Course Coordinator: Andrew Lang

Tutor: Shena Machin

Torin Flanagan: 1169130

Due Date: 17/10/2025

Word Count: 1,273

Task 3: Document and SQL Database Project

ICT 320 – Database Programming

Contents

[Introduction 2](#_Toc211617661)

[Database Design 2](#_Toc211617662)

[NoSQL Schema Design 2](#_Toc211617663)

[NoSQL Rationale 5](#_Toc211617664)

[SQL Schema Design 5](#_Toc211617665)

[SQL Rationale 6](#_Toc211617666)

[System Architecture 6](#_Toc211617667)

[Discussion 7](#_Toc211617668)

[Testing 7](#_Toc211617669)

[Sample Test Cases and Associated Outcomes 7](#_Toc211617670)

[Screenshots 8](#_Toc211617671)

[References 13](#_Toc211617672)

**I acknowledge the use of ChatGPT in this assessment for generating database ideas, assisting initial ideas, and editing my code application.**

# Introduction

Joe’s Pizzeria, a franchise with over 25 independently operated stores across Australia, faces challenges from inconsistent order management, incomplete data, and limited insights into customer trends and store performance. To address this, the company is pursuing a digital transformation strategy by developing a cloud-based application for order management, driver allocation, and daily reporting. The proof-of-concept system for the Little Joe’s store integrates NoSQL and SQL databases on Microsoft Azure, enabling streamlined order processing, docket creation, driver management, and automated daily summaries. This improves data consistency, provides reliable performance insights, and lays the foundation for a company-wide rollout.

# Database Design

A NoSQL (Not Only SQL) database is a design approach that stores and queries data outside the traditional tabular structures of relational databases. Instead of tables, NoSQL databases store data in a single structure, offering rapid scalability and efficient management of large, semi-structured, or unstructured datasets (IBM, 2022). In this project, a document-based schema was used, where each record (document) stores key-value pairs in JSON format. Documents can include various data types (e.g. strings, integers, lists, etc) and are grouped into collections (MongoDB, 2024).

In contrast, a SQL database is relational and uses structured query language to manage data stored in tables. Data is organised in rows and columns, with each table representing a structured dataset for consistent storage, aggregation, and reporting (Microsoft Azure, 2025).

## NoSQL Schema Design

**Customers Collection:**

{

“\_id”: “ObjectId”,

“first\_name”: “string”,

“last\_name”: “string”,

“phone”: “integer”,

“email”: “string”,

“address”: {

“street”: “string”,

“suburb”: “string”,

“postcode”: “integer”

}

}

**Orders Collection:**

{

“\_id”: “ObjectId”,

“customer\_id”: “ObjectId”,

“order\_date”: “datetime”,

“order\_date\_only”: “string”,

“pizzas”: [

{

“pizza\_name”: “string”,

“size”: “string”,

“quantity”: “integer”,

“price”: “double”

}

],

“total\_price”: “double”,

“driver\_id”: “ObjectId”,

“status”: “string”

}

**Drivers Collection:**

{

“\_id”: “ObjectId”,

“name”: “string”,

“delivery\_suburbs”: [“string”],

“commission\_rate”: “double”,

“is\_active”: “boolean”

}

**Dockets Collection:**

{

“\_id”: “ObjectId”,

“docket\_type”: “string”,

“order\_id”: “ObjectId”,

“order\_date”: “string”,

“status”: “string”,

“customer\_info”: {

“first\_name”: “string”,

“last\_name”: “string”,

“email”: “string”,

“phone”: “integer”,

“address”: {

“street”: “string”,

“suburb”: “string”,

“postcode”: “integer”

}

},

“driver\_info”: {

“name”: “string”,

“delivery\_suburbs”: [“string”],

“commission\_rate”: “double”,

“commission\_amount”: “double”

},

“order\_summary”: {

“total\_price”: “double”,

“item\_count”: “integer”

},

“ordered\_items”: [

{

“pizza\_name”: “string”,

“size”: “string”,

“quantity”: “integer”,

“price”: “double”

}

],

“format”: string

}

**Document References:**

orders.customer\_id >> customers.\_id

orders.driver\_id >> drivers.\_id

dockets.order\_id >> orders\_id

## NoSQL Rationale

The Joe’s Pizzeria NoSQL schema was designed for flexibility, scalability, and efficient management of orders, drivers, and dockets. Using a document oriented model allows for diverse data structures and frequent updates without rigid schema constraints. Core collections (customers, orders, drivers, and dockets) are stored separately to maintain logical separation and simplify queries. References (e.g. customer\_id and driver\_id) manage relationships between documents, while embedded subdocuments, such as addresses and ordered items, provide fast access to essential details.

The orders collection serves as the central link between customers and drivers, while the dockets collection embeds full order, customer, and driver information to preserve immutable records for printing and auditing. This hybrid approach balances embedding and referencing, ensuring data integrity, performance, and support for real-time operations and automated daily reporting.

## SQL Schema Design

CREATE TABLE Pizzeria\_Daily\_Summaries (

summary\_id INT IDENTITY(1,1) PRIMARY KEY,

summary\_date DATE NOT NULL UNIQUE,

total\_orders INT NOT NULL,

total\_sales\_revenue DECIMAL(10,2) NOT NULL,

most\_popular\_pizza NVARCHAR(100) NOT NULL,

total\_driver\_commission DECIMAL(10,2) NOT NULL,

creation\_date DATETIME NOT NULL GETDATE()

);

## SQL Rationale

The Joe’s Pizzeria SQL schema provides a structured and reliable repository for daily system summaries. Key operational metrics, such as total sales revenue, are recorded for each trading day in the Pizzeria\_Daily\_Summaries table. With there being a unique constraint on summary\_date, duplicate entries are prevented, further ensuring data integrity.

The relational structure supports aggregation, reporting, and integration with analytical tools. By using numeric and date data types, these help enable precise financial calculations, while the creation\_date column serves as a timestamp for traceability. Overall, the design complements the NoSQL database by enabling consistent, accurate, and easy-to-query historical performance analysis for each store.

# System Architecture

The Joe’s Pizzeria system architecture integrates a Python application with NoSQL and SQL databases hosted on Microsoft Azure, enabling seamless data flow between operational and analytical components. The Python application acts as the central control layer by managing order processing, driver allocation, docket generation (including PDFs), and daily summary reporting (including PDFs).

When a new order is placed, the application captures customer, order, and driver information and stores it in the NoSQL database (Azure Cosmos DB). Its flexible schema and high-speed retrieval make it ideal for real-time operations. Orders then reference existing customer and driver records to maintain efficient linkages without redundancy. Cooking and delivery dockets are automatically generated as PDFs and saved in the NoSQL database for operational use and printing.

At the end of each day, the application aggregates order data using MongoDB query pipelines and stores the resulting summaries (e.g. total orders and most popular pizza) in the SQL database (Azure SQL Database) for structured reporting. Daily summary PDFs are also automatically generated and accessible via the application.

This two-tier architecture leverages the strengths of each database, where NoSQL manages flexible, document-based data for daily operations, while SQL provides structured storage for long-term analysis. The Python application ensures consistent communication between both databases and automates document generation, improving efficiency and reducing manual workload.

# Discussion

One core challenge was integrating the NoSQL and SQL databases into the same Python application. By trying to establish reliable and secure connections to Microsoft Azure, this required correct configurations of connection strings, drivers, and firewall permissions (allowing IP ranges from 0.0.0.0 to 255.255.255.255) for the SQL database. Another challenge involved maintaining data consistency between NoSQL order data and SQL daily summaries, which was addressed by performing aggregation directly in NoSQL using query pipelines rather than Python iterations, improving performance and accuracy.

The NoSQL schema uses a hybrid approach, embedding subdocuments for contextual data and referencing collections for relationships, to further balance query efficiency and data integrity. The SQL schema uses a single relational table with a unique date constraint to simplify reporting and enforce data consistency for daily summaries.

A key technical decision was implementing automatic PDF generation for order dockets (combining cooking and delivery) and daily summaries. While this enhanced functionality and usability, it introduced dependencies on external libraries and formatting complexities.

Other limitations included restricted scalability due to Azure Free Tier constraints and single-store testing, leaving multi-store integration and high-volume performance as future considerations. Despite these challenges, the system meets all core requirements and one advanced feature, providing a scalable foundation for company-wide deployment.

# Testing

The system was tested using a combination of Python unit tests and manual verification. Unit tests covered all core functionalities, including customer, driver, and order management, docket creation, SQL daily summaries, and PDF generation. The NoSQL and SQL databases were mocked to isolate application logic, ensuring accurate and reproducible test results.

## Sample Test Cases and Associated Outcomes

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Description/Expected Outcome** | **Result** |
| Add Customer | Verified inserting new customers and retrieving exiting customer IDs. | PASS |
| Add Driver | Verified inserting new drivers and handling existing driver records. | PASS |
| Add Order and Generate Dockets | Ensures adding an order creates cooking and delivery dockets and generates PDF. | PASS |
| Generate Daily Summaries | Confirms correct aggregation of NoSQL order data and inserting/updating in SQL summaries. | PASS |
| PDF Generations | Verifies automatic PDF creation for orders and daily summaries. | PASS |
| View Functions | Tests displaying drivers, orders, and SQL summaries. | PASS |
| Database Cleaning | Verifies that all NoSQL collections can be cleared without errors. | PASS |

## Screenshots

**A screenshot of a computer

AI-generated content may be incorrect.Unit test terminal output:**

**A screenshot of a computer

AI-generated content may be incorrect.Sample customers document in NoSQL:**

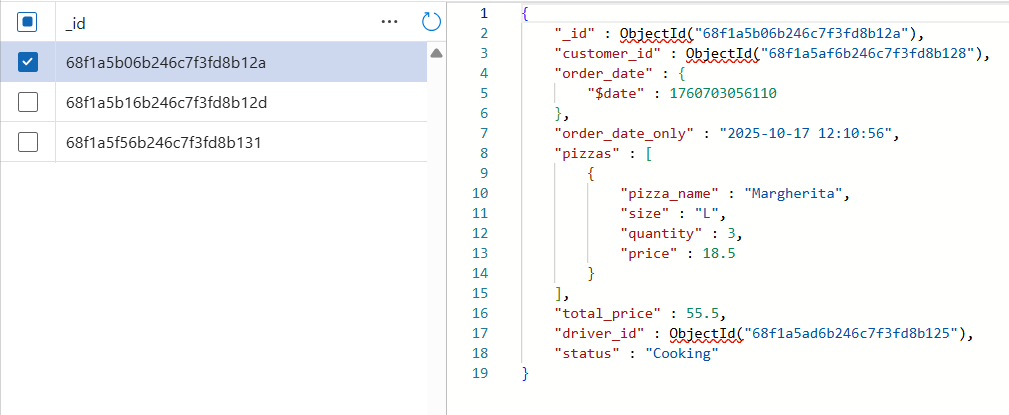
**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.Sample dockets document in NoSQL:**

**A screenshot of a computer

AI-generated content may be incorrect.Sample drivers document in NoSQL:**

**Sample orders document in NoSQL:**

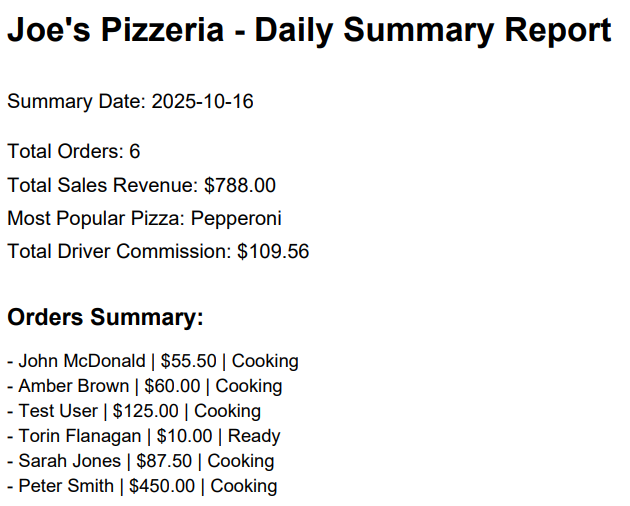
A screenshot of a computer

AI-generated content may be incorrect.**NoSQL collection:**

A screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**Generated PDF cooking and delivery dockets:**

**Generated PDF daily summary report:**

A screenshot of a computer

AI-generated content may be incorrect.**Sample daily summaries in SQL:**

A screenshot of a computer

AI-generated content may be incorrect.**SQL table:**

# References

* IBM. (2022, December 12). NoSQL Databases. Ibm.com. https://www.ibm.com/think/topics/nosql-databases
* Microsoft Azure. (2025). What is a SQL Database? | Microsoft Azure. Azure.microsoft.com. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-sql-database
* MongoDB. (2024). How to design schema for NoSQL data models. MongoDB. https://www.mongodb.com/resources/basics/databases/nosql-explained/data-modeling