**COMP3350**

**Advanced Database**

**Assignment 1**

Database Design & Implementation

Database Documentation

Tasty’n’Yummy Pizzas

Liam Craft | c3339847

Lukas Binninger | c3332295

Contents

[Preface 2](#_Toc67681241)

[Part I: Assumptions 4](#_Toc67681242)

[1. Business Rules 4](#_Toc67681243)

[1.1. Member 6](#_Toc67681244)

[2. Assumptions 7](#_Toc67681245)

[Part II: Entity Relationship Diagram (ERD) 8](#_Toc67681246)

[3. EER Model 8](#_Toc67681247)

[4. Data Dictionary 9](#_Toc67681248)

[4.1. Entity Types 9](#_Toc67681249)

[4.2. Relationship Types 10](#_Toc67681250)

[4.3. Attributes 11](#_Toc67681251)

[Part III: Database Schema 16](#_Toc67681252)

[5. Relational Model 16](#_Toc67681253)

[6. Normalization 20](#_Toc67681254)

[7. Final Normalized Relational Schema 27](#_Toc67681255)

# Preface

This is the database documentation bla bla.

From the assignment and project specification:

The **School of Computer Science (SCS) at the University of Sunshine** manages and lends resources to staff and students for assignments and project purposes. Resources like rooms, books, cameras, microphones, speakers, etc. are hard to manage which is why they want to setup a database for their application.

While the first two assignments were about the analysis of requirements and the conceptual and logical design of the database, this one builds on the previous work realizing the physical design of the database. The normalized relations that were the main result of the second assignment are implemented in in the form of tables in SQL with the help of the data dictionary**.**

**Main mission and objectives** of the project called “**SCS Resource Management**”:

* Organising resources to provide efficient access to them
* Supporting research and academic activities by providing access to the required resources
* Monitoring equipment demand and use to facilitate best practical delivery of resources at the SCS

**Main features**

* Catalogue service: Web-based searchable catalogue of all resources
* Loan service: Members issue and return movable resources
* Reservation service: Members can reserve resources, if available, ensuring that they gain access to resources for a specific period of time
* Acquisition service: Suggestions for acquiring access to newer and updated materials are elicited from members

# Part I: Assumptions

## Business Rules

Notes for myself

* Pizza outlet at Charlestown
* Computerize records

**Order processing**

* Orders via phone, online (app, website), walk-in
* Deliver and pickup
* Phone:
  + customer’s phone number
  + id of person taking order
  + if previously ordered by phone: name and address appear on screen
  + Customer asked for name, address
  + If not ordered before or info not match, new customer created
* Walk-in:
  + Name recorded to identify order
* App or website:
  + Login via email or phone number and PW
  + Alternatively as guest
* Each order:
  + Date and time of order
  + One or more items
  + Quantity of each item
  + Price of each item
  + Subtotal
  + Discount amount (if any)
  + Discount code (if any)
  + Tax
  + Total amount due
  + Payment method
    - if via card: payment approval number recorded
  + Status
  + Type (phone, walk-in, online)
    - If delivery: driver who delivered recorded
  + Description
  + Possible to pre-order and order to eb fulfilled at a later date/time
* Different type of discount programs all recorded in DB
  + Discount code
  + Description
  + Start date
  + End date
  + Requirements for discount
  + Discount percentage
  + Description

**Menu items, ingredients, suppliers**

* Menu item:
  + Item code (unique)
  + Name
  + Size
  + Current selling price
  + Made up of a number of ingredients
  + Ingredients and quantities used for item
* Ingredient:
  + Code (unique)
  + Name
  + Type
  + Description
  + Stock level at stocktake
  + Date last stocktake was taken
  + Suggested current stock level
  + Reorder level
  + List of suppliers who supply ingredients, supplier can supply many ingredients
* Stocktake
  + Taken each week
  + Actual level of ingredients in store recorded
  + Compared with suggested levels (based on orders for the week)
  + This report used by store manager to order ingredients for following week, information about supplier orders needs to be maintained in DB

**Employees**

* Two types
  + Work in shop
    - Paid hourly
  + Carry out deliveries
    - Paid by number of deliveries
    - Drivers also have driver’s license number
* Employee number
* Firstname
* Lastname
* Postal address
* Contact number
* Tax file number
* Bank details
  + Bank code
  + Bank name
  + Account number
* Payment rate
* Status
* Description
* Hours not regular, record kept for each time employee works
* A shift
  + Start date
  + Start time
  + End date
  + End time
* Orders a driver delivers during a shift kept on record
* Payment rates for shop workers and drivers maintained in DB
* Payments made for each shift to employee’s bank account at end of each week
* Employee payment records maintained in DB

### Member

The SCS has several members, i.e. people associated with it. The members do not just exist as members but rather fall into one of either student members or staff members.

This relationship makes Member the superclass for Student and Staff.

Student

A student member is a member studying at the SCS.

Additional to the data describing members in general, each student is enrolled in a degree at the SCS. Also, every student has a number of points starting at 12 (default) that decrease with certain penalties occurring while loaning or reserving resources.

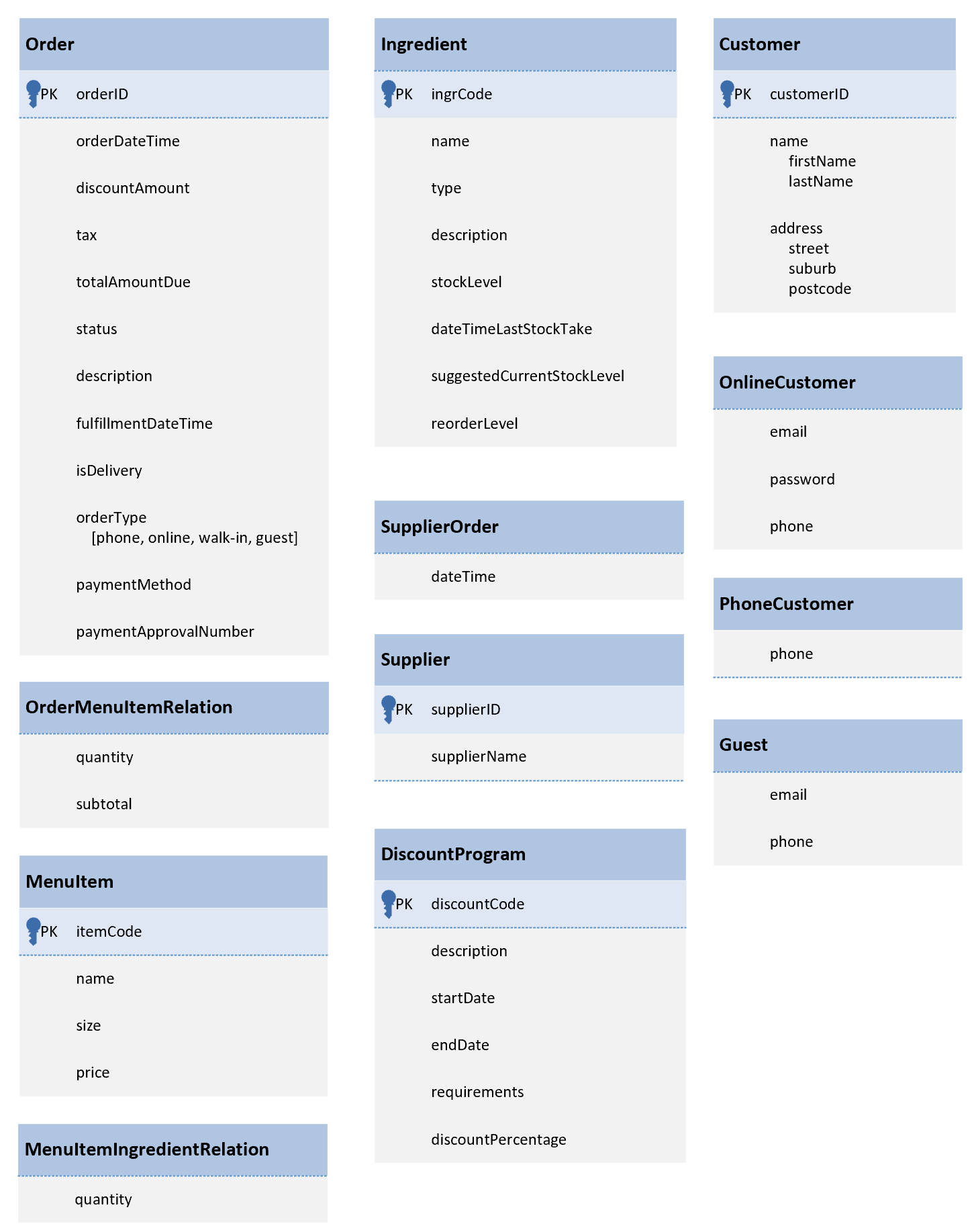
## Assumptions

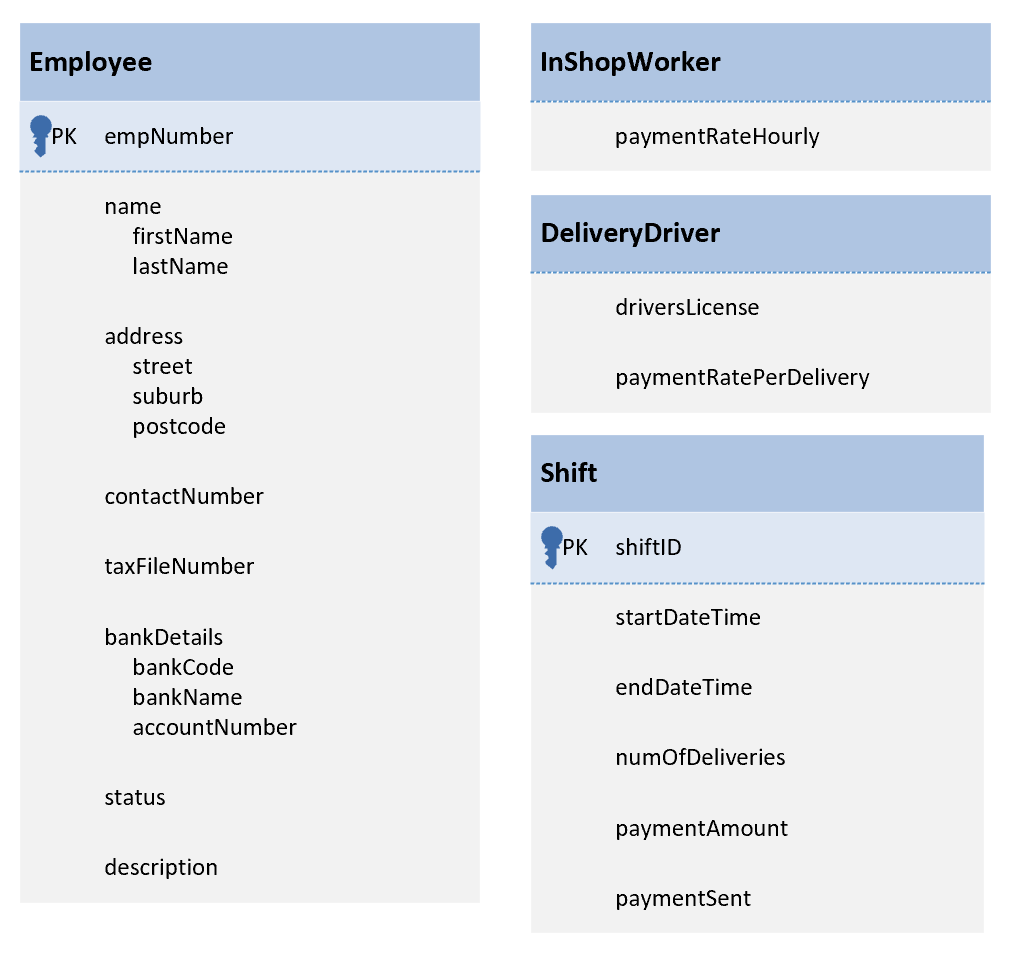
* All types of orders (online, phone, walk-in) whether delivery or pickup are required to be associated with an InShopWorker in our model. The InShopWorker takes the order in our EER’s terminology, but this could just be acknowledging e.g. an online order.

# Part II: Entity Relationship Diagram (ERD)

## EER Model

## Entities and Attributes





## Data Dictionary

### Entity Types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity Name** | | **Description** | **Aliases** | **Occurrence** |
| Customer | | Base class for a store customer; Superclass for OnlineCustomer, PhoneCustomer and (online) Guests; Optional relationship with disjoint subclasses | In-store customer | Can exist independently or as one of it’s subclasses |
| {Optional, OR} | OnlineCustomer | A customer who is logged in and places an order via the store website/app; Optional subclass of Customer |  | When a customer is logged in to the online ordering system |
| PhoneCustomer | A customer who places an order over the phone; Optional subclass of Customer |  | When a customer orders via the phone |
| Guest | A customer who orders via the stores website or app but is not logged in/registered; Optional subclass of Customer |  | When a customer is not logged in, but places an order via the store’s website or app |
| Order | | Describes a request from a customer to be fullfilled by the store | Purchase, Request | When a customer makes a request for goods from the store |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |

### Relationship Types

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entity Name** | **Multiplicity** | **Relationship** | **Multiplicity** | **Entity Name** | |
| Member | 1..1 | creates | 0..\* | AcquisitionRequest |
| 1..1 | makes | 0..\* | Reservation |
| 1..1 | makes | 0..\* | Loan |
| {Mandatory, Or} | Generalisation | {Mandatory, Or} | Student |
| {Mandatory, Or} | Generalisation | {Mandatory, Or} | Staff |
| Student | 1..1 | has | 0..\* | Enrolment |
| Resource | 1..\* | is at | 1..1 | Location |
| 0..\* | has | 1..1 | Category |
| 1..1 | reserved via | 0..\* | Reservation |
| {Mandatory, Or} | Generalisation | {Mandatory, Or} | Immovable |
| {Mandatory, Or} | Generalisation | {Mandatory, Or} | Movable |
| Movable | 1..1 | loaned via | 0..\* | Loan |
| Category | 1..1 | is associated with | 0..\* | Resource |
| 1..1 | is available via | 0..\* | Privilege |
| Location | 1..1 | holds | 1..\* | Resource |
| CourseOffering | 1..1 | enrolled in via | 0..\* | Enrolment |
| 0..\* | grants | 0..\* | Privilege |
| Enrolment | 0..\* | for | 1..1 | Student |
| 0..\* | in | 1..1 | CourseOffering |
| Privilege | 0..\* | for | 1..1 | Category |
| 0..\* | is granted by | 0..\* | CourseOffering |
| Reservation | 0..\* | by | 1..1 | Member |
| 0..\* | for | 1..1 | Resource |
| Loan | 0..\* | is of | 1..1 | Member |
| 0..\* | to | 1..1 | Movable |
| AcquisitionRequest | 0..\* | created by | 1..1 | Member |

### Attributes

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Entity Name** | **Attributes** | **Description** | **Data Type & Length** | **Nulls** | **Multi-**  **valued** | **De-rived** | **Default** |
| Member  superclass | memberId | Unique identifier {PK} given to member of SCS | CHAR(7) | N | N | N |  |
| firstName | First name of a member | VARCHAR(30) | N | N | N |  |
| lastName | Last name of a member | VARCHAR(30) | N | N | N |  |
| streetNo | Street number of the member’s house / apartment | VARCHAR(10) | N | N | N |  |
| street | Street in which the member is living in | VARCHAR(30) | N | N | N |  |
| city | City in which the member is living in | VARCHAR(30) | N | N | N |  |
| postcode | Postcode for the member’s address | INT | N | N | N |  |
| phone | Member’s preferred contact phone number | VARCHAR(30) | Y | N | N |  |
| email | Member’s preferred contact email address | VARCHAR(50) | N | N | N |  |
| status | Status of member; ‘active’ or ‘disabled’; disabled unless enrolled in active courseOffering | VARCHAR(8) | N | N | N | ‘disabled’ |
| comments | Space for some comments about the member | VARCHAR(100) | Y | N | N |  |
| Student  subclass  (Member) | degree | Degree the student is enrolled in | VARCHAR(30) | N | N | N |  |
| points | Points a student has; get reduced via penalties; 0 points make student status ‘disabled’ | INT | N | N | N | 12 |
| Staff  subclass  (Member) | isAdmin | Flag in case staff member is administrator, 0 = No, 1 = Yes | BIT | N | N | N | 0 |
| Resource  superclass | resourceId | Unique resource identifier {PK} | CHAR(10) | N | N | N |  |
| description | Description of the resource | VARCHAR(100) | Y | N | N |  |
| status | Status of a resource; ‘available’, ‘in use’, ‘borrowed’, ‘maintenance’, ‘lost’, ‘damaged’ | VARCHAR(15) | N | N | N | ‘available’ |
| Immovable  subclass  (Resource) | capacity | Capacity or working spaces of an immovable resource | INT | N | N | N |  |
| Movable  subclass  (Resource) | name | Name of the movable resource | VARCHAR(30) | N | N | N |  |
| make | Make of the movable resource | VARCHAR(30) | Y | N | N |  |
| model | Model of the movable resource | VARCHAR(30) | N | N | N |  |
| manufacturer | Manufacturer of the movable resource | VARCHAR(30) | N | N | N |  |
| year | Year of the movable resource | INT | N | N | N |  |
| assetValue | Monetary value of the asset | DECIMAL(9, 2) | N | N | N |  |
| Category | catCode | Unique category code {PK} | CHAR(7) | N | N | N |  |
| name | Name of a category | VARCHAR(30) | N | N | N |  |
| description | Description of the category | VARCHAR(100) | Y | N | N |  |
| maxBookTime | Maximum duration an item of a category can be loaned / reserved | INT | N | N | N |  |
| Location | locationId | Unique location identifier {PK} | CHAR(8) | N | N | N |  |
| room | Room associated with the location | VARCHAR(15) | N | N | N |  |
| building | Building the location is in | VARCHAR(20) | N | N | N |  |
| campus | Campus the location is on | VARCHAR(20) | N | N | N |  |
| Course-Offering  (no hyphen, hyphen just due to limited space) | offeringId | Unique course offering identifier {PK} | CHAR(9) | N | N | N |  |
| courseId | Course identification associated with the offering | CHAR(8) | N | N | N |  |
| name | Name of the course offering | VARCHAR(30) | N | N | N |  |
| semOffered | Semester the course is offered in;  ‘S1’ or ‘S2’ | CHAR(2) | N | N | N |  |
| yearOffered | Year the course is offered in | INT | N | N | N |  |
| begDateTime | Date and time when the course offering commences | datetime | N | N | N |  |
| endDateTime | Date and time when the course offering ends | datetime | N | N | N |  |
| Enrolment | enrolDateTime | Time and date when the student enrolled in a course offering | datetime | N | N | N |  |
| Privilege | privilegeId | Unique privilege identifier {PK} | CHAR(10) | N | N | N |  |
| name | Name of the privilege | VARCHAR(30) | N | N | N |  |
| description | Description of the privilege | VARCHAR(100) | Y | N | N |  |
| maxNum-Resources  (no hyphen) | Maximum number of resources of a certain category that can be borrowed/booked at any given time | INT | N | N | N |  |
| Reservation | reservId | Unique reservation identifier {PK} | CHAR(12) | N | N | N |  |
| startDateTime | Date and time the reservation period starts | datetime | N | N | N |  |
| endDateTime | Date and time the reservation period ends | datetime | N | N | N |  |
| Loan | loanId | Unique loan identifier {PK} | CHAR(12) | N | N | N |  |
|  | loanDateTime | Date and time the loan is made | datetime | N | N | N |  |
|  | dueDateTime | Date and time the loan is due | datetime | N | N | N |  |
|  | returnDateTime | The date and time the loan is returned | datetime | Y | N | N |  |
| Acquisition-Request  (no hyphen) | acqRequestId | Unique acquisition request identifier {PK} | CHAR(8) | N | N | N |  |
| filedDateTime | Timestamp when request was filed | DATETIME | N | N | N |  |
| itemName | Name of the requested item | VARCHAR(30) | N | N | N |  |
| make | Make of the requested resource | VARCHAR(30) | Y | N | N |  |
| manufacturer | Manufacturer of the requested resource | VARCHAR(30) | Y | N | N |  |
| model | Model of the requested resource | VARCHAR(30) | N | N | N |  |
| year | Year of the requested resource | INT | N | N | N |  |
| description | Description about the requested resource | VARCHAR(100) | Y | N | N |  |
| urgency | Describes how urgent the acquisition of a resource is;  Different levels of priority; set by admin | VARCHAR(10) | N | N | N | ‘normal’ |
| status | Status of the request; admin can assign; ‘acquired’, ‘pending’, ‘rejected’, ‘approved‘ | VARCHAR(8) | N | N | N | ‘pending’ |
| fundCode | Code for the funds for the request | VARCHAR(10) | Y | N | N |  |
| vendorCode | Code for the vendor that sells the resource | VARCHAR(10) | Y | N | N |  |
| price | Price of the resource | DECIMAL(11,2) | Y | N | N |  |
| notes | Some space for notes | VARCHAR(100) | Y | N | N |  |

# Part III: Database Schema

## Relational Model

**DiscountProgram** (discountCode, description, startDate, endDate, requirements, discountPercentage)

**Primary Key** discountCode

**Alternate Key** -

**Customer** (customerID, firstName, lastName, street, suburb, postcode)

**Primary Key** customerID

**Alternate Key** -

**CustomerOnlineCustomer** (customerID, email, password, phone)

**Primary Key** customerID

**Alternate Key** email

**Foreign Key** customerID **references** Customer(customerID)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**CustomerGuest** (customerID, email, phone)

**Primary Key** customerID

**Alternate Key** -

// assume several guests could use same email, e.g. same person orders several times as guest

// therefore email not unique

**Foreign Key** customerID **references** Customer(customerID)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**CustomerPhoneCustomer** (customerID, phone)

**Primary Key** customerID

**Alternate Key** -

**Foreign Key** customerID **references** Customer(customerID)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Employee** (empNumber, firstName, lastName, street, suburb, postcode, contactNumber,

taxFileNumber, bankCode, bankName, accountNumber, status, description)

**Primary Key** empNumber

**Alternate Key** taxFileNumber, (bankCode, accountNumber), (firstName, lastName, street,

suburb, postcode)

**EmployeeInShopWorker** (empNumber, paymentRateHourly)

**Primary Key** discountCode

**Alternate Key** -

**Foreign Key** empNumber **references** Employee(empNumber)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**EmployeeDeliveryDriver** (empNumber, driversLicense, paymentRatePerDelivery)

**Primary Key** discountCode

**Alternate Key** driversLicense

**Foreign Key** empNumber **references** Employee(empNumber)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Shift** (shiftID, startDateTime, endDateTime, numOfDeliveries, paymentAmount, paymentSent,

empNumber)

**Primary Key** shiftID

**Alternate Key (**empNumber, startDateTime, endDateTime)

**Foreign Key** empNumber **references** Employee(empNumber)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Order** (orderID, orderDateTime, discountAmount, tax, totalAmountDue, status, description,

fulfillmentDateTime, isDelivery, orderType, paymentMethod, paymentApprovalNumber,

customerID, workerID, driverID)

**Primary Key** orderID

**Alternate Key** (orderDateTime, customerID), (orderDateTime, workerID)

**Foreign Key** customerID **references** Customer(customerID)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Foreign Key** workerID **references** EmployeeInShopWorker(empNumber)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Foreign Key** driverID **references** EmployeeDeliveryDriver(empNumber)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**MenuItem** (itemCode, name, size, price)

**Primary Key** itemCode

**Alternate Key** -

**OrderMenuItemRelation** (orderID, itemCode, quantity, subtotal)

**Primary Key** (orderID, itemCode)

**Alternate Key** -

**Foreign Key** orderID **references** Order(orderID)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Foreign Key** itemCode **references** MenuItem(itemCode)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Ingredient** (ingrCode, name, type, description, stockLevel, dateTimeLastStockTake,

suggestedCurrentStockLevel, reorderLevel)

**Primary Key** ingrCode

**Alternate Key** -

**MenuItemIngredientRelation** (itemCode, ingrCode, quantity)

**Primary Key** (itemCode, ingrCode)

**Alternate Key** -

**Foreign Key** itemCode **references** MenuItem(itemCode)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Foreign Key** ingrCode **references** Ingredient(ingrCode)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Supplier** (supplierID, supplierName)

**Primary Key** supplierID

**Alternate Key** -

**SupplierOrders** (ingrCode, supplierID, dateTime)

**Primary Key** (ingrCode, supplierID)

**Alternate Key** -

**Foreign Key** ingrCode **references** Ingredient(ingrCode)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

**Foreign Key** supplierID **references** Supplier(supplierID)

**ON UPDATE CASCADE, ON DELETE NO ACTION**

## Normalization

In the following the abbreviations for primary key (PK), candidate key (CK), foreign key (FK) and functional dependency (FD) will be used.

Primary keys will be underlined in the relations.

**DiscountProgram** (discountCode, description, startDate, endDate, requirements, discountPercentage)

**Functional dependencies:**

* discountCode (PK) -> description, startDate, endDate, requirements, discountPercentage

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**Customer** (customerID, firstName, lastName, street, suburb, postcode)

**Functional dependencies:**

* customerID (PK) -> firstName, lastName, street, suburb, postcode

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**CustomerOnlineCustomer** (customerID, email, password, phone)

**Functional dependencies:**

* customerID (PK) -> email, password, phone
* email (CK) -> customerID, password, phone

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**CustomerGuest** (customerID, email, phone)

**Functional dependencies:**

* customerID (PK) -> email, phone

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**CustomerPhoneCustomer** (customerID, phone)

**Functional dependencies:**

* customerID (PK) -> phone

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**Employee** (empNumber, firstName, lastName, street, suburb, postcode, contactNumber,

taxFileNumber, bankCode, bankName, accountNumber, status, description)

**Functional dependencies:**

* empNumber (PK) -> firstName, lastName, street, suburb, postcode, contactNumber,

taxFileNumber, bankCode, bankName, accountNumber, status,

description

* TaxFIleNumber (CK) -> empNumber, firstName, lastName, street, suburb, postcode,

contactNumber, bankCode, bankName, accountNumber, status,

description

* bankCode, accountNumber (CK) -> empNumber, firstName, lastName, street, suburb,

postcode, contactNumber, taxFileNumber, bankName,

status, description

* firstName, lastName, street, suburb, postcode (CK) -> empNumber, contactNumber,

taxFileNumber, bankCode,

bankName, accountNumber, status,

description

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**EmployeeInShopWorker** (empNumber, paymentRateHourly)

**Functional dependencies:**

* empNumber (PK) -> paymentRateHourly

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**EmployeeDeliveryDriver** (empNumber, driversLicense, paymentRatePerDelivery)

**Functional dependencies:**

* empNumber (PK) -> driversLicense, paymentRatePerDelivery
* driversLicense (CK) -> empNumber, paymentRatePerDelivery

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**Shift** (shiftID, startDateTime, endDateTime, numOfDeliveries, paymentAmount, paymentSent,

empNumber)

**Functional dependencies:**

* shiftID (PK) -> startDateTime, endDateTime, numOfDeliveries, paymentAmount, paymentSent, empNumber
* empNumber, startDateTime, endDateTime (CK) -> shiftID, numOfDeliveries, paymentAmount, paymentSent

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**Order** (orderID, orderDateTime, discountAmount, tax, totalAmountDue, status, description,

fulfillmentDateTime, isDelivery, orderType, paymentMethod, paymentApprovalNumber,

customerID, workerID, driverID)

**Functional dependencies:**

* orderID (PK) -> orderDateTime, discountAmount, tax, totalAmountDue, status, description, fulfillmentDateTime, isDelivery, orderType, paymentMethod, paymentApprovalNumber, customerID, workerID, driverID
* orderDateTime, customerID (CK) -> orderID, discountAmount, tax, totalAmountDue, status, description, fulfillmentDateTime, isDelivery, orderType, paymentMethod, paymentApprovalNumber, workerID, driverID
* orderDateTime, workerID (CK) -> orderID, orderDateTime, discountAmount, tax, totalAmountDue, status, description, fulfillmentDateTime, isDelivery, orderType, paymentMethod, paymentApprovalNumber, customerID, driverID

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**MenuItem** (itemCode, name, size, price)

**Functional dependencies:**

* itemCode (PK) -> name, size, price

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**OrderMenuItemRelation** (orderID, itemCode, quantity, subtotal)

**Functional dependencies:**

* orderID, itemCode (PK) -> quantity, subtotal

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**Ingredient** (ingrCode, name, type, description, stockLevel, dateTimeLastStockTake,

suggestedCurrentStockLevel, reorderLevel)

**Functional dependencies:**

* ingrCode (PK) -> name, type, description, stockLevel, dateTimeLastStockTake, suggestedCurrentStockLevel, reorderLevel

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**MenuItemIngredientRelation** (itemCode, ingrCode, quantity)

**Functional dependencies:**

* itemCode, ingrCode (PK) -> quantity

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**Supplier** (supplierID, supplierName)

**Functional dependencies:**

* supplierID (PK) -> supplierName

1st NF by default (relation formed out of EER).

2nd NF because there are no partial dependencies (candidate keys are not composite)

3rd NF because there are no transitive dependencies

* BCNF because for all X -> Y in FDs, X is a CK

**SupplierOrders** (ingrCode, supplierID, dateTime)

**Functional dependencies:**

* none
* BCNF, because the only component of the relation is the composite PK (so it only determines itself)

------

**CourseOffering** (offeringId, courseId, name, semOffered, yearOffered, begDateTime, endDateTime)

**Functional dependencies:**

* offeringId (PK) -> courseId, name, semOffered, yearOffered, begDateTime, endDateTime
* courseId, semOffered, yearOffered (CK) -> offeringId, name, begDateTime, endDateTime
* courseId -> name

1st NF by default (relation formed out of EER).

2nd NF? No, because (courseId -> name) is a partial dependency

-> decompose in CourseOffering 1 and 2, use meaningful names

**CourseOffering** (offeringId, courseId, semOffered, yearOffered, begDateTime, endDateTime)

PK: offeringId

FK: courseId references Course(courseId)

ON UPDATE CASCADE, ON DELETE NO ACTION

FDs:

* offeringId (PK) -> courseId, semOffered, yearOffered, begDateTime, endDateTime
* courseId, semOffered, yearOffered (CK) -> offeringId, begDateTime, endDateTime

**Course** (courseId, name)

PK: courseId

FDs:

* courseId (PK) -> name

Now criteria for 2nd NF is satisfied.

3rd NF because there are no transitive dependencies in either relation.

* BCNF because for all X -> Y in FDs, X is a CK

-----

**As a second example to demonstrate the process of normalization**

***Assume for***

**Member** (memberId, firstName, lastName, streetNo, street, city, postcode, phone, email, status, comments)

***postcode uniquely determines city.***

**Functional dependencies:**

* memberId (PK) -> firstName, lastName, streetNo, street, city, postcode, phone, email, status, comments
* email (CK) -> memberId, firstName, lastName, streetNo, street, city, postcode, phone, status, comments
* postcode -> city

1st NF by default (relation formed out of EER).

2nd NF because no partial dependencies (candidate keys are not composite)

3rd NF? No, because memberId -> postcode -> city is a transitive dependency

-> decompose in Member 1 and 2, use meaningful names

**Member** (memberId, firstName, lastName, streetNo, street, postcode, phone, email, status, comments)

PK: memberId

FK: postcode references City(postcode)

ON UPDATE CASCADE, ON DELETE NO ACTION

FDs:

* memberId (PK) -> firstName, lastName, streetNo, street, postcode, phone, email, status, comments
* email (CK) -> memberId, firstName, lastName, streetNo, street, postcode, phone, status, comments

**City** (postcode, cityName)

// Note: It makes more sense to call this relation something meaningful like “City”, therefore I decided to rename the attribute “city” from the original relation to “cityName” in the decomposed relation for this hypothetical case.

PK: postcode

FDs:

* postcode (PK) -> cityName

Now criteria for 3rd NF is satisfied.

* BCNF because for all X -> Y in FDs in either relation, X is a CK

## Final Normalized Relational Schema