# CS4416 Project Report

## Nicole Berty - 18246702

## Michele Cavaliere - 18219365

## Michael Donegan - 18235549

# Table of Contents

[**CS4416 Project Report**](#_p3s8bw3cxyze) **1**

[Nicole Berty - 18246702](#_x1ufyrp6up7) 1

[Michele Cavaliere - 18219365](#_rgp7agjxb31a) 1

[Michael Donegan - 18235549](#_se3pcat448ne) 1

[**Table of Contents**](#_glqfvv62o73o) **2**

[What parts of the project each student worked on:](#_1xvlnytlii7p) 4

[*Part A – Design a relational database schema*](#_4c84ic9w9qkg) *4*

[*Part B – Implement a MySQL-compatible relational database schema*](#_2q89i15h1kcl) *4*

[*Part C – Write a report which includes the following*](#_z0tplr1p1b6u) *4*

[What is our database about?](#_n9t89muh6fs2) 5

[*Fast Food Restaurant*](#_hpgwwgte4ugz) *5*

[*Suppliers:*](#_x4zx6s06ex6) *5*

[*Purchases:*](#_nwp5701r8iv1) *5*

[*Inventory:*](#_kuuug3v731mf) *5*

[*Menu:*](#_z6c9grfy4anz) *6*

[*Sales:*](#_4ko81io64b2m) *6*

[*Staff:*](#_9f34lpqgc0fv) *6*

[Entity-relationship diagram for our database:](#_od4esa6qjsid) 8

[Example of each table with some data and PK attributes identified](#_ljwh2mi3p979) 9

[Functional Dependencies and their keys for each table:](#_7sosqd1pvwyj) 11

[*Suppliers:*](#_bjs4zawfvzr3) *11*

[*Purchases:*](#_ekmyx1o41xio) *11*

[*Inventory:*](#_ov92bw4vplaq) *11*

[*Menu:*](#_f5ueqra7h4sv) *11*

[*Sales:*](#_hulnifau6gz7) *11*

[*Staff:*](#_4xepechycf1l) *12*

[Proof that each table is in BCNF and thus 3NF](#_l2t79oqtacjr) 13

[Justification for the usefulness of the views proposed in part B](#_hy3eflg04g) 14

[*View 1: SALES\_PER\_EMPLOYEE*](#_bvffmvt9gq8q) *14*

[*View 2: MONTHLY\_PAID\_WITH\_CASH*](#_8zcezvforclr) *14*

[*View 3: SALES\_PER\_DAY*](#_xml1k88wjaz9) *14*

[Analysis of the speed of the queries in your views and justification for the indexes proposed in part B](#_t154gnkunjp7) 14

[Justification for the necessity of the triggers, stored procedure and function proposed in part B](#_yck0z3cmvzq2) 15

[*Function: StaffLevel*](#_ott6mdo75h99) *15*

[*Stored Procedure: getQuantitySoldByEmployee*](#_x6g4uu74q7pc) *15*

[*Trigger 1: giveBonusForSalesMade*](#_yzem4fxl055) *15*

[*Trigger 2: UpdateStock*](#_j8pnj72zjjxk) *15*

### What parts of the project each student worked on:

#### Part A – Design a relational database schema

1. All students worked on creating the relational database schema. We brainstormed and came up with the idea of a database for a fast food restaurant.
2. Nicole asserted the functional dependencies and identified their keys
3. All students created and contributed to the entity relationship diagram, which we drew using Lucidchart

#### Part B – Implement a MySQL-compatible relational database schema

1. All students wrote the code for the schema.sql file with the insert and create statements. We came up with names of companies, products, staff, etc. to use for the insert statements.
2. Mighele wrote the code for the views.sql file
3. Mighele and Michael wrote the triggers.sql file

#### Part C – Write a report which includes the following

1. All students wrote the description of the database and explained what it was about
2. Michael took the screenshots to provide an example of each table with data and primary key attributes identified
3. Nicole came up with the list of functional dependencies for each table
4. Nicole wrote the proof of each table being in 3NF/BCNF form
5. Mighele provided the justification for the usefulness of the views we created
6. We all provided justification for why we did not use indexes in our tables
7. Michael provided the justification for the necessity of the triggers, stored procedure and function from part B.

### What is our database about?

#### Fast Food Restaurant

Our database models a fast food restaurant, both behind the cash register and in front. We imagine that this could be used in a software system for a place such as Burger King. Our database consists of six tables: Suppliers, Purchases, Inventory, Menu, Sales, and Staff. When the business purchases ingredients, it stores the details in the purchases table, which is linked to the suppliers table using the supplier name. The items bought are put into the inventory with their quantity and expiry date noted. These ingredients are made into a dish and they are added to the menu with their price, size and allergies. When a customer purchases something from the menu, it is added to the sales table. Staff members have their details listed in the staff table, and are linked to the sales table by the sales they make.

##### Suppliers:

The suppliers table contains details of the companies that supply the fast food restaurant and their contact details. It contains three attributes: company name, phone number and address. Company name is the primary key for this table, as two companies cannot operate under the same name so it must be unique, and not null. Phone number is unique as two suppliers cannot have the same number, but it is also not necessary to have a phone number so this attribute can be null. Address is not unique as multiple businesses may be operating in one building for example. The suppliers table has a zero-or-many relationship to the purchases table, as a supplier may have supplied zero or multiple purchases. Purchases has a one or many relationship to suppliers as each purchase must have one or more suppliers involved.

##### Purchases:

The purchases table details the products purchased by the restaurant. The table contains the attributes: product supplied, price, quantity of the product, and suppliers. This table has a composite key consisting of product supplied and supplier as it’s primary key. This is due to the fact that a particular type of product, eg. potatoes, may be supplied by multiple different companies, and a particular company may supply various products but a specific product supplied by a specific company is a unique and non null value. These two values can be used to determine the quantity of the product supplied as well as the price. Supplier is also a foreign key in this table as it is the same attribute as the company name in the Suppliers table (where it is a primary key). Purchases has a one or many relationship to inventory, as each purchase will involve adding one or multiple items to the inventory. Inventory has a one or many relationship to purchases as when a purchase is made the products purchased are added to the inventory.

##### Inventory:

The inventory table is used to keep a record of the products contained in the restaurant’s inventory. It contains the following four attributes: product id, product name, quantity stored and expiry date. This table’s primary key is product id. If you have the id of the product in the inventory you can find the name of the product, the quantity stored and it’s expiry date. Name is a foreign key in this table as it refers the to product supplied attribute in the Purchases table. Name cannot be used to find the product id as you can have a product such as “Chicken Breasts” in the table twice with different expiry dates and quantities, meaning this attribute is not unique in this table. Quantity is not unique as you could have the same number of different products and it can also be null. Expiry date is not unique as products could expire on the same date, and the default expiry date is null as many products such as those in jars and tins will not expire. Inventory has a zero or many relationship with the Menu table as the menu may not contain particular inventory item or it may contain multiple. Menu has a one or many relationship to Inventory as the menu must contain at least one item from inventory.

##### Menu:

The menu table contains information about the food offered in the restaurant. It contains six attributes: dish name, price, size, allergies, dish id and ingredient id. Dish id is the primary key for this table as it can be used to identify the other attributes, it is unique and it cannot contain a null value. Dish name is not unique as you can have two dishes with the same name such as “Snack Box” but which refer to different sizes (large, medium, small) and thus also have different prices. Price and size are not unique as multiple dishes can have the same price or size. ‘Regular’ is the default value for size. Allergies is not unique and it is also not guaranteed to be a non null value, as some dishes have no allergens in them. NULL is the default value for allergies. Ingredient\_id is a foreign key in this table, as it is a primary key (product\_id) in the inventory table. Ingredient id is the id of the ingredients used to make the dish. Menu has a zero or many relationship to the Sales table as a menu item may not have been sold or it could have been sold one or multiple times. Sales has a one or many relationship to menu as each sale must contain at least one item from the menu.

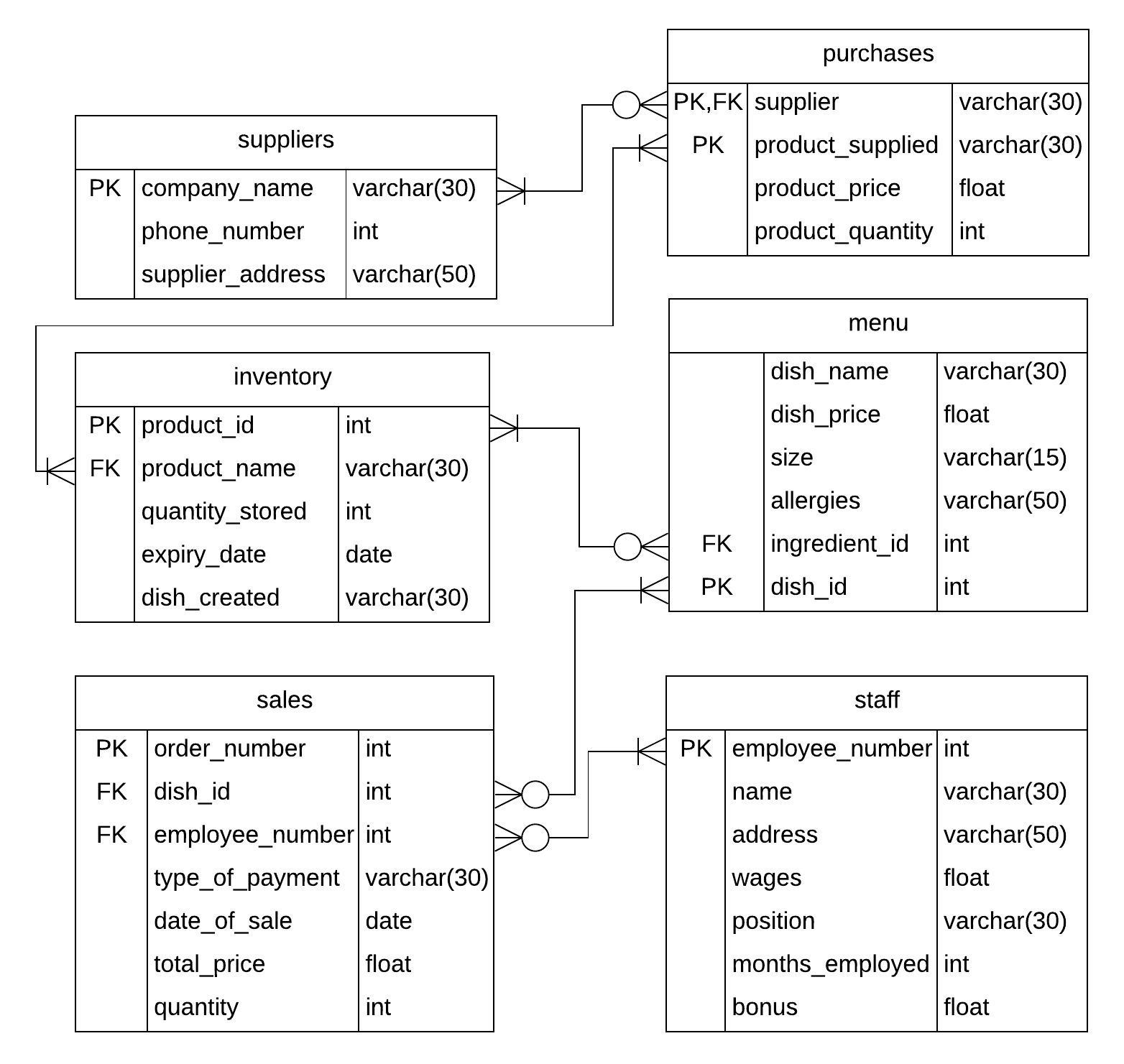
##### Sales:

The sales table is used to track the sales made in the restaurant. It contains the following attributes: type of payment (cash or debit/credit card), date of sale, total price, quantity, dish id (a foreign key in this table, as it references the primary key dish id in the Menu table), order number and employee number (another foreign key, it references the primary key employee number in the Staff table). Order number is the primary key for the Sales table as it can uniquely identify all other attributes of the table. ‘Cash’ is the default value for the type of payment attribute, as it is most commonly used for fast food restaurants, where most meals are quite cheap. Sales has a one or many relationship to the staff table, as each sale must involve at least one staff member. Staff has a zero or many relationship to the sales table as not every staff member has made or will make a sale but some make multiple sales.

##### Staff:

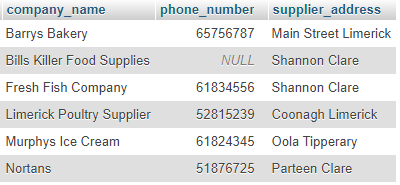
The staff table contains the details about the staff employed in the restaurant. It contains the following attributes: employee number (primary key), name, address, wages, position (waiter, cashier, chef), months employed and bonus. Employee number is the primary key for this table. None of the other attributes are unique and bonus can contain NULL values (and NULL is the default value for the bonus attribute). Zero is the default value for the months employed attributes, as when a new employee is inserted into the staff table they are assumed to have just started working for the business.

### Entity-relationship diagram for our database:



### Example of each table with some data and PK attributes identified

**Suppliers**

****

*Primary key:* company\_name

**Purchases**

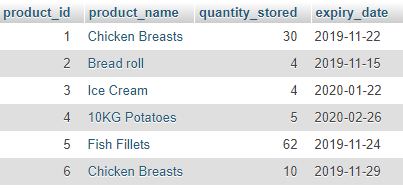
****

*Primary key:* (composite) product\_supplied, supplier *Foreign key:* supplier

(References attribute company\_name

in Suppliers table)

**Inventory**



*Primary key:* product\_id *Foreign Key:* product\_name

(References product\_supplied attribute

in purchases table)

**Menu**

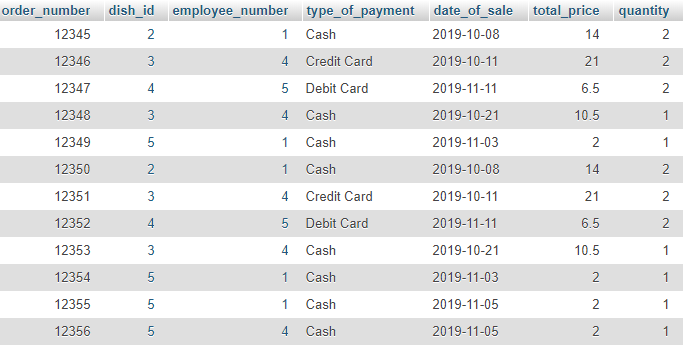
****

*Primary key:* dish\_id *Foreign Key:* ingredient\_id

(References product\_id attribute

in inventory table)

**Sales**



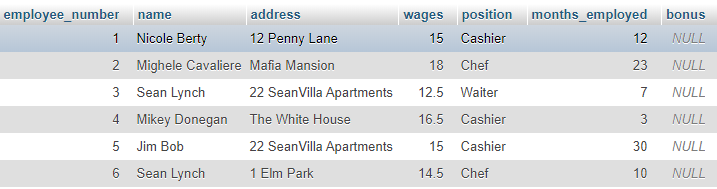
*Primary key:* order\_number  *Foreign Keys:* dish\_id, employee\_number

(Dish\_id referenced from menu

table, employee\_number referenced

from staff table)

**Staff**

****

*Primary\_key:* employee\_number

### Functional Dependencies and their keys for each table:

#### Suppliers:

Company\_name → phone\_number

Company\_name → supplier\_address

{company\_name}+ = {company\_name, phone\_number, supplier\_address}

Key: company\_name

#### Purchases:

Product\_supplied, supplier → product\_price

Product\_supplied, supplier → quantity

{Product\_supplied, supplier}+ = {product\_supplied, supplier, product\_price, quantity}

(Composite) key: product\_supplied, supplier

#### Inventory:

Product\_id → product\_name

Product\_id → expiry\_date

Product\_id → quantity\_stored

Product\_id → dish\_created

{product\_id}+ = {product\_id, product\_name, expiry\_date, quantity\_stored, dish\_created}

Key: product\_id

#### Menu:

dish\_id → dish\_name

dish\_id → dish\_price

dish\_id → size

dish\_id → allergies

dish\_id → ingredient\_id

{dish\_id}+ = {dish\_id, dish\_name, dish\_price, size, allergies, ingredient\_id}

Key: dish\_id

\*ingredient\_id cannot determine the other attributes as one ingredient (eg potatoes) may be used to make more than one dish, as seen in the screenshot in section 4.

#### Sales:

order\_number → type\_of\_payment

order\_number → date\_of\_sale

order\_number → total\_price

order\_number → quantity

order\_number → dish\_id

order\_number → employee\_number

{order\_number}+ = {order\_number, type\_of\_payment, date\_of\_sale, total\_price, quantity, dish\_id, employee\_number}

Key: order\_number

#### Staff:

employee\_number → name

employee\_number → address

employee\_number → wages

employee\_number → position

employee\_number → months\_employed

employee\_number → bonus

{employee\_number}+ = {employee\_number, name, address, wages, position, months\_employed, bonus}

Key: employee\_number

### Proof that each table is in BCNF and thus 3NF

A table is in Boyce Codd Normal Form (BCNF) if for every non-trivial FD A → B, A is a superkey. A table is in third normal form (3NF) if for every FD either the LHS is a key or superkey or RHS is prime.

Suppliers:

The left hand side of each functional dependency consists of the minimal superkey company\_name.

Purchases:

The left hand side of each functional dependency consists of the key product\_supplied, supplier.

Inventory:

The left hand side of each functional dependency consists of the key product\_id.

Menu:

The left hand side of each functional dependency consists of the key dish\_id.

Sales:

The left hand side of each functional dependency consists of the key order\_number.

Staff:

The left hand side of each functional dependency consists of the key employee\_number.

Each table above is in Boyce Codd Normal Form as the left hand side of the functional dependencies for each table is a superkey for the table, as seen in [part 5](#_7sosqd1pvwyj) above.

### Justification for the usefulness of the views proposed in part B

#### View 1: SALES\_PER\_EMPLOYEE

This view will be useful as it can show the business which employees are selling the most items and can allow the business the opportunity to give bonuses based on employee sales, which ties in with one of our triggers also. If the business can see which employees are most effective at selling, they can also schedule these employees to work during peak times and put them on the register instead of clearing tables for their shifts, for example. This view is based on the quantity of products sold by the employee (using SUM(quantity)) rather than being calculated using COUNT(order\_number) because while an employee cannot really control whether or not a person buys food in a fast food restaurant, they can potentially convince people to buy extra items and thus sell more products.

#### View 2: MONTHLY\_PAID\_WITH\_CASH

This view shows the number of orders along with the total price of those orders and the date of them that were paid for in cash within the last month. It is important for the company to know how many people paid with cash as they need to keep records and receipts for revenue/tax purposes. People who have paid via debit/credit card will have a digital record of their payment, but for cash the company needs to keep track of how much is paid and when.

#### View 3: SALES\_PER\_DAY

This view shows the amounts of the sales made for each date in the sales table. It can help the company to see which days bring in the most customers so that they can adjust their business plan accordingly and decide on staff rotas for the restaurant, putting more staff on when there tends to be a larger influx of customers. It can also give them the opportunity to put special deals (eg. 2 chicken buckets for €34) on the days of the week when customers are less frequent, in an effort to boost their business on those days).

### Analysis of the speed of the queries in your views and justification for the indexes proposed in part B

Indexes are used to improve the response time of queries, but have the potential to slow queries down as every time data is updated in the tables the indexes may need to be updated also. We did not create any indexes in our database. Indexes should not be used on tables that will be manipulated frequently, which will be the case for our restaurant database as columns such as expiry date, dish name, dish price, product id, product name will all change on a regular basis depending on the time of year, how the business is doing financially, which suppliers are selling a particular product and for what price and so on.

### Justification for the necessity of the triggers, stored procedure and function proposed in part B

#### Function: StaffLevel

The function we created sets an employee’s “Level” based on the quantity of items they have sold in the past year. Their level can be one of: Platinum (sold more than 25 products), Gold (sold between 15 and 25 products) or Silver (sold between 10 and 15 products) . This function returns one of these levels based on the product quantity that is used as an input for it.

#### Stored Procedure: getQuantitySoldByEmployee

This stored procedure is used to calculate the quantity of products sold by a particular employee in the past year by taking their ID number as an input. This can be used in conjunction with the StaffLevel function to reward employees, as we did with our first trigger.

#### Trigger 1: giveBonusForSalesMade

This trigger is activated after there is an insert on the sales table, and it uses both the function and stored procedure. The trigger gives an employee a bonus which is calculated by passing the value produced by the procedure (the quantity of goods sold) to the function in order to calculate the employee’s level and thus the bonus they should receive. A Platinum level employee gets a €250 bonus, a Gold level employee gets €150 bonus, and a Silver level employee gets a €75 bonus. This trigger is useful as it allows the business to reward their employees for selling more goods, for example by convincing people to buy a dessert with their meal or an extra sauce.

#### Trigger 2: UpdateStock

This trigger is activated after there is an insert on the purchases table. It updates the quantity\_stored attribute in the Inventory table for a particular product by adding the new quantity to the current stock. This trigger is necessary as without it, when the business inserted a new purchase with, for example, 5 bags of potatoes, the quantity of potatoes in inventory would not be updated to match.