p_time + INTERVAL
'2 hours' > r.reservation_time)
            )
        )
    ORDER BY
        ABS(t.capacity - p_party_size), -- Get tables closest to party
size
        t.section;
END;
$$ LANGUAGE plpgsql;

-- ============================================================
-- MANAGEMENT REPORTS
-- ============================================================

-- 1. Staff productivity report
CREATE OR REPLACE VIEW staff_productivity AS
SELECT
    s.employee_id,
    s.first_name,
    s.last_name,
    COUNT(o.order_id) AS orders_served,
    SUM(o.final_amount) AS total_sales,
    AVG(p.tip_amount) AS average_tip
FROM
    Staff s
JOIN
    Orders o ON s.employee_id = o.server_id
JOIN
    Payments p ON o.order_id = p.order_id
WHERE
    o.order_date BETWEEN CURRENT_DATE - INTERVAL '30 days' AND
CURRENT_DATE
```

```
GROUP BY
    s.employee_id, s.first_name, s.last_name
ORDER BY
    total_sales DESC;

-- 2. Inventory usage projection
CREATE OR REPLACE VIEW inventory_projection AS
SELECT
    i.  ingredient_id,
    i.  name,
    i.  quantity_on_hand,
    i.  unit,
    COALESCE(SUM(mi.quantity * oi.quantity), 0) AS projected_usage,
    i.  quantity_on_hand - COALESCE(SUM(mi.quantity * oi.quantity), 0)
        AS remaining_inventory,
    i.  reorder_level,
    CASE
        WHEN i.quantity_on_hand - COALESCE(SUM(mi.quantity *
oi.quantity), 0) < i.reorder_level THEN TRUE
        ELSE FALSE
    END AS needs_reorder,
    s.name AS supplier,
    s.phone AS supplier_phone
FROM
    Ingredients i
LEFT JOIN
    Menu_Item_Ingredients mi ON i.ingredient_id = mi.ingredient_id
LEFT JOIN
    Order_Items oi ON mi.dish_id = oi.dish_id
LEFT JOIN
    Orders o ON oi.order_id = o.order_id AND o.order_date BETWEEN
CURRENT_DATE - INTERVAL '7 days' AND CURRENT_DATE
LEFT JOIN
    Suppliers s ON i.supplier_id = s.supplier_id
GROUP BY
    i.  ingredient_id, i.name, i.quantity_on_hand, i.unit,
        i.reorder_level, s.name, s.phone
ORDER BY
    needs_reorder DESC, remaining_inventory;

-- 3. Revenue by day of week (helps with staffing decisions)
CREATE OR REPLACE VIEW revenue_by_day AS
SELECT
    TO_CHAR(o.order_date, 'Day') AS day_of_week,
    COUNT(o.order_id) AS order_count,
    ROUND(AVG(o.final_amount), 2) AS average_order_value,
    SUM(o.final_amount) AS total_revenue
FROM
    Orders o
WHERE
    o.order_date BETWEEN CURRENT_DATE - INTERVAL '90 days' AND
CURRENT_DATE
GROUP BY
    TO_CHAR(o.order_date, 'Day'), EXTRACT(DOW FROM o.order_date)
ORDER BY
    EXTRACT(DOW FROM o.order_date);

-- 4. Customer loyalty report
```

```
CREATE OR REPLACE VIEW customer_loyalty AS
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    c.email,
    c.phone,
    c.loyalty_points,
    COUNT(o.order_id) AS visit_count,
    MAX(o.order_date) AS last_visit,
    CURRENT_DATE - MAX(o.order_date) AS days_since_last_visit,
    SUM(o.final_amount) AS total_spent,
    ROUND(AVG(o.final_amount), 2) AS average_order_value
FROM
    Customers c
LEFT JOIN
    Orders o ON c.customer_id = o.customer_id
GROUP BY
    c.customer_id, c.first_name, c.last_name, c.email, c.phone,
c.loyalty_points
ORDER BY
    c.loyalty_points DESC;

-- 5. Menu item profitability analysis
CREATE OR REPLACE VIEW menu_profitability AS
WITH ingredient_costs AS (
    SELECT
        mi.dish_id,
        SUM(mi.quantity * i.cost_per_unit) AS total_ingredient_cost
    FROM
        Menu_Item_Ingredients mi
    JOIN
        Ingredients i ON mi.ingredient_id = i.ingredient_id
    GROUP BY
        mi.dish_id
)
SELECT
    m.dish_id,
    m.dish_name,
    mc.name AS category,
    m.price,
    COALESCE(ic.total_ingredient_cost, 0) AS ingredient_cost,
    m.price - COALESCE(ic.total_ingredient_cost, 0) AS gross_profit,
    CASE
        WHEN m.price > 0 THEN
            ROUND(((m.price - COALESCE(ic.total_ingredient_cost, 0)) /
m.price) * 100, 2)
        ELSE 0
    END AS profit_margin_percent,
    COUNT(oi.order_item_id) AS times_ordered,
    SUM(oi.quantity) AS quantity_sold,
    SUM(oi.price * oi.quantity) AS total_revenue,
    SUM((oi.price - COALESCE(ic.total_ingredient_cost, 0)) *
oi.quantity) AS total_profit
FROM
    Menu m
JOIN
    Menu_Categories mc ON m.category_id = mc.category_id
```

```
LEFT JOIN
    ingredient_costs ic ON m.dish_id = ic.dish_id
LEFT JOIN
    Order_Items oi ON m.dish_id = oi.dish_id
LEFT JOIN
    Orders o ON oi.order_id = o.order_id
WHERE
    o.order_date IS NULL OR -- Include menu items with no orders
    o.order_date BETWEEN CURRENT_DATE - INTERVAL '30 days' AND
CURRENT_DATE
GROUP BY
    m.dish_id, m.dish_name, mc.name, m.price, ic.total_ingredient_cost
ORDER BY
    profit_margin_percent DESC;

-- ============================================================
-- TRANSACTION PROCEDURES
-- ============================================================

-- 1. Create a new reservation
CREATE OR REPLACE PROCEDURE create_reservation(
    p_customer_id INTEGER,
    p_reservation_date DATE,
    p_reservation_time TIME,
    p_party_size INTEGER,
    p_special_requests TEXT DEFAULT NULL,
    p_reserved_by VARCHAR(50) DEFAULT NULL,
    p_contact_phone VARCHAR(20) DEFAULT NULL
) AS $$
DECLARE
    v_table_id INTEGER;
BEGIN
    -- Find an appropriate table
    SELECT table_id INTO v_table_id
    FROM check_table_availability(p_reservation_date,
p_reservation_time, p_party_size)
    LIMIT 1;

    IF v_table_id IS NULL THEN
        RAISE EXCEPTION 'No tables available for the requested time and
party size';
    END IF;

    -- Create the reservation
    INSERT INTO Reservations (
        customer_id,
        reservation_date,
        reservation_time,
        party_size,
        table_id,
        special_requests,
        reserved_by,
        contact_phone
    ) VALUES (
        p_customer_id,
        p_reservation_date,
        p_reservation_time,
        p_party_size,
```

```
        v_table_id,
        p_special_requests,
        p_reserved_by,
        p_contact_phone
    );

    -- Update table status
    UPDATE Tables
    SET status = 'Reserved'
    WHERE table_id = v_table_id;

    COMMIT;
END;
$$ LANGUAGE plpgsql;

-- 2. Create a new order
CREATE OR REPLACE PROCEDURE create_order(
    p_table_id INTEGER,
    p_customer_id INTEGER,
    p_server_id INTEGER,
    p_special_instructions TEXT DEFAULT NULL
) AS $$
DECLARE
    v_order_id INTEGER;
BEGIN
    -- Create the order
    INSERT INTO Orders (
        table_id,
        customer_id,
        server_id,
        special_instructions
    ) VALUES (
        p_table_id,
        p_customer_id,
        p_server_id,
        p_special_instructions
    ) RETURNING order_id INTO v_order_id;

    -- Update table status
    UPDATE Tables
    SET status = 'Occupied'
    WHERE table_id = p_table_id;

    COMMIT;

    -- Return the order_id for adding items
    RAISE NOTICE 'Created order ID: %', v_order_id;
END;
$$ LANGUAGE plpgsql;

-- 3. Add item to order
CREATE OR REPLACE PROCEDURE add_order_item(
    p_order_id INTEGER,
    p_dish_id INTEGER,
    p_quantity INTEGER DEFAULT 1,
    p_modifications TEXT DEFAULT NULL
) AS $$
DECLARE
```

```
    v_price NUMERIC(10,2);
BEGIN
    -- Get the current price of the dish
    SELECT price INTO v_price
    FROM Menu
    WHERE dish_id = p_dish_id;

    IF v_price IS NULL THEN
        RAISE EXCEPTION 'Invalid dish ID';
    END IF;

    -- Add the item to the order
    INSERT INTO Order_Items (
        order_id,
        dish_id,
        quantity,
        price,
        modifications,
        sent_to_kitchen_time
    ) VALUES (
        p_order_id,
        p_dish_id,
        p_quantity,
        v_price,
        p_modifications,
        CURRENT_TIMESTAMP
    );

    -- Update order totals
    UPDATE Orders
    SET
        total_amount = total_amount + (v_price * p_quantity),
        tax_amount = (total_amount + (v_price * p_quantity)) * 0.08, --
Assuming 8% tax
        final_amount = (total_amount + (v_price * p_quantity)) * 1.08 -
discount_amount
    WHERE order_id = p_order_id;

    COMMIT;
END;
$$ LANGUAGE plpgsql;

-- 4. Complete an order and process payment
CREATE OR REPLACE PROCEDURE complete_order(
    p_order_id INTEGER,
    p_payment_method VARCHAR(50),
    p_tip_amount NUMERIC(10,2) DEFAULT 0,
    p_card_last_four VARCHAR(4) DEFAULT NULL
) AS $$
DECLARE
    v_final_amount NUMERIC(10,2);
    v_table_id INTEGER;
BEGIN
    -- Get the final amount and table_id
    SELECT final_amount, table_id INTO v_final_amount, v_table_id
    FROM Orders
    WHERE order_id = p_order_id;
```

```
    IF v_final_amount IS NULL THEN
        RAISE EXCEPTION 'Invalid order ID';
    END IF;

    -- Create payment record
    INSERT INTO Payments (
        order_id,
        amount,
        tip_amount,
        payment_method,
        card_last_four,
        transaction_id,
        receipt_number
    ) VALUES (
        p_order_id,
        v_final_amount,
        p_tip_amount,
        p_payment_method,
        p_card_last_four,
        'TXN-' || TO_CHAR(CURRENT_TIMESTAMP, 'YYYYMMDDHH24MISS') || '-'
|| p_order_id,
        'RCT-' || TO_CHAR(CURRENT_TIMESTAMP, 'YYYYMMDDHH24MISS') || '-'
|| p_order_id
    );

    -- Update order status
    UPDATE Orders
    SET status = 'Completed'
    WHERE order_id = p_order_id;

    -- Update table status
    UPDATE Tables
    SET status = 'Available'
    WHERE table_id = v_table_id;

    -- Update customer loyalty points (if applicable)
    UPDATE Customers
    SET
        loyalty_points = loyalty_points + (v_final_amount * 0.1), --
Assuming 10% of bill as points
        last_visit_date = CURRENT_DATE
    FROM Orders
    WHERE Orders.order_id = p_order_id
    AND Customers.customer_id = Orders.customer_id;

    COMMIT;
END;
$$ LANGUAGE plpgsql;

-- 5. Update inventory after an order is completed
CREATE OR REPLACE FUNCTION update_inventory() RETURNS TRIGGER AS $$
BEGIN
    -- Only process when an order is marked as completed
    IF NEW.status = 'Completed' AND (OLD.status IS NULL OR OLD.status
<> 'Completed') THEN
        -- Update ingredient quantities based on order items
        UPDATE Ingredients
```

```
        SET quantity_on_hand = quantity_on_hand - (mi.quantity *
oi.quantity)
        FROM Order_Items oi
        JOIN Menu_Item_Ingredients mi ON oi.dish_id = mi.dish_id
        WHERE oi.order_id = NEW.order_id
        AND Ingredients.ingredient_id = mi.ingredient_id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create trigger to update inventory when order is completed
CREATE TRIGGER trg_update_inventory
AFTER UPDATE ON Orders
FOR EACH ROW
EXECUTE FUNCTION update_inventory();
```