UNIVERSITETI I EVROPËS JUGLINDORE
УНИВЕРЗИТЕТ НА ЈУГОИСТОЧНА ЕВРОПА
SOUTH EAST EUROPEAN UNIVERSITY

Faculty of Contemporary Sciences and Technologies

NoSQL Databases

(Semester 6, 2024/2025)

Project Title: Optical Store Management System

Students:

Merita Aliu

Gjilizar Zhuta

Mentor:

Prof. Benjamin Besimi

June 2025, Tetovo

Table of contents

1. Introduction

This report outlines the full process of migrating an Optical Store Management System database from SQL Server (relational model) to MongoDB (document-oriented NoSQL model). The system includes core entities like clients, employees (medical workers, sales staff, managers), products, invoices, and eye examination records. The migration was undertaken to improve performance, enable schema flexibility, and simplify data access patterns. This document presents the rationale for selecting MongoDB, the transformation of the relational schema to a document model, migration strategy using Python, challenges encountered, and supporting artifacts such as diagrams, scripts, and JSON structures.

2. Relational Database Design and Data Modeling

2.1 Business plan

Our database's business plan will contain information about the Optical Store's activities, which include clients buying/looking at products in the store, employees where we have medical workers, retail workers and managers, who maintain the enterprise, and invoices which are generated when a client makes a purchase.

Constructing this database will help the staff of the optical store ensure a consistent eye care settlement to clients. Another feature of this database is the easy access to add and modify information about clients' and improving the general management of the shop. With this database we will be able to smoothly document the details of the prescriptions, eye examination results and customer preferences.

The business plan will have some crucial, key objectives including:

- Optimizing client satisfaction through a well-structured database using different applications and systematized transaction processing.
- To strengthen the supplier relationships in order to have a diverse, rich range of high-quality eyewear products.
- Elevate the overall efficiency of the optical store's operational process, through high-level data integrity and data security which DBMS offers.
- Spur financial growth by attracting new customers as well as leveraging existing sales opportunities.

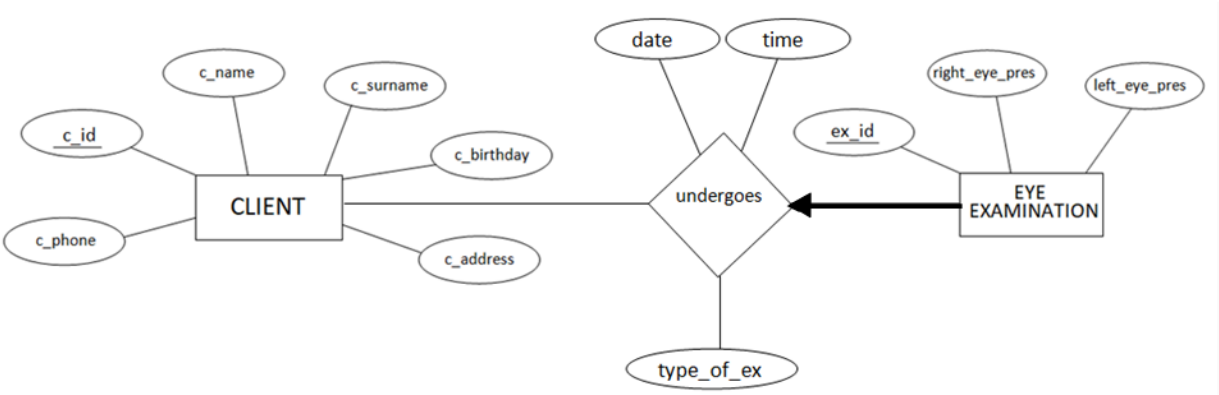
The detailed business plan is described as follows:

1. Each client who enters the shop is assisted by at most one sales worker.

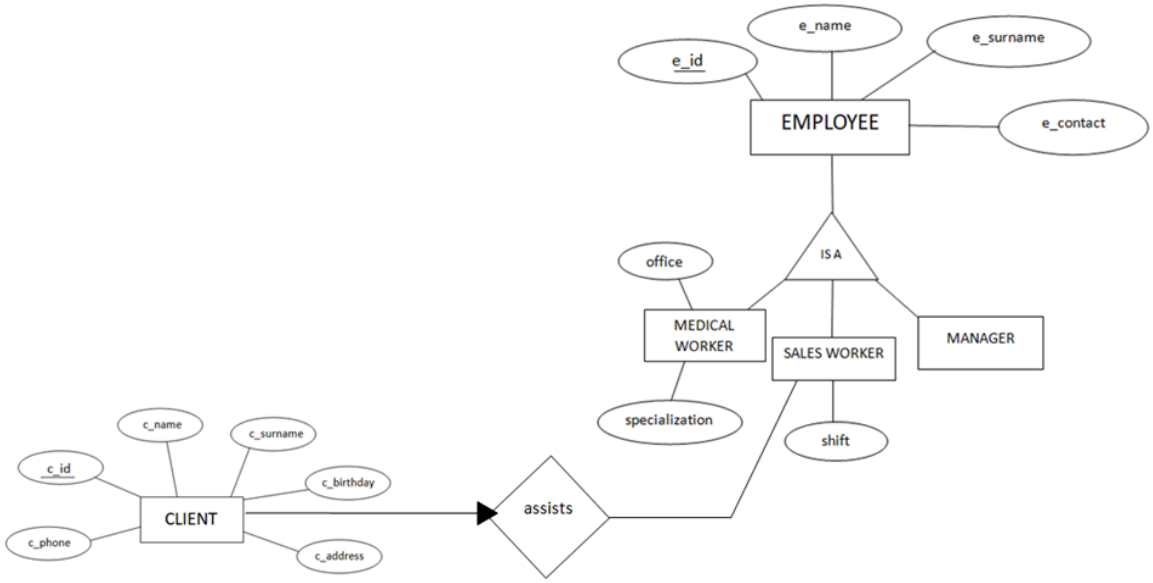
2. Then a client can undergo more than one eye examination and one eye examination must be taken by at most one client. The data about the eye examination like date, time, type of examination will be saved when a client undergoes an eye examination.
3. After the examination, a client can then proceed to make any amount of purchases, and a purchase can be made by more than one client. Each purchase is associated with exactly one invoice, and one purchase must be included in a single invoice, and each purchase is identified by a unique invoice. Therefore, a purchase entity cannot exist without its corresponding invoice.
4. A purchase can include more than one product, and a product can be included in more than one purchase. When the purchase is made by the client, one can client receive multiple invoices, and an invoice must be received by at most one client.
5. An employee at the store can either be a sales worker, medical worker, or a manager.
6. When a purchase is done, a sales worker prepares more than one invoice, and an invoice must be prepared by at most one sales worker.
7. A purchase must be handled by at most one sales worker, but a sales worker can handle more than one purchase.
8. A medical worker can conduct more than one eye examination, and more than one eye examination must be conducted by at most one medical worker.
9. A manager can manage more than one supplier, but one supplier is managed by at most one manager, where we also have the date from and date until that manager manages a certain supplier.
10. A supplier can supply more than one product and a product can be supplied by more than one supplier.

2.2 Diagrams, Constraints and Cardinality

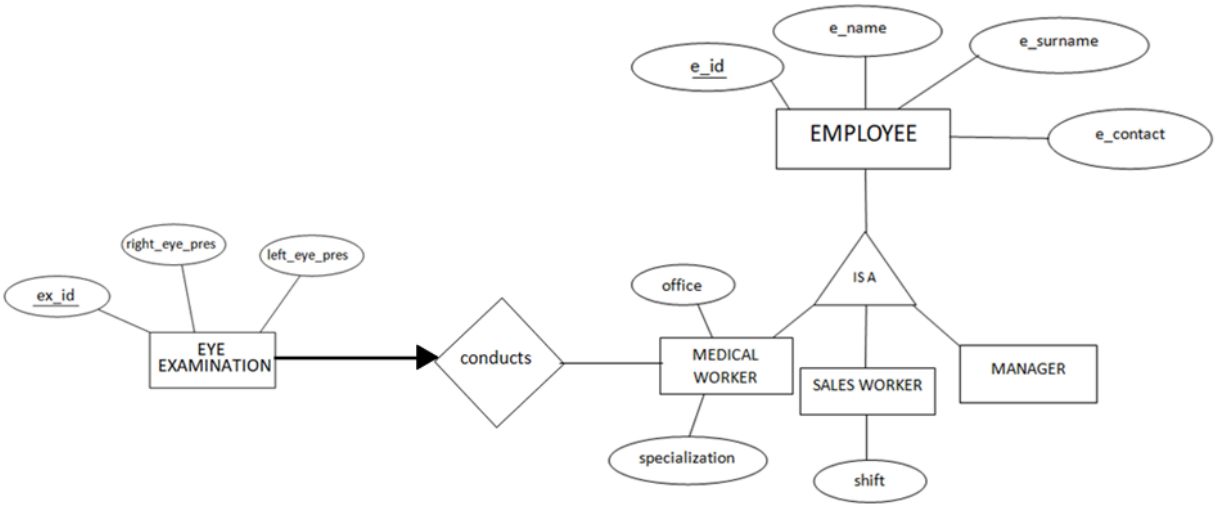
Based on the given problem description and the business requirements when a Client undergoes an Eye Examination at the optical store, the entity Eye examination has a key constraint and a participation constraint to the relationship because an eye examination belongs to a client only and the eye examination exists only if the client undergoes the eye examination. While the Client entity does not have any constraints because a client can attend multiple eye examinations.



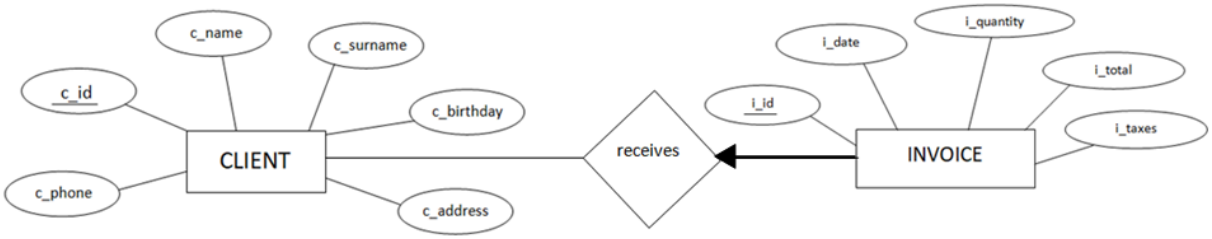
Based on the given problem description where a client goes into an optical store the client is assisted by a sales worker in which in the relationship between the Client entity and the Sales Worker entity, the Sales Worker doesn't have a key constraint not a participation constraint, but the Client entity has a key constraint since a sales worker can assist more than one client. Overall, the relationship formed is of the type one-to-many.



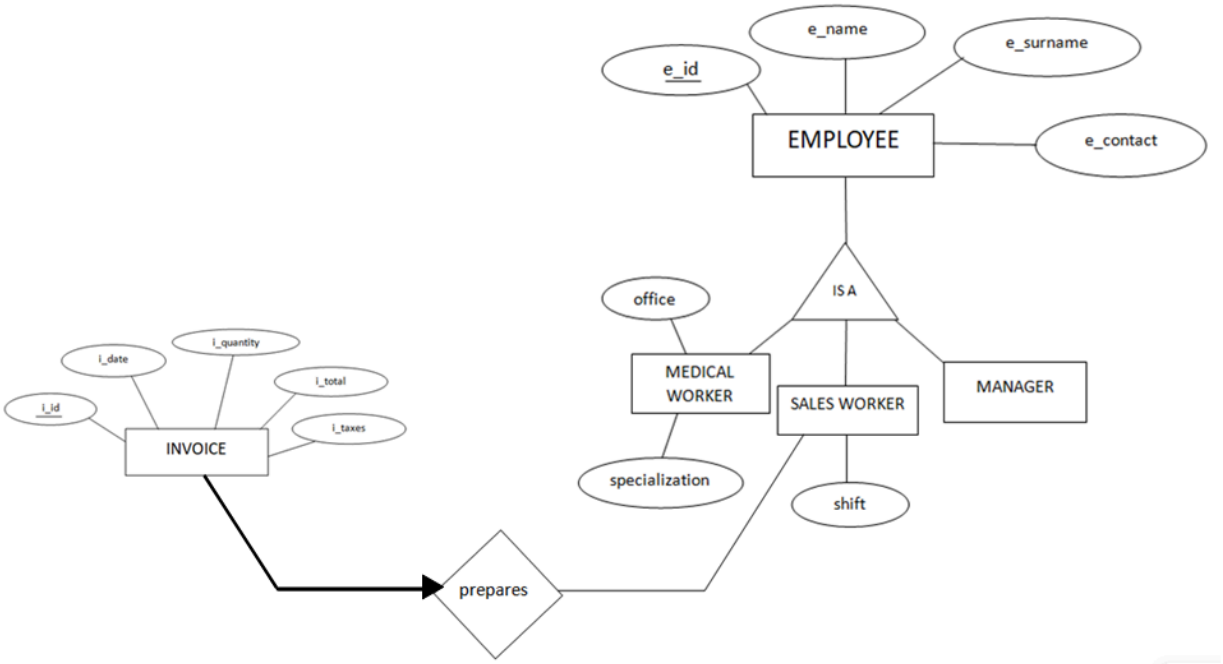
In addition, when a client undergoes an eye examination, it is conducted by a medical worker, in which in the relationship between the Eye Examination entity and Medical Worker entity, Eye Examination has both key constraint and participation constraint, where a Eye Examination can be conducted by at least one medical worker, and its existence depends on the Medical Worker entity. But the Medical Worker entity doesn't have a key nor a participation constraint.



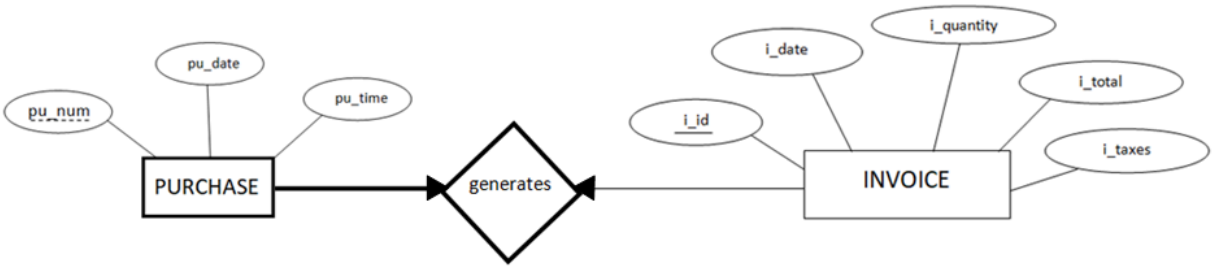
Furthermore, the relationship between Client and Invoice named “receives”, based on the request one client can receive many invoices, and an invoice can be received by one client where the entity Invoice has both key and participation constraints, because the invoice will exist only if it is paid from a client. The entity Client doesn’t have key or participation constraints. Making this relationship one-to-many.



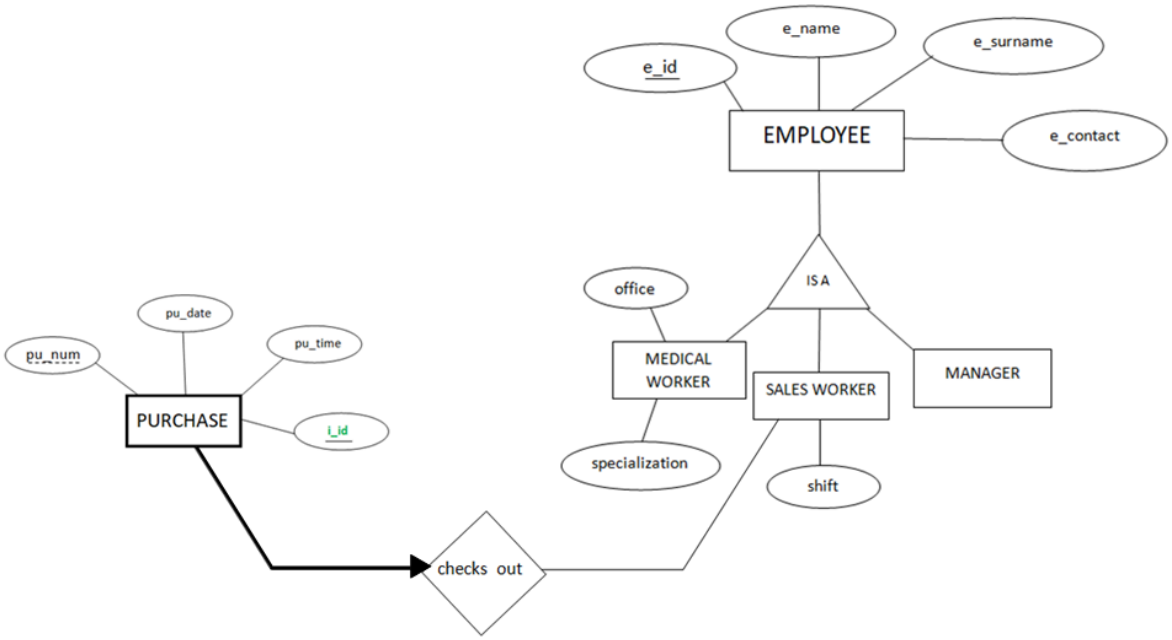
Next we have the relationship “prepares” between Sales Worker and Invoice, an invoice is prepared by one Sales Worker of the optical store, in which case the Invoice entity has a key constraint and a participation constraint, because the invoice will exist only if it is prepared by a Sales Worker, resulting in a one-to-many relationship between Invoice entity and Sales Worker entity. While the Sales Worker entity does not have a key nor participation constraint.



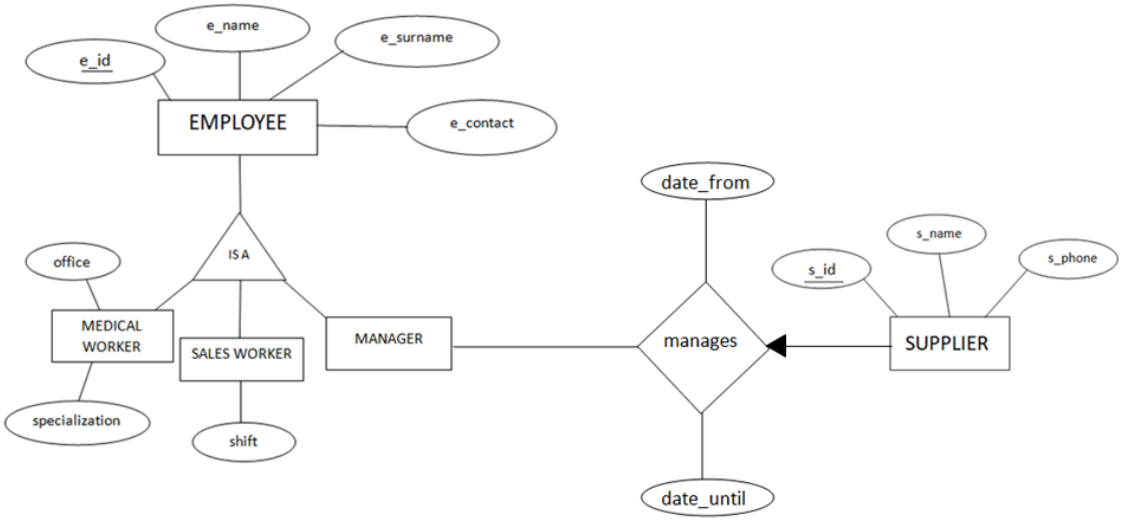
Furthermore, based on the business plan, the Purchase generates an Invoice, in which the Purchase entity has a key constraint as well as a participation constraint, since a Purchase can generate at most one Invoice, and its existence depends on the Invoice because its a weak entity. Invoice has a key constraint but not a participation constraint, in which an Invoice belongs at most to one Purchase, making this relationship overall of the type one-to-one.



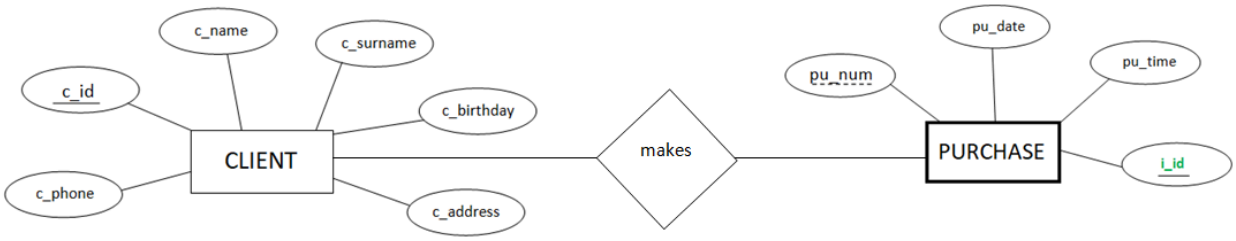
Based on the next step pointed out in the problem description, the Sales Worker checks out the Purchase, in which, the Purchase entity has a key constraint and a participation constraint in the relationship with the Sales Worker, since a purchase must be handled by at most one sales worker and the purchase exists if the Sales Worker checks it out. The Sales Worker Entity does not have a key constraint in this case, since a sales worker can handle more purchases done by multiple clients. Making this relationship 1:M.



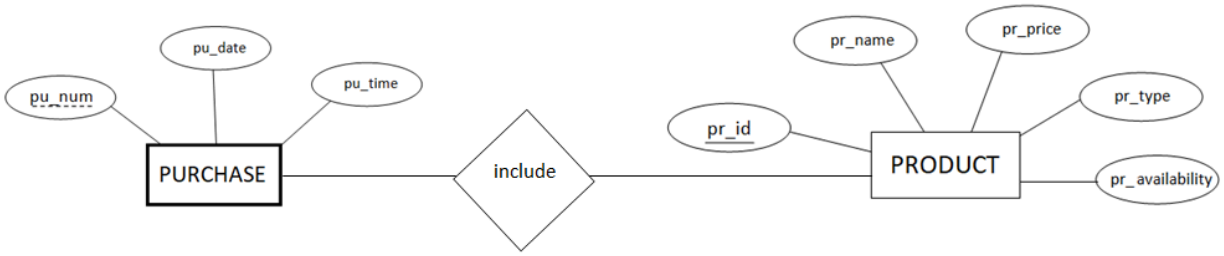
The optical store database also has a Manager and Supplier, in which in the relationship between Manager entity and Supplier entity called “manages”, Supplier has a key constraint but not participation constraint, since it can be managed by at most one Manager. The Manager entity has neither participation constraint, nor key constraint, making this relationship overall of the type one-to-many as well.



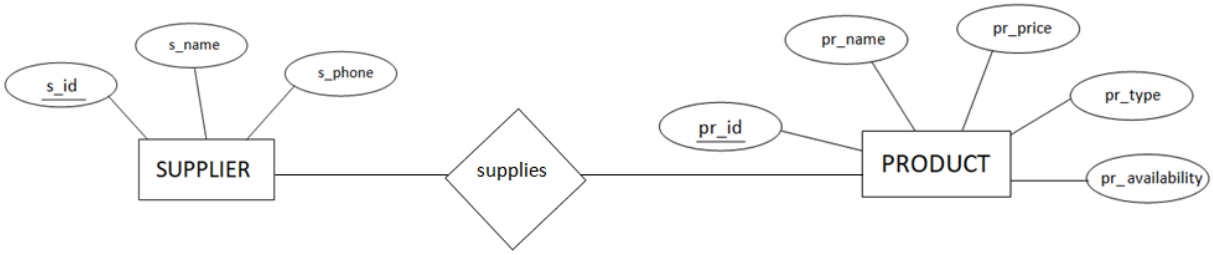
From the business rule we can also notice the relationship “makes” between Client and Purchase where a Client can make many purchases and a purchase can be made by many clients, meaning that neither Client nor Purchase have key or participation constraints.



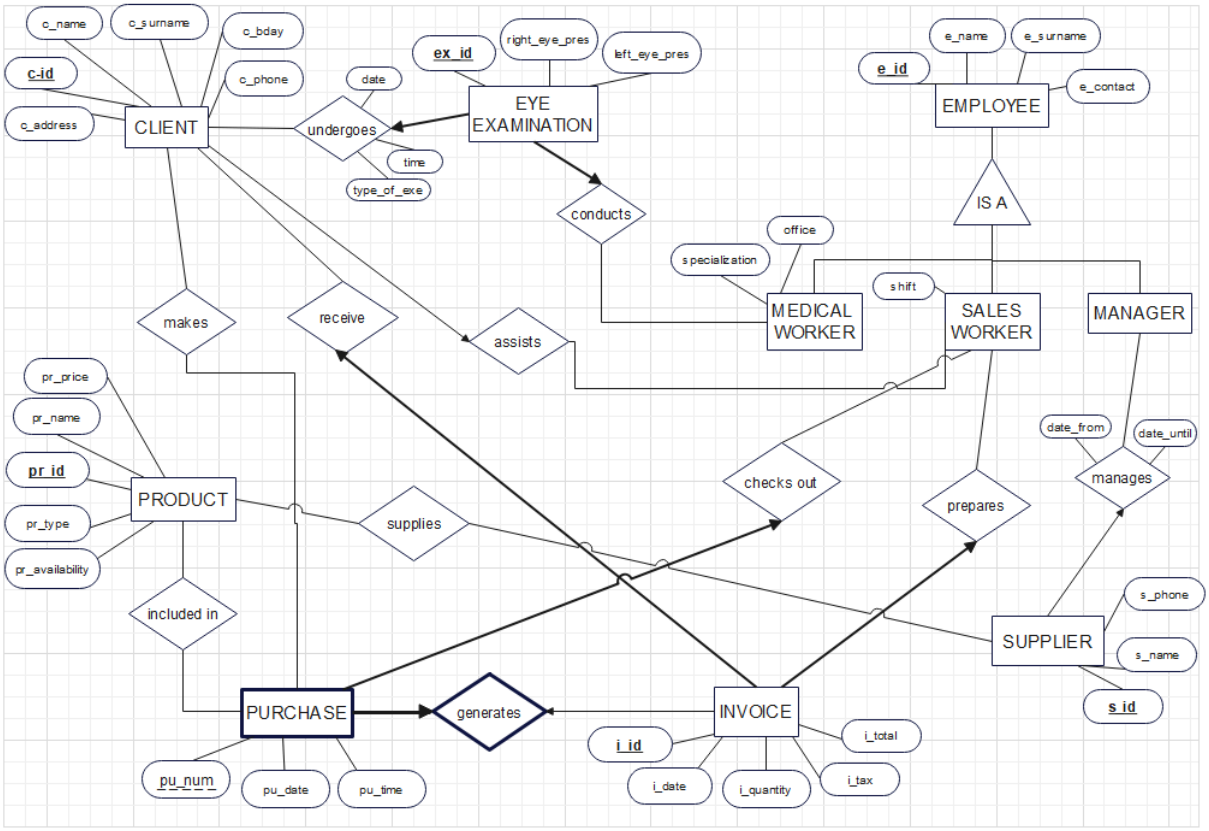
The relationship named “included in” between Product and Purchase is a many-to-many relationship because one purchase can include multiple products, and one product can be included in multiple products. And neither of the entities have participation constraints.



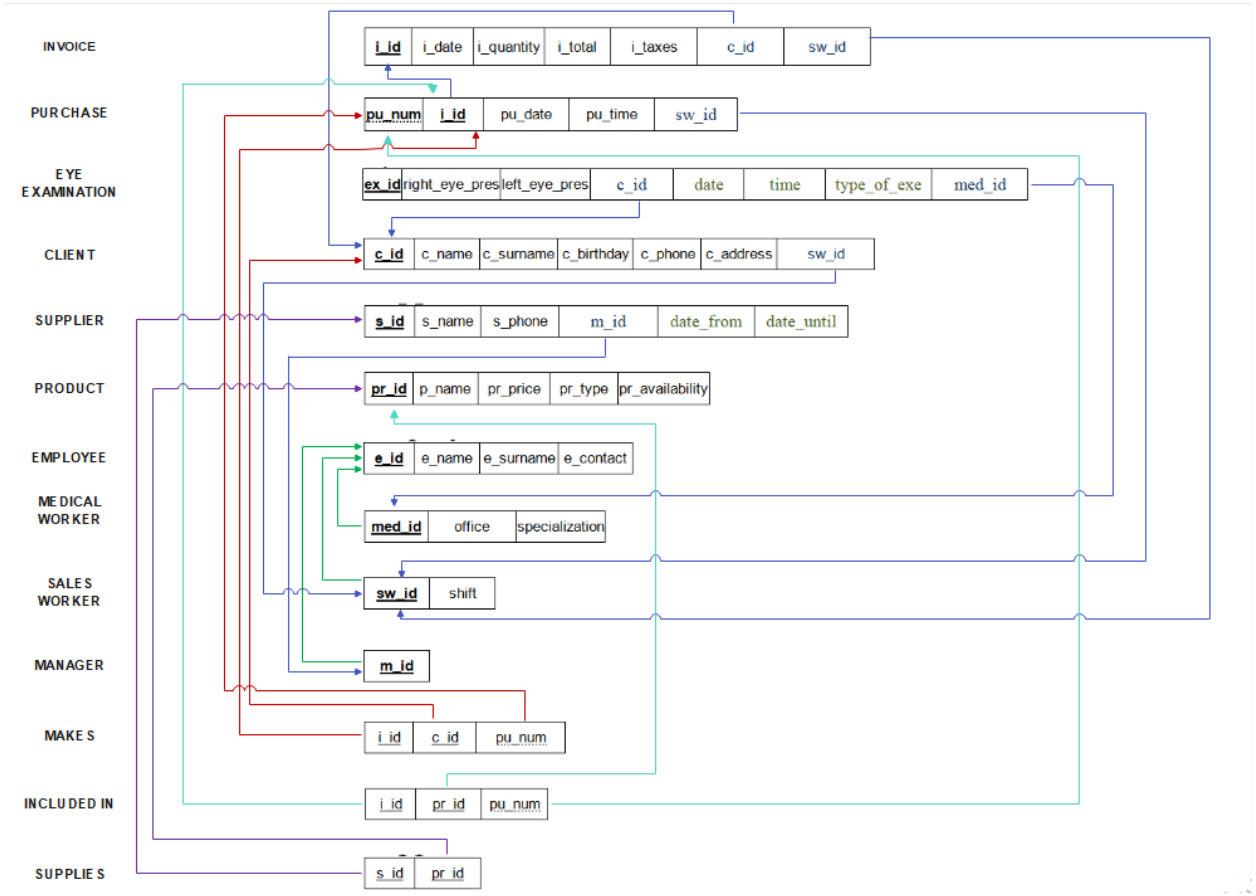
Lastly, we have the relationship named “supplies” between Supplier and Product, where a supplier can supply multiple products, and a product can be supplied by multiple suppliers, so the relationship is many-to-many without any participation constraint.



2.3 E-R Diagram



2.4 Relational Design



3. Data Population

Screenshots showing the populated data in the relational database can be found in the docs/Query Results.pdf

4. Choice of NoSQL Database

4.1 MongoDB supports:

Sharding for horizontal scalability

Replication for high availability

Indexes on any field to enhance query performance

4.2 Comparison with Alternatives

To justify the choice of MongoDB, we compared it with three other popular NoSQL databases: Redis, Cassandra, and Neo4j. The comparison was based on data model flexibility, use case fit, query capabilities, and overall suitability for an Optical Store Management System.

- **MongoDB** uses a document-oriented model (BSON/JSON), which is particularly well-suited for handling nested and hierarchical data. In our project, clients may have multiple embedded eye examinations, and invoices contain embedded purchase items — both of which fit naturally into MongoDB's flexible schema. It allows us to avoid complex JOIN operations and access full data with a single query. MongoDB also supports powerful query features, indexing on any field, aggregation pipelines, and dynamic schema evolution, making it an ideal solution for a business domain with moderately complex but structured relationships.
- **Redis** is a key-value store optimized for caching, pub/sub, and real-time analytics. While it's extremely fast, it does not support complex queries or nested data structures, and lacks a native way to represent relationships or perform multi-field searches. Therefore, it's unsuitable for our case, where structured records and cross-entity relationships are essential.
- **Cassandra** employs a wide-column data model, excelling in high-throughput, write-intensive use cases such as telemetry or logging. However, its schema is rigid and does not accommodate nested documents or hierarchical data easily. Additionally, Cassandra's query language (CQL) offers limited JOIN capabilities and weaker aggregation support compared to MongoDB. For a mid-sized business application like ours, Cassandra would impose unnecessary constraints and complexity without delivering proportional benefits.
- **Neo4j** is a graph database designed for applications with highly interconnected data, such as social networks or fraud detection. While powerful for relationship traversal and path queries using Cypher, our Optical Store data model consists primarily of hierarchical and transactional data (e.g., clients → exams, invoices → items), not deep graphs. Neo4j's graph model would be overkill for our needs and would complicate data storage and querying compared to MongoDB's more intuitive document model.

5. NoSQL Database Modeling

The Optical Store Management System was structured into multiple MongoDB collections, each reflecting core entities from the original SQL Server database. MongoDB's flexible document model enabled natural representation of nested and reference-based relationships.

5.1 Collections and Structure:

- clients: Stores client information and embeds eye examinations as an array.
- employees: A unified collection containing managers, salesworkers, and medical workers, each distinguished by a role field.
- products: References supplierId, linking to the suppliers collection.
- suppliers: Contains supplier data and a managerId referencing an employee.
- invoices: Stores invoice data, embedding line items (product purchases).
- purchases, makes, supplies, included_in: Modeled with referencing relationships or flattened where appropriate.

5.2 Mapping from SQL to MongoDB:

Relational Concept	**MongoDB Representation**
Tables	Collections (e.g., employee→ employees)
Rows	Documents
Foreign Keys	Embedded documents or referenced IDs
JOINs	Avoided using embedding or $lookup if needed

This hybrid strategy of embedding where tight coupling exists (e.g., examinations within clients), and referencing where data is reused (e.g., suppliers, managers), balances performance with normalization.

5.3 Example Mapping

Relational SQL:

SELECT * FROM eye_examination WHERE c_id = 'CL0001';

MongoDB Equivalent:

db.eye_examinations.find({ client_id: "CL0001" })

6. Data Migration Process

6.1 Tools Used:

- Source: SQL Server 2019
- Target: MongoDB (local instance)
- Language: Python
- Libraries: pyodbc, pymongo
- Version Control: GitHub

6.2 Migration Workflow:

1. Connection: Python scripts connect to the SQL Server using pyodbc and to MongoDB using pymongo.

2. Data Retrieval: SQL SELECT queries fetch records from relational tables.

3. Transformation: Data is reshaped into document format, including nesting and field renaming where necessary.

4. Insertion: Transformed documents are inserted into MongoDB using insert_many() or insert_one().

6.3 Migration Scripts

Script	Migrates From SQL Table	Target MongoDB Collection
client_migration.py	client	Clients
eye_examination_migration.py	eye examination	eye_examinations
employee_migration.py	employee	employees
sales_worker_migration.py	salesworker	salesworkers
medical_worker_migration.py	medicalworker	medicalworkers
manager_migration.py	manager	managers
product_migration.py	product	products
supplier_migration.py	supplier	suppliers
invoice_migration.py	invoice, included_in	invoices
purchase_migration.py	purchase	purchases
makes_migration.py	makes	makes
supplies_migration.py	supplies	supplies

included_in_migration.py included_in included_in

6.4 Challenges faced

During the development and migration of the Optical Store Management System from SQL Server to MongoDB, several technical and practical challenges were encountered:

Python-to-SQL Server Connectivity Issues

One of the primary challenges was establishing a reliable connection between Python and SQL Server. Initial attempts to use the pyodbc library were met with:

- Timeout errors
- Authentication failures
- Incorrect or missing driver configurations (SQL Server vs SQL Server Native Client)

These issues required:

- Adjusting connection strings to properly include Trusted_Connection=yes
- Verifying server name, database name, and driver compatibility

```
sql_conn = pyodbc.connect(
    'DRIVER={ODBC Driver 17 for SQL Server};'
    'SERVER=DESKTOP-XXXXXXX;'
    'DATABASE=NOSQL_PROJECT_OPTICAL;'
    'Trusted_Connection=yes;'
)
```

7. Conclusion

This project successfully demonstrated the end-to-end migration of a relational Optical Store Management System into a NoSQL (MongoDB) format. The transformation preserved key data relationships, optimized data access patterns, and highlighted the advantages of flexible schema modeling.

7.1 What Was Accomplished

- Designed SQL and NoSQL schemas with complete sample data.
- Wrote and executed 13 Python scripts to programmatically migrate data.
- Replaced relational JOIN-heavy access with embedded documents and references.
- Verified data integrity using record counts and document structure inspection.

7.2 Lessons Learned

- MongoDB’s flexibility is ideal for evolving business logic and nested records.
- Data modeling in NoSQL requires understanding of access patterns, not just normalization.
- Validating large migrations requires a mix of automation and manual checks.
- Planning the order of script execution is key to resolving references properly.