RELATIONAL DATABASE DESIGN

Report submission as a requirement for the module of "Advance Data Management"

MSc Degree in Data Analysis

at the school of computing

Robert Gordon University

Aberdeen, Scotland

October 2020

Table of Contents

Table of Figures & Tables

Introduction

This project aims to implement industry relational database design methods and processes for a database structure, demonstration of this methods and thought process will be produced with the goal to evidence my skills and results.

Firstly, I will give a project overview followed by the identification of entities, attributes and relations. I will create a conceptual design and translate it into tables where I will apply the normalisation process to eliminate redundancy in tables. I will share one last update of my design and final schema, which will be ready for SQL implementation.

1. Conceptual Model Design

The aim of this step is to display a visual representation of all the objects of interest that we wish to store for this project. By identifying all **entities**, their corresponding **attributes** as well as **relationships** between them, we can create a conceptual model which facilitates the understanding and flow of the data in our design.

1.1. Project Overview

The requirements and specifications of the project are found below.

Little Panda is a takeaway who wants to open up its business to accept online orders. To do this, it needs a database to store its food menu, customer data and orders.

Your task is to design a relational database that runs on MySQL.

- Customers must register before they can make orders.
- They must provide enough details for home delivery.
- Menu item prices may change.
- Customers are charged prices at the time of order.
- Little Panda needs to know the status of an order so that they can follow up. e.g. either it is "waiting to be cooked", "cooked and to be delivered", or delivered, etc…
- You can assume all orders are paid before they enter the system.
- Order details must be stored for accounting purpose, even after they are completed.

Other than these it is a usual takeaway scenario. You can apply your common sense to make reasonable assumptions.

1.1.1. Identifying Entities

Three entities have been initially identified based on the requirements and specifications; hence a **Customer**, **Menu**, and **Order** table will be needed. Note that new tables can emerge from the result of normalisation.

1.1.2. Identifying Attributes

Each of the entities found above contains **properties**, listing only the important ones is paramount to achieve full functionality and efficiency for our design.

Attributes for **Customer**

Attribute	Justification
❖ customerID	In order to uniquely identify each row in this table I have included this attribute as **primary key.**
❖ customerName	This attribute stores the name of the customer at registration time.
❖ address	This attribute is needed to store the address of the customer for order delivery purposes.
❖ phone	For contacting the customer in real time.
❖ email	Account username and second point of contact. Since is unique it could have worked as primary key but decided to keep it as a non-primary because is subject to change.
❖ password	Little panda wants to allow for user account registration and online orders, this attribute authenticate the user account.

Table 1: Justification of attributes in customer table

Attributes for **Order**

Attribute	Justification
❖ orderID	Needed in conjunction with dishName to create a **composite key** and uniquely identify each row.
❖ dishName	**Foreign key** which connects the order table and menu table. Also is part of the **composite key** which uniquely identifies each row.
❖ customerID	**Foreign key** which connects the order table and customer table.
❖ quantity	Helps eliminate redundancy and can be used to query the sum with pricePerUnit for the total price.
❖ pricePerUnit	Stores the price of a meal through time. keeps a record of old and new prices.
❖ orderStatus	Informs the status of an order in real time and keeps the record weather it was delivered or not.
❖ extraNotes	Holds customer data in case specific meals within an order want to be customized.
❖ toAddress	Holds the address where the order is to be delivered
❖ totalPrice	Contains the total price of the order. Useful for charging the customer and accounting purposes.
❖ dateTaken	Keeps track of when orders where taken. Useful for business insights, customer service and accounting purposes.
❖ dateDelivered	Keeps track of when orders were delivered. Useful for business insights or customer service.

Table 2: Justification of attributes in order table

Attributes for Menu

Attribute	Justification
❖ dishName	Stores the name of the meals. Since all meals have a unique name this attribute has been selected as the **primary key**
❖ description	Introduces a description of the meal to customers
❖ ingredients	Contains meal ingredients for customer awareness and as per UK food labelling regulation.
❖ currentPrice	Stores the price of a meal in present time.

Table 3: Justification of attributes in menu table

1.1.3. Identifying Relations

In this section, we **associate** each entity with each other; this step describes how data should flow in our design.

Relation	Reasoning
Customer → Order	An instance of a customer might exist; this customer might have purchased no orders yet or might have bought many of them. Its relation towards Order is **zero to many.**
Order → Customer	An instance of an order can't exist without being created by a customer, and only one customer owns each order. Order relation towards customer is **one and only one.**
Order → Menu	An instance of an order can't exist without containing at least one menu item, but there can be more than one item on an order. Order relation towards menu is **one or many.**
Menu → Order	An instance of a menu item exists without being contained within an order. One or many menu items might be ordered. Menu relation towards order is **zero to Many.**

Table 4. This is a table representing the relationship between entities

1.2. ER Diagram

The following diagram represents the relationship between entities and their attributes in a visual way. It also includes the **primary** and **composite keys** that uniquely identify each row.

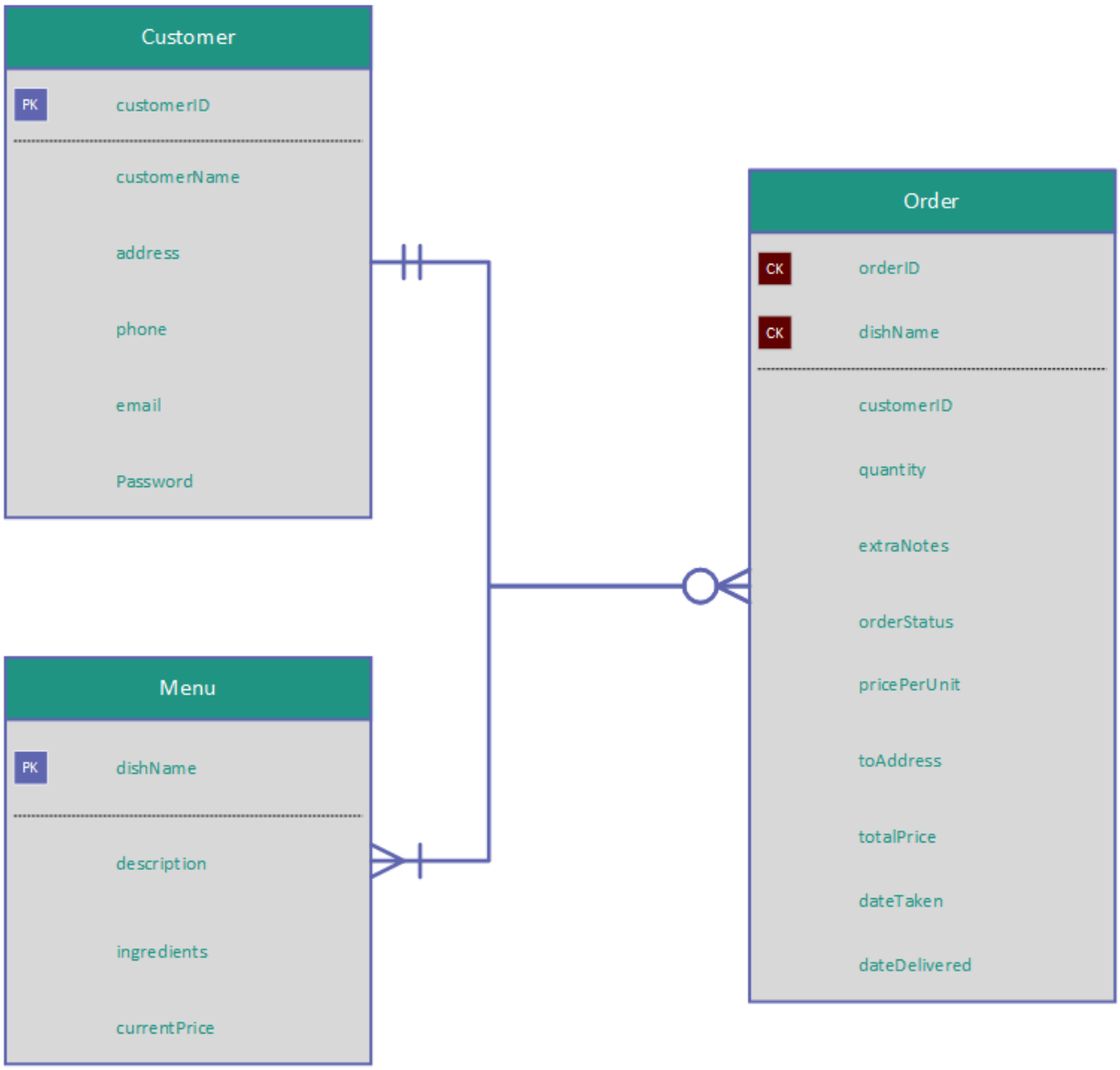

Figure 1: This is a visual representation, the ER Diagram for Little Panda Takeaway

2. Schematization and Normalization

In this section, I apply the **normalisation** process to minimise data redundancy, improve data integrity and represent the data more accurately. Each entity has been **schematised** into tables to allow for a better representation of this process.

2.1. Schema & Normalization of Customer Table

customerID	customerName	address	phone	email	password
#C1685	Rafael Castillo	19 Morningside Grove AB10 7DJ, Aberdeen	07572018284	r.castillo@rgu.ac.uk	Gh47fk8d7
#C2658	Kit Ying Hui	55 Hadoop street AB72 4PG, Aberdeen	07875036968	k.hui@rgu.ac.uk	Fghfghf45
#C9658	Robert Gordon	Garthdee House, Garthdee Road AB10 7QB, Aberdeen	01224263666	soc@rgu.ac.uk	68hndfgd5

Table 5. Schematisation of the Customer table for normalisation purposes

Normalisation in 1NF Customer Table

customerName could be decomposed into **first** and **last** name, but there is no sign of Little Panda needing this decomposition. (could be added if required)

address could be decomposed into **street**, **postcode** and **city** but I am assuming Little Panda is not a national or international takeaway restaurant but a local one hence there is no need for this decomposition. There is room for discussing whether it would be suitable to decompose the address in case we would like to list the streets or sectors from the city were the restaurant benefits the most for cases such as (e.g. Delivering flyers, promotions, re-location or the opening of a new franchise) but that would me more suited for a data warehousing structure (i.e. OLAP) and not this operational one.

- ✓ All attributes are considered in their **atomic** form hence this table is already in 1NF

Normalisation in 2NF Customer Table

- ✓ 2NF only applies when the key is composite, but the customer table only contains a primary key.

Normalisation in 3NF Customer Table

- ✓ The customer table has customerID as the primary key which relates directly to each of its attributes and no **transitive functional dependency** exist hence why this table is in 3NF

2.2 Schema & Normalization of Menu Table

dishName	currentPrice	description	ingredients
Fish curry	8.99	Lorem ipsu	Fish, Curry, poppadum
Chicken Noodles	6.50	Lorem ipsu	Chicken, veggies, noodles
Tikka Masala	10.00	Lorem ipsu	Chicken, veggies, masala spice

Table 6. Schematisation of the Menu table for normalisation purposes

Normalisation in 1NF Menu Table

- ✓ All attributes are already in their **atomic** form hence the menu table is already in 1NF

Normalisation in 2NF Menu Table

- ✓ 2NF only applies when the key is composite, but the menu table only contains a primary key.

Normalisation in 3NF Menu Table

- ✓ The menu table has dishName as the primary key which relates directly to each of its attributes and no **transitive functional dependency** exist hence why this table is in 3NF

2.3 Schema & Normalization of Order Table

orderID	dishName	customerID	quantity	pricePerUnit	orderStatus	extraNotes	toAddress	totalPrice	dateTaken	dateDelivered
#1	noddle	#C2344	2	6.50	cooking	Lorem i	19 mor	13.00	1/11/2020	3/11/2020
#2	masala	#C764	3	10.00	deliver	Lorem i	56 sun	30.00	1/11/2020	1/11/2020
#2	curry	#C764	1	8.99	deliver	Lorem i	56 sun	8.99	1/11/2020	1/11/2020

Table 7. Schematisation of the Order table for normalisation purposes

Normalisation in 1NF Order Table

- ✓ All attributes are already in their **atomic** form hence this table is already in 1NF

Normalisation in 2NF Order Table

Functional dependencies from Table 7

- ✓ {orderID, dishName} → quantity
- ✓ {orderID, dishName} → pricePerUnit
- ✓ {orderID, dishName} → extraNotes
- ✓ {orderID, dishName} → totalPrice

Partial dependencies from Table 7

We can see above all of the dependencies that our composite key determines; this is mostly because those attributes can be different regardless if they belong to the same orderID. In contrast, we have **partial dependencies** that are determined by the orderID alone; these attributes become redundant when a customer orders multiple diverse dishes and are repeated over and over; 2NF helps us identify this issue. We can use table decomposition to eliminate redundancy.

- ✓ orderID → toAddress
- ✓ orderID → customerID
- ✓ orderID→ dateTaken
- ✓ orderID → dateDelivered
- ✓ {orderID, dishName} → orderStatus

Examples of **failed functional dependencies**

dishName can't be determined by the orderID because we can have the same order number requesting for different dishes as it is the case in out dummy example in table 7.

- orderID→ dishName **X**

dishName can't be determined by customerID and orderID because a customer might have multiple dishes in the same order.

- {orderID, customerID } → dishName **X**

orderID can't be determined by dishName and customerID because the same customer might order the same dish on the same or different date.

- { dishName, customerID } → orderID **X**
- { dishName, customerID, anyDate } → orderID **X**

pricePerUnit can't be determined by dishName because prices are subject to change.

- { dishName } → pricePerUnit **X**

Result of normalisation

orderID	dishName	quantity	pricePerUnit	extraNotes
#1	noddle	2	6.50	Lorem ipsu
#2	masala	3	10.00	Lorem ipsu
#2	curry	1	8.99	Lorem ipsu

Table 8: result of Table 7 after normalisation. Table new name: OrderDetails

Normalisation in 3NF Menu Table

- ✓ The OrderDetails table has dishName and orderID as the composite key which relates directly to each of its attributes and no **transitive functional dependency** exist hence why the table is in 3NF

2.4 Schema & Normalization of New Order Table

During **2NF** all partial dependencies must be decomposed from the current table into a new one. Below is the new table after the normalising process in Table 7. The name I have chosen for this table is **Order,** and I have changed the name of Table 7 to **OrderDetails**.

Design change

After critically evaluating the outcome of this normalisation, I have decided to move the **totalPrice** attribute to the order table. The reasoning behind this decision is because after decomposition the OrderDetails table became more focus on the individual meal which is being prepared and a total price makes sense in a table which deals with the total outcome of the customer's purchase hence why it has been moved to the Order table.

orderID	customerID	toAddress	totalPrice	dateTaken	dateDelivered	orderStatus
#1	#C2344	19 mor	13.00	1/11/2020	3/11/2020	cooking
#2	#C6764	67 dec	38.99	1/11/2020	1/11/2020	deliver
#3	#C7645	80 pla	8.99	1/11/2020	1/11/2020	deliver

Table 9. New emerged order table out of 2NF normalization

Normalisation in 1NF new Order Table

- ✓ All attributes are already in their **atomic** form hence the order table is already in 1NF

Normalisation in 2NF new Order Table

- ✓ 2NF only applies when the key is composite, but the order table only contains a primary key.

Normalisation in 3NF new Order Table

- ✓ The order table has orderID as the primary key which relates directly to each of its attributes and no **transitive functional dependency** exist hence why this table is in 3NF
- ✓

Examples of **failed transitive dependencies**

toAddress is not a transitive dependency of customerID because a registered customer with an existing address might order food from and for a different location (address ≠ toAddress)

- orderID → customerID → toAddress **X**

2.5 Updated ER Diagram

Updated relations

Relation	Argument
Customer → Order	An instance of a customer might exist; this customer might have purchased no orders yet or might have bought many of them. Its relation towards Order is **zero to many.**
Order → Customer	An instance of an order can't exist without being created by a customer, and only one customer owns each order. Order relation towards customer is **one and only one.**
Order → Order Details	An instance of an order can't exist without having the details of the order. An order might contain one or more order detail instances. Relation towards order details is **one to many.**
Order Details → Order	One or many instances of order details belong to an order. Order details relation towards order is **one and only one.**
Order Details → Menu	An instance of an order detail can't exist without containing one menu item, and there can only be one item per Order detail. relation towards menu is **one and only one.**
Menu → Order Details	An instance of a menu item exists without being contained within an order. One or many menu items might be ordered. Menu relation towards order is **zero to Many.**

Table 10: Updated relations between tables for the final Schema

3 Final Schema

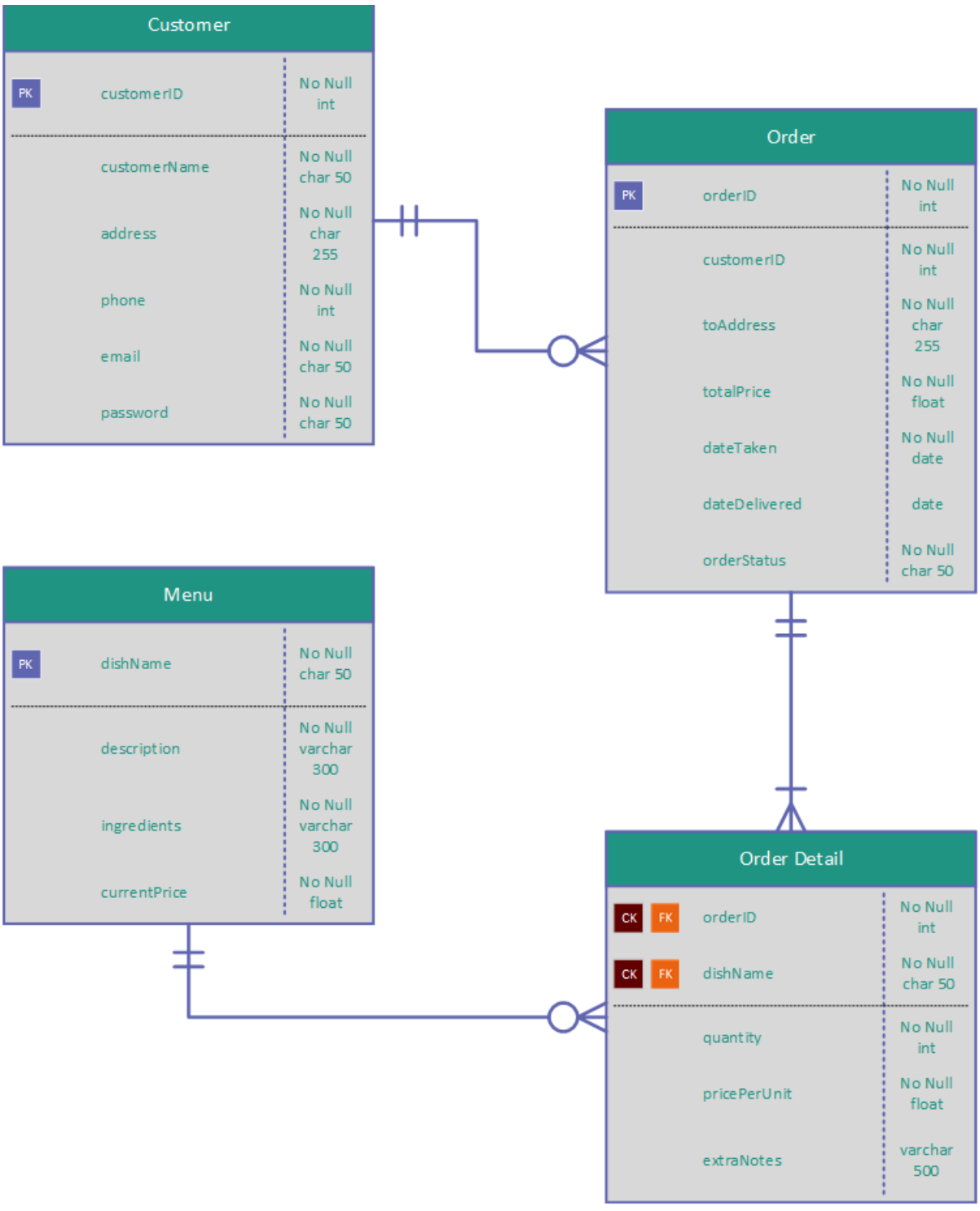

Figure 2:The final schema

4 Limitations

- If Little Panda wanted to include discounts, promotions or combos this design wouldn't be able to handle it efficiently or **store** this information through time which could be useful for accounting purposes or **BI** (i.e. Business Intelligence)
- There is a date but not a time attribute. Time can allow for more insight as to which times of the day the restaurant is busier. This attribute would be useful for **BI** (i.e. Business intelligence)
- Per coursework specification I am not including any attributes or tables that discusses payment, but Order table could see the addition of an attribute whether an order was paid or not.

5 Conclusions

In this project, I organised Little Panda specifications into relevant entities and evaluated different attributes and relations between the entities which pertain to the domain. Justification of all relevant assumptions of attributes and relations was produced. I also Designed an initial and final ER which evidence can be found in this document. Structured the ER diagram into schematic tables and implemented the normalisation processes in 1NF, 2NF and 3NF; as a result, I reduced redundancy in the design as well as gained insight and improve my relational database design skills. I also analysed, changed and improved the design to better fit entities. Delivered a final ER Diagram containing the solution for Little Panda database system.