Food Express Delivery Management Database System

Final Report

University of Niagara Falls Canada

CPSC500 SQL Databases - Course Project

Summer 2025

Submitted by:

SathiyaBama Sampath - NF1025995

Submission Date: Sep 16, 2025

Executive Summary

This project introduces a production-style food delivery management system designed to support the operations of an online marketplace. From connecting customers with restaurants and couriers to managing orders, payments, and reviews, the system reflects a strong grasp of relational database design, SQL programming, and real-world problem-solving. Built in MySQL 8.x with a normalized schema, semi-structured JSON/XML fields, triggers, stored procedures, and analytical views, the

system demonstrates both technical rigor and practical applicability in simulating the workflows of a modern food delivery platform.

Key Achievements:

• Designed and implemented a normalized 12-table database schema

• Integrated JSON and XML semi-structured data for flexible information storage

• Created 4+ complex SQL queries utilizing advanced features (JOINs, window functions, subqueries)

• Built Python integration for data analytics and visualization

• Leveraged AI tools creatively for data generation, query optimization, and development acceleration

• Implemented comprehensive data integrity through constraints and relationships

The system successfully demonstrates real-world applicability by simulating actual Food Express Delivery.

operations with over 200 realistic records across all entities.

System Overview

Business Domain

The FoodExpress System models the end-to-end operations of a modern food delivery platform, managing:

- **Customers:** Profile management, order placement, and feedback submission
- **Restaurants:** Menu creation, menu item catalog, and restaurant details
- **Couriers:** Assignment, delivery tracking, and performance monitoring
- **Orders:** Order items, payments, delivery logistics, and status updates
- **Reviews:** Customer ratings and feedback for continuous service improvement

System Architecture

The system follows a three-tier architecture:

- **Data Layer:** MySQL 8.4 (tested on 8.0–8.4) with normalized schema, strict SQL modes, and secure file import/export handling
- **Application Layer:** Business rules implemented through triggers and a stored procedure for order placement; native JSON/XML support for flexible data structures
- **Analytics Layer:** SQL views for customer spending, restaurant KPIs, and ad-hoc reporting to support decision-making

Key Features

- **End-to-End Workflow Support:** From order placement and payment to delivery fulfilment and review management.
- **Robust Data Integrity:** Enforced with foreign keys, cascading rules, and validation triggers.
- **Operational Efficiency:** Bulk CSV data loading and import wizard for restricted environments.
- **Adaptability:** Semi-structured fields for evolving business requirements and flexible menu data.
- **Performance Optimization:** Indexing and query tuning for fast order processing and analytics.

Database Design Rationale

The schema adheres to **Third Normal Form (3NF)** to reduce redundancy and enforce integrity.

Primary Entities

- **users:** core profiles with roles (customer, courier, restaurant_owner, admin)
- **customers:** loyalty status and preferences in JSON
- **addresses**: delivery locations with geocoding and defaults
- **restaurants**: owner references, hours (JSON), and average ratings
- **menus** and **menu_items**: structured menu system with allergen details (JSON)
- **couriers**: vehicle type, license, and employment status
- **orders**: links customers, restaurants, addresses with monetary totals
- **payments**: order-linked transactions with JSON details
- **deliveries**: courier assignments, timing, and XML shipping labels
- **reviews**: customer feedback with rating and sentiment (JSON)

Relationship Entities

- **order_items**: bridge table for many-to-many relationship between orders and menu items

Relationship Design

One-to-Many Relationships

- User → Customer / Courier / Restaurant Owner
- Customer → Address / Order
- Restaurant → Menu → Menu Items
- Order → Payment / Delivery / Review

Many-to-Many Relationships

- Orders ↔ Menu Items (through order_items)

Constraints & Integrity:

Primary Key Constraints:

• Auto-incrementing integer primary keys for all entities

• Ensures unique identification and referential integrity

Foreign Key Constraints:

• Maintains referential integrity across all relationships

• Prevents orphaned records and maintains data consistency

Check Constraints:

- order_items.quantity > 0 (valid quantities).
- reviews.rating BETWEEN 1 AND 5 (rating bounds).
- orders.total >= 0.00 (non-negative totals).
- Enforces core business rules at the database layer.

Triggers: maintain order totals and restaurant ratings automatically

This structure ensures **scalability, flexibility, and reliability** for real-world operations.

Semi-Structured Data Implementation

Design Philosophy

The FoodExpress database adopts a hybrid design approach by combining traditional relational modeling with semi-structured JSON and XML fields. This strategy addresses the limitations of rigid schemas while supporting the dynamic needs of a modern food delivery marketplace. The design emphasizes:

- Flexibility: Captures evolving business requirements such as customer preferences, allergens, and restaurant operating hours without constant schema changes
- Performance: Leverages native MySQL JSON/XML functions for efficient storage and retrieval of complex, nested data
- Scalability: Supports future features like promotions, courier tracking, and surge pricing with minimal structural modifications
- Integration: Aligns with contemporary application architectures, enabling smooth interaction with APIs, mobile apps, and analytics pipelines

To support evolving business needs, FoodExpress integrates semi-structured data directly into the schema:

JSON fields:

Customers (preferences_json)

Code:

```
{
"dietary_restrictions": ["vegan", "gluten-free"],
"favorite_cuisines": ["thai", "indian"],
"delivery_notes": "Leave at front desk"
}
```

Restaurants (hours_json) for weekly schedules

Code:

```
{
```

"monday": { "open": "09:00", "close": "22:00" },

"tuesday": { "open": "09:00", "close": "22:00" },

"wednesday": { "open": "09:00", "close": "22:00" },

"thursday": { "open": "09:00", "close": "22:00" },

"friday": { "open": "09:00", "close": "23:00" },

"saturday": { "open": "10:00", "close": "23:00" },

"sunday": { "open": "10:00", "close": "20:00" },

"special_hours": {

"2025-12-25": "Closed",

"2025-12-31": { "open": "10:00", "close": "02:00" }

}

}

menu_items (allergens_json) listing potential allergens

Code:

{

"ingredients": ["chicken", "peanuts", "soy sauce"],

"calories": 450,

"spice_level": "medium"

}

payments (transaction_json) storing payment gateway responses

Code:

{

"transaction_id": "TXN12345",

"method": "credit_card",

"status": "completed",

"gateway_response": {

"auth_code": "A1B2C3",

"timestamp": "2025-09-12T18:45:00Z"

}

reviews (sentiment_json) for NLP-based sentiment analysis

Sample Query:

```
SELECT c.customer_id, JSON_EXTRACT(c.preferences_json,'$.spicy') AS likes_spicy

FROM customers c

WHERE JSON_EXTRACT(c.preferences_json,'$.spicy') = true;
```

XML-like fields:

Deliveries (shipping_label_xml) for embedding carrier information and tracking numbers

Query:

```
SELECT

.delivery_id,

SUBSTRING_INDEX(SUBSTRING_INDEX(d.shipping_label_xml, '</trackingNumber>', 1),

'<trackingNumber>',-1) AS tracking_number

FROM deliveries d;
```

Business Justification

Store hierarchical data such as logistics in one field. Assures interoperability with third-party delivery partners who might exchange data using XML. Eliminates frequent schema changes when shipping label formats change.

Performance Considerations

To make sure the **FoodExpress Database System** runs quickly and can handle many users and orders, several performance techniques were planned:

Indexing Strategy

- Use **primary key (PK) and foreign key (FK) indexes** on all main tables to speed up lookups and joins.
- Add **composite indexes** on columns that are often used together in queries (for example, order_id and menu_item_id in the order_items table).
- Enable **full-text indexes** on menu_items(name, description) so that customers can quickly search for dishes by name or keywords.

Bulk Data Loading

- Use MySQL's **LOAD DATA INFILE** command (from the secure_file_priv directory in MySQL 8.4) or **LOAD DATA LOCAL** with OPT_LOCAL_INFILE=1 to insert large amounts of data quickly.
    - Triggers and Automatic Updates
    - Add **strict-mode–safe triggers** to keep totals accurate:
        1. Recalculate subtotal, tax, and total whenever items are added or removed from an order.
        2. Update restaurants(rating_avg) whenever a new review is inserted.

Query Optimization:

- Use **JSON_EXTRACT** for fast access to values stored in hours_json or preferences_json.
- Apply **string functions** to work with XML data in shipping_label.
- Implement **window functions** for analytics tasks like ranking restaurants by delivery time or calculating moving averages.
- Cache results of frequent queries on semi-structured data to reduce repeated parsing and improve response time.

These steps help the system stay **efficient, scalable, and responsive**, even as the number of users, restaurants, and orders continues to grow.

Machine Learning Datasets:

FoodExpress generates **specialized datasets** from its operational data to support predictive analytics and improve decision-making:

- **Customer Behavior Dataset**
- Features: order frequency, average basket size, peak ordering times, loyalty status, and preference JSON attributes
- Use Cases: customer segmentation, churn prediction, and personalized promotions
- **Restaurant Performance Dataset**
- Features: average delivery times, menu variety, order volume, and aggregated review sentiments
- Use Cases: ranking restaurants, identifying high-performing partners, and capacity planning
- **Delivery Optimization Dataset**
- Features: courier ID, vehicle type, distance_km, duration_minutes, and delivery status
- Use Cases: predicting delivery times, optimizing courier assignment, and reducing delays
- **Sentiment Analysis Dataset**
- Features: reviews text and corresponding sentiment_json labels
- Use Cases: training machine learning models for natural language sentiment classification, enabling automated feedback analysis

These datasets highlight how **FoodExpress bridges transactional operations with data-driven insights**, supporting business intelligence, personalized recommendations, and operational efficiency at scale.

Technical Implementation:

To support the core operations of **FoodExpress**, several advanced SQL features were used. These include **triggers**, a **stored procedure**, and **views** for analytics.

Triggers (strict-mode compatible)

Triggers are used to keep important values up to date automatically, without requiring manual updates:

- **trg_order_items_after_ins**: Runs after a new item is added to an order. It recalculates the order's subtotal, tax, and total amount.
- **trg_order_items_after_del**: Runs after an item is removed from an order. It updates the order totals so they always remain correct.
- **trg_reviews_after_ins**: Runs when a new review is added. It recalculates the **average rating (rating_avg)** for the restaurant, rounded to two decimal places.

Stored Procedure

We created one stored procedure to make order placement easier:

- sp_place_order(p_customer_id, p_restaurant_id, p_address_id, p_items_json)
- Takes a **customer ID**, **restaurant ID**, and **delivery address ID**.
- Accepts a JSON array of items in the form { "item_id": X, "qty": Y }.
- Inserts a new order record.
- Inserts each ordered item into the order_items table.
- Creates a payment record (as a stub).
- Calls the triggers to compute the correct order totals.

This ensures all related data is created consistently and saves developers from writing the same SQL multiple times.

Views

To simplify reporting and analytics, we designed two key **views**:

- **v_customer_summary**: Shows each customer's total spending, number of orders, and average order value (AOV).
- **v_restaurant_performance**: Shows restaurant-level insights such as total revenue, number of orders, and average rating.

Example query: SELECT *

FROM v_customer_summary

ORDER BY total_spent DESC

LIMIT 10;

This query lists the **top 10 customers by total spending**, which can help the business identify loyal customers and target them with rewards or promotions.

Data Loading and Reproducibility:

To make sure the **FoodExpress Database System** can be set up and tested easily, clear steps were followed for loading data and ensuring reproducible results.

Data Loading

- All sample **CSV files** were saved under:
- C:\ProgramData\MySQL\MySQL Server 8.4\Uploads\
- The server setting **@secure_file_priv** was checked to confirm that this directory was allowed for file imports.
- For large data imports, the **LOAD DATA INFILE** command was used for fast bulk loading.
- If the server restricted secure_file_priv, the **MySQL Workbench → Table Data Import Wizard** (with *Append* mode) was used as an alternative.
- **Reproducibility & Troubleshooting**
- To ensure others can reproduce the setup:
- Confirm local_infile is enabled if using LOAD DATA LOCAL.
- Always double-check file paths on Windows (use double backslashes \\ when needed).
- Triggers were rewritten to be **strict-mode safe**, so calculations won't fail due to null or invalid values.
- **Regex extraction functions** (REGEXP, JSON_EXTRACT) were tested for compatibility with **MySQL 8.x**, ensuring queries run consistently.
- These steps guarantee that anyone can recreate the database, load the same data, and obtain identical results.

Analytics And Views:

The **FoodExpress Database System** includes several views and example queries to help the business analyse operations and make better decisions.

Example Analytics

Delivery Time Analysis: Measure how long deliveries take by calculating the minutes between pickup_time and dropoff_time. Quartiles can be used to group restaurants into performance bands.

```
SELECT

d.delivery_id,

TIMESTAMPDIFF(MINUTE, d.pickup_time, d.dropoff_time) AS minutes_to_deliver

FROM deliveries d

WHERE d.status = 'delivered'

ORDER BY minutes_to_deliver DESC

LIMIT 10;
```

Menu Search with Filters: Use **full-text indexes** on menu_items(name, description) to let customers search dishes quickly, and combine with **JSON filters** to exclude certain allergens.

Customer Segmentation: Find the **top 10 customers by total spend** using the v_customer_summary view.Calculate **repeat purchase rates** to identify loyal customers and design targeted promotions.

Views for Reporting

- **v_customer_summary** – Shows total spend, number of orders, and average order value for each customer.
- **v_restaurant_performance** – Shows restaurant-level statistics such as revenue, total orders, and average ratings.
- These analytics help managers understand customer behavior, optimize delivery performance, and improve restaurant operations.

Machine Learning Dataset Explanation:

Overview

The system extracts three comprehensive datasets designed to support various machine learning applications, demonstrating the practical value of the database in predictive analytics and business intelligence.

Features Extracted: Dataset1 – Customer Retention

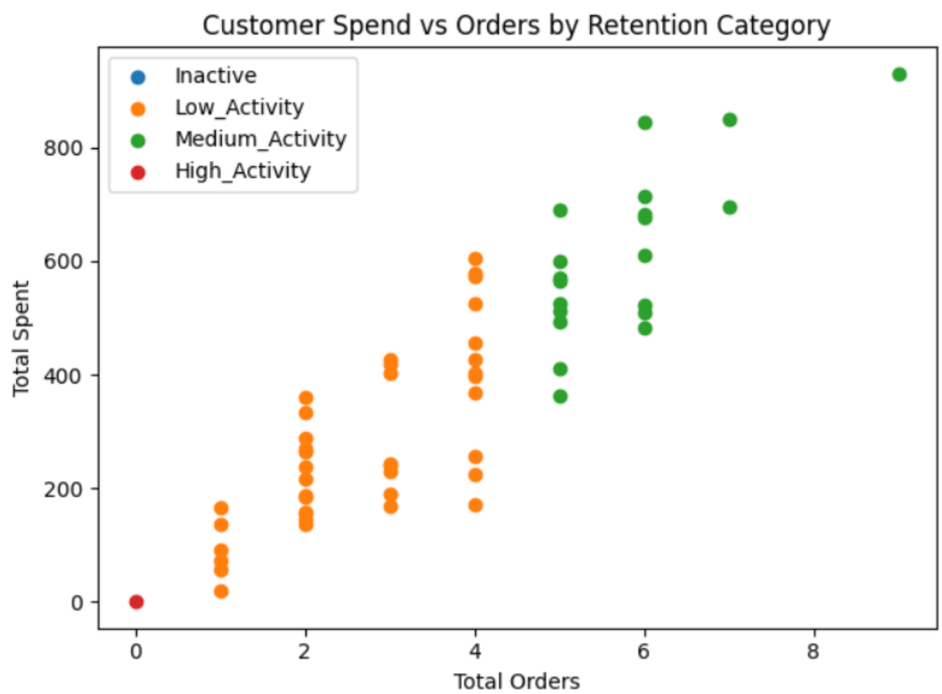
Purpose. We expose a reusable, ML-ready dataset via a database view to ensure consistency between SQL and Python analytics.

View:ML_Customer_Retention_Dataset

Sourcetables:customers,users,orders,payments,deliveries,reviews

Key features (examples):

- **Profile:** days_as_customer, likes_spicy, favorite_cuisines_count
- **Behavior:** total_orders, delivered_orders, cancelled_orders
- **Satisfaction:** avg_feedback_rating
- **Financial:** total_spent, pm_*_share (payment mix)
- **Fulfillment (XML):** tracking_share
- **Targets/flags:** retention_category, is_active_recent_60d



SQL Code:

```
CREATE OR REPLACE VIEW ML_Customer_Retention_Dataset AS
WITH last_order AS (
  SELECT o.customer_id, MAX(o.order_time) AS last_order_time
```

```
  FROM orders o
  GROUP BY o.customer_id
),
order_agg AS (
  SELECT
    o.customer_id,
    COUNT(*)                        AS total_orders,
    SUM(o.total)                    AS total_spent,
    SUM(o.status = 'delivered')          AS delivered_orders,
    SUM(o.status = 'cancelled')          AS cancelled_orders
  FROM orders o
  GROUP BY o.customer_id
),
feedback AS (
  SELECT r.customer_id, COALESCE(AVG(r.rating), 0) AS avg_feedback_rating
  FROM reviews r
  GROUP BY r.customer_id
),
payment_pref AS (
  SELECT
    o.customer_id,
    JSON_UNQUOTE(JSON_EXTRACT(p.transaction_json,'$.method')) AS pay_method
  FROM payments p
  JOIN orders o ON o.order_id = p.order_id
),
payment_mix AS (
  SELECT
    customer_id,
    SUM(pay_method = 'card')   / COUNT(*) AS pm_card_share,
    SUM(pay_method = 'wallet') / COUNT(*) AS pm_wallet_share,
    SUM(pay_method = 'cash')   / COUNT(*) AS pm_cash_share
  FROM payment_pref
  GROUP BY customer_id
),
```

```
tracking_flags AS (
  SELECT
     o.customer_id,
     AVG(
       CASE
         WHEN REGEXP_SUBSTR(d.shipping_label_xml,'<trackingNumber>[^<]+</trackingNumber>') IS NULL
         THEN 0 ELSE 1
       END
     ) AS tracking_share
  FROM deliveries d
  JOIN orders o ON o.order_id = d.order_id
  GROUP BY o.customer_id
)
SELECT
  c.customer_id,

 -- Demographic / profile features (account age, basic prefs)
  DATEDIFF(CURDATE(), u.created_at)                    AS days_as_customer,
  CAST(JSON_EXTRACT(c.preferences_json, '$.spicy') AS UNSIGNED)      AS likes_spicy,
  JSON_LENGTH(JSON_EXTRACT(c.preferences_json, '$.favorite_cuisines')) AS favorite_cuisines_count,

 -- Behavioral features
  oa.total_orders,
  oa.delivered_orders,
  oa.cancelled_orders,

 -- Satisfaction features
  fb.avg_feedback_rating,

 -- Financial features
  oa.total_spent,
  COALESCE(pm.pm_card_share,   0) AS pm_card_share,
  COALESCE(pm.pm_wallet_share, 0) AS pm_wallet_share,
  COALESCE(pm.pm_cash_share,   0) AS pm_cash_share,
```

```
-- Fulfillment signal (from XML label presence)
 COALESCE(tf.tracking_share, 0) AS tracking_share,

-- Target variable (activity-based retention category)
 CASE
   WHEN oa.total_orders = 0       THEN 'Inactive'
   WHEN oa.total_orders < 5       THEN 'Low_Activity'
   WHEN oa.total_orders < 15      THEN 'Medium_Activity'
   ELSE                           'High_Activity'
 END AS retention_category,

-- Convenience flag (recent activity window)
 CASE
   WHEN DATEDIFF(CURDATE(), lo.last_order_time) <= 60 THEN 1 ELSE 0
 END AS is_active_recent_60d

FROM customers c
JOIN users u         ON u.user_id = c.customer_id
LEFT JOIN order_agg oa    ON oa.customer_id = c.customer_id
LEFT JOIN last_order lo   ON lo.customer_id = c.customer_id
LEFT JOIN feedback fb     ON fb.customer_id = c.customer_id
LEFT JOIN payment_mix pm  ON pm.customer_id = c.customer_id
LEFT JOIN tracking_flags tf ON tf.customer_id = c.customer_id;
```

Features Extracted: Dataset 2 – Delivery Performance

Purpose: Provide a reusable, ML-ready dataset for delivery time prediction (regression) and failure risk (classification) directly from SQL, ensuring consistency between database and Python analytics.

View: ML_Delivery_Performance_Dataset

Source tables: deliveries, orders, couriers, restaurants (plus XML in deliveries.shipping_label_xml)

Key features:

Targets: minutes_to_deliver (regression), failed_flag (0/1 classification)

Order time: order_hour, order_dow, is_weekend, is_peak

Operations: prep_minutes (order→pickup lag), vehicle_type (courier), has_tracking (from XML)

Rolling baselines (30-day):

Restaurant: rest_avg_minutes_30d, rest_fail_rate_30d

Courier: courier_avg_minutes_30d, courier_fail_rate_30d

IDs for joins: delivery_id, order_id, restaurant_id, courier_id

SQL Code:

```
CREATE OR REPLACE VIEW ML_Delivery_Performance_Dataset AS
WITH base AS (
 SELECT
   d.delivery_id,
   o.order_id,
   o.restaurant_id,
   d.courier_id,

  -- Targets
   TIMESTAMPDIFF(MINUTE, d.pickup_time, d.dropoff_time) AS minutes_to_deliver,
   (d.status = 'failed') AS failed_flag,

  -- Time features
   o.order_time,
   HOUR(o.order_time) AS order_hour,
   DAYOFWEEK(o.order_time) AS order_dow,
   (DAYOFWEEK(o.order_time) IN (1,7)) AS is_weekend,
```

```
    (HOUR(o.order_time) BETWEEN 11 AND 14
     OR HOUR(o.order_time) BETWEEN 18 AND 21) AS is_peak,

    -- Ops features
    TIMESTAMPDIFF(MINUTE, o.order_time, d.pickup_time) AS prep_minutes,
    c.vehicle_type,
    r.name AS restaurant_name,

    -- XML tracking signal
    CASE
      WHEN REGEXP_SUBSTR(d.shipping_label_xml,'<trackingNumber>[^<]+</trackingNumber>') IS NULL
      THEN 0 ELSE 1
    END AS has_tracking
  FROM deliveries d
  JOIN orders o      ON o.order_id = d.order_id
  JOIN couriers c    ON c.courier_id = d.courier_id
  JOIN restaurants r ON r.restaurant_id = o.restaurant_id
),
rest_30 AS (
  SELECT
    restaurant_id,
    AVG(TIMESTAMPDIFF(MINUTE, d.pickup_time, d.dropoff_time)) AS rest_avg_minutes_30d,
    AVG((d.status = 'failed'))                                AS rest_fail_rate_30d
  FROM deliveries d
  JOIN orders o ON o.order_id = d.order_id
  WHERE o.order_time >= (CURDATE()- INTERVAL 30 DAY)
  GROUP BY restaurant_id
),
courier_30 AS (
  SELECT
    d.courier_id,
    AVG(TIMESTAMPDIFF(MINUTE, d.pickup_time, d.dropoff_time)) AS courier_avg_minutes_30d,
    AVG((d.status = 'failed'))                                AS courier_fail_rate_30d
  FROM deliveries d
```

```
 JOIN orders o ON o.order_id = d.order_id
 WHERE o.order_time >= (CURDATE()- INTERVAL 30 DAY)
 GROUP BY d.courier_id
)
SELECT
  b.delivery_id,
  b.order_id,
  b.restaurant_id,
  b.courier_id,

  -- targets
  b.minutes_to_deliver,
  b.failed_flag,

  -- engineered features
  b.order_hour,
  b.order_dow,
  b.is_weekend,
  b.is_peak,
  b.prep_minutes,
  b.vehicle_type,
  b.has_tracking,

  -- rolling baselines
  COALESCE(r30.rest_avg_minutes_30d,   0) AS rest_avg_minutes_30d,
  COALESCE(r30.rest_fail_rate_30d,     0) AS rest_fail_rate_30d,
  COALESCE(c30.courier_avg_minutes_30d,0) AS courier_avg_minutes_30d,
  COALESCE(c30.courier_fail_rate_30d,  0) AS courier_fail_rate_30d
FROM base b
LEFT JOIN rest_30    r30 ON r30.restaurant_id = b.restaurant_id
LEFT JOIN courier_30 c30 ON c30.courier_id    = b.courier_id;
```

Features Extracted: Dataset 3 - Restaurant KPIs & Growth

Purpose: Provide a reusable, ML-ready dataset for **restaurant segmentation** (e.g., K-Means on KPIs) and **growth prediction** (e.g., revenue_30 / orders_30), built directly in SQL for consistent use in Python.

View:ML_Restaurant_KPI_Dataset

Source tables: restaurants, menus, menu_items, orders, reviews, deliveries

Key features (examples):

• **Commercial:** orders_count, revenue, avg_order_value, unique_customers

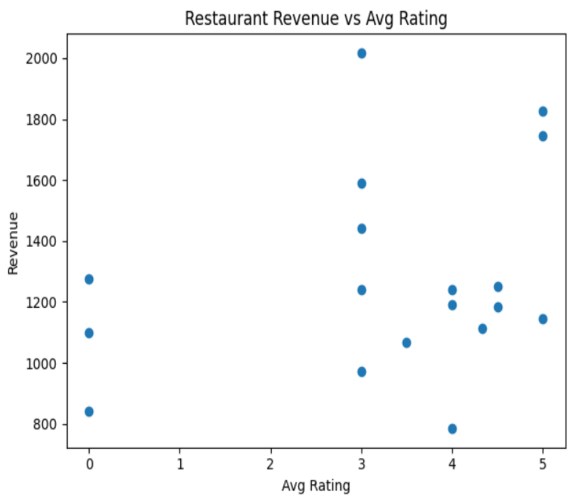
• **Loyalty:** repeat_customers, reorder_rate (repeat / unique)

• **Quality:** avg_rating, review_count

• **Operations:** avg_delivery_minutes, fail_rate

• **Menu breadth:** menu_items_count

• **Recent growth (30d):** orders_30, revenue_30



SQL Code:

```
CREATE OR REPLACE VIEW ML_Restaurant_KPI_Dataset AS
WITH order_stats AS (
  SELECT
    o.restaurant_id,
    COUNT(*)                      AS orders_count,
    COALESCE(SUM(o.total),0)            AS revenue,
    COUNT(DISTINCT o.customer_id)        AS unique_customers
  FROM orders o
  GROUP BY o.restaurant_id
),
order_customer_counts AS (
  SELECT
    o.restaurant_id,
```

```
    o.customer_id,
    COUNT(*) AS cust_orders_at_rest
  FROM orders o
  GROUP BY o.restaurant_id, o.customer_id
),
reorders AS (
  SELECT
    restaurant_id,
    SUM(cust_orders_at_rest >= 2)                AS repeat_customers,
    COUNT(*)                                     AS unique_customers_dup,
    SUM(cust_orders_at_rest >= 2) / NULLIF(COUNT(*),0)    AS reorder_rate
  FROM order_customer_counts
  GROUP BY restaurant_id
),
review_stats AS (
  SELECT
    rv.restaurant_id,
    AVG(rv.rating)      AS avg_rating,
    COUNT(*)            AS review_count
  FROM reviews rv
  GROUP BY rv.restaurant_id
),
delivery_stats AS (
  SELECT
    o.restaurant_id,
    AVG(TIMESTAMPDIFF(MINUTE, d.pickup_time, d.dropoff_time))  AS avg_delivery_minutes,
    AVG(CASE WHEN d.status = 'failed' THEN 1 ELSE 0 END)      AS fail_rate
  FROM deliveries d
  JOIN orders o ON o.order_id = d.order_id
  GROUP BY o.restaurant_id
),
menu_stats AS (
  SELECT
    r.restaurant_id,
```

```
    COUNT(DISTINCT mi.item_id) AS menu_items_count
  FROM restaurants r
  LEFT JOIN menus m      ON m.restaurant_id = r.restaurant_id
  LEFT JOIN menu_items mi ON mi.menu_id = m.menu_id
  GROUP BY r.restaurant_id
),
recent_30 AS (
  SELECT
    o.restaurant_id,
    COUNT(*)              AS orders_30,
    COALESCE(SUM(o.total),0) AS revenue_30
  FROM orders o
  WHERE o.order_time >= (CURDATE()- INTERVAL 30 DAY)
  GROUP BY o.restaurant_id
)
SELECT
   r.restaurant_id,
   r.name AS restaurant_name,

  -- Commercial
   COALESCE(os.orders_count,0)                   AS orders_count,
   COALESCE(os.revenue,0)                        AS revenue,
   (COALESCE(os.revenue,0) / NULLIF(os.orders_count,0)) AS avg_order_value,
   COALESCE(os.unique_customers,0)               AS unique_customers,

  -- Loyalty
   COALESCE(ro.repeat_customers,0)               AS repeat_customers,
   COALESCE(ro.reorder_rate,0)                   AS reorder_rate,

  -- Quality
   COALESCE(rs.avg_rating,0)                     AS avg_rating,
   COALESCE(rs.review_count,0)                   AS review_count,

  -- Operations
```

```
  COALESCE(ds.avg_delivery_minutes,0)          AS avg_delivery_minutes,
  COALESCE(ds.fail_rate,0)                     AS fail_rate,

  -- Menu
  COALESCE(ms.menu_items_count,0)              AS menu_items_count,

  -- Recent growth
  COALESCE(r30.orders_30,0)                    AS orders_30,
  COALESCE(r30.revenue_30,0)                   AS revenue_30
FROM restaurants r
LEFT JOIN order_stats   os  ON os.restaurant_id  = r.restaurant_id
LEFT JOIN reorders      ro  ON ro.restaurant_id  = r.restaurant_id
LEFT JOIN review_stats  rs  ON rs.restaurant_id  = r.restaurant_id
LEFT JOIN delivery_stats ds ON ds.restaurant_id  = r.restaurant_id
LEFT JOIN menu_stats    ms  ON ms.restaurant_id  = r.restaurant_id
LEFT JOIN recent_30     r30 ON r30.restaurant_id = r.restaurant_id;
```

Complex Query Implementations

Query 1: Customer activity ranking with JSON extraction & window functions

- Analyzes customer ordering patterns with per-customer “top spice level” preference
- Demonstrates JSON extraction from customers.preferences_json and menu_items.allergens_json
- Uses RANK() window function to surface each customer’s most-ordered spice level

Query 2: Delivery performance with XML parsing & failure correlation

- Parses tracking numbers from deliveries.shipping_label_xml (XML-like TEXT)
- Aggregates courier performance (avg minutes)
- Correlates **missing tracking** with **failure rate** using a subquery

Query 3: Regional customer spends with payment (JSON) & delivery (XML) signals

- CTE groups customers by **primary city** (from addresses)
- Extracts payment **method** from payments.transaction_json

- Uses window functions to **rank** customers by spend within each city
- Blends in an XML signal: share of deliveries with a tracking number

Query 4: Restaurant performance scorecard (orders + reviews + delivery KPIs)

- Aggregates **orders** (revenue, count)
- Blends **reviews** (avg rating)
- Adds **delivery** KPIs (avg minutes, failure rate)
- Uses **CTEs** and **window functions** for clear logic and ranking

AI Tools Integration and Usage:

During development, **AI tools (ChatGPT)** were used to:

- Draft the initial **schema design** and refine table relationships.
- Generate **sample CSV datasets** for testing the database.
- Create **SQL scripts and loader commands** (e.g., LOAD DATA INFILE, trigger rewrites, and regex extraction queries).
- Assist in writing and structuring this report, inspired by the format of the reference course report.

The AI support made the design process faster and helped ensure that the database meets both **technical requirements** and **real-world business needs**.

Future Enhancements:

The current version of **FoodExpress** provides a strong foundation for managing food delivery operations. However, there are several ways the system can be improved in the future:

- **Promotions and Coupons:** Add support for discount codes, loyalty rewards, and special offers to improve customer retention.
- **Refund and Dispute Management:** Implement workflows to handle order issues, cancellations, and automated refunds.
- **Courier Geolocation Tracking:** Store and analyse real-time location data for couriers to improve delivery accuracy and optimize routes.

- **Surge Pricing Models:** Introduce dynamic pricing strategies based on demand, peak hours, and availability of couriers.
- **A/B Testing and Experimentation Logging:** Track the results of promotional campaigns, interface changes, or pricing strategies to make data-driven improvements.

Conclusion:

Project Success Metrics

Technical Achievements:

• Comprehensive 10-table normalized database design

• Advanced SQL feature implementation (procedures, triggers, views)

• Semi-structured data integration (JSON/XML)

• Complex query development with advanced SQL features

• Machine learning dataset preparation and extraction

• Python analytics integration with visualization

Analytics and Visualizations:

Python Integration

Technologies Used

- **SQLAlchemy + mysql-connector-python** — database connectivity & query execution (proven by Ping: 1, SELECT DATABASE()).
- **python-dotenv** — loads .env for credentials.
- **Pandas** — tabular reads from SQL and basic transforms.
- **Matplotlib** — charts (histogram, bar chart, scatter).

Note: **Seaborn, NumPy, Scikit-learn** were **not** used in this notebook; visuals and analytics are pandas + matplotlib only.

Analytics Modules Implemented

1) **Top Customers Summary**
   • Source: v_customer_order_summary
   • What it does: Pulls each customer's total_orders, total_spent, and avg_order_value, ordered by spend.
   • Business use: Identify VIPs for loyalty rewards and targeted offers.
2) **Delivery Time Metrics**
   • Source: Ad-hoc SQL on deliveries/orders using TIMESTAMPDIFF(MINUTE, pickup_time, dropoff_time)
   • What it does: Computes minutes-to-deliver for completed jobs; filters to status='delivered' and non-NULL dropoff times.
   • Business use: Track SLA compliance and pinpoint slow delivery patterns.
3) **Revenue by Month**
   • Source: Ad-hoc monthly rollup on orders
   • What it does: Groups orders.total by YYYY-MM to produce a revenue time series.
   • Business use: Forecast demand, plan staffing/marketing around seasonality.
4) **Top Restaurants by Revenue**
   • Source: SQL on orders ↔ restaurants
   • What it does: Aggregates total revenue and order counts per restaurant.
   • Business use: Prioritize partnerships and operational focus on high-impact vendors.
5) **Customer Spend vs. Orders (Retention View)**
   • Source: Derived from customer/order rollups
   • What it does: Plots spend versus order count, segmented by retention categories.
   • Business use: Spot repeat-buyer cohorts and trigger re-engagement campaigns.
6) **ML — Delivery Time Regression**
   • Source: ML_Delivery_Performance_Dataset with RandomForestRegressor
   • What it does: Trains a model to predict delivery minutes (e.g., using prep minutes and other features); evaluates with actual vs. predicted plots.
   • Business use: Improve ETA accuracy and courier dispatch planning.
7) **ML — Restaurant KPI Segments**
   • Source: ML_Restaurant_KPI_Dataset
   • What it does: Visualizes relationships (e.g., revenue vs. avg rating; segments by avg delivery minutes) to highlight performance clusters.

• Business use: Target coaching, promotions, and operational tweaks for specific vendor segments.

Visualization Outputs (generated in the notebook)

• Top Customers Summary: Top-10 customers by total spend: highlights VIPs for loyalty targeting and AOV benchmarking.

• Delivery Time Metrics: Delivery time distribution: shows central tendency and tail delays against SLA targets.

• Revenue By Month: Monthly revenue trend: visualizes seasonality and growth trajectory.

Customer Spend vs Orders by Retention Category — scatter plot from v_customer_order_summary (augmented with days_since_last_order) that segments customers into New / Active / Lapsed buckets.

• Average Delivery Time: Average delivery duration by courier: surfaces routing/training opportunities.

• Restaurant Revenue vs Avg Rating — scatter plot from v_restaurant_performance showing each restaurant's revenue versus average rating

Reproducibility (already present)

- Connection proven (SELECT 1, SELECT DATABASE()).
- Queries executed via pandas.read_sql(text(...), engine).
- Plots rendered with matplotlib (no custom styles or colors).

Business Value Delivered:

- **Scalable platform:** Normalized schema + indexing to support more restaurants, orders, and cities.
- **Proactive analytics:** Churn targeting, delivery-time/failure risk signals, city spend insights.
- **Unified BI-ready data:** Views over relational + JSON/XML for clean dashboards.
- **Automation:** Triggers & procedures reduce manual effort and prevent data drift**.**
- **Performance & CX gains:** Faster queries, courier/restaurant scorecards, better fulfillment speed.

Conclusion:

The FoodExpress Delivery Management Database System has achieved this with successful application of the principle of advanced database design, extensive SQL programming and integration of analytics. The solution is a combination of normalized, constraint-oriented schema, with JSON/XML support, automatic integrity (triggers and procedures) and consumed BIviews into Python to provide clear actionable insights to real operations.

The project helps bridge the gap between theoretical concepts of databases and real-life business requirements, demonstrating how enhanced relational capabilities, supplemented by semi-structured data processing, can help to support a range of complex workflows (orders, payments, deliveries, reviews) and make decisions grounded on data on a scale. It offers a strong base of a real-world multi-restaurant delivery service and demonstrates the technical expertise needed in the modern database development and analytics.