## **CMPU 2007 – Databases 1 – Continuous Assessment – Part 2**

## **Burger Shack Database**

## Group Members:

* Adrian Capacite C21348423
* Ernest Decina C21394933

## Domain Types:

|  |  |  |  |
| --- | --- | --- | --- |
| **Domain Type Name** | **Type Definition** | **Reason Introduced** | **Example** |
| Identifier | NUMERIC(10) | All entities must have a unique field to identify each record | RestaurantID in entity restaurant |
| Name | VARCHAR2(50) | Entity name attributes | staffName in entity Staff |
| Address | VARCHAR2(200) | There is Multiple Entities that hold a Address attribute | A Person Entity must have a Address attribute and cannot be Empty |
| Number | VARCHAR(10) | There exists more than 1 Entity that has an attribute of PhoneNumber | A staff Entity must have a Phone Number Linked to them |
| Email | VARCHAR2(50) | Multiple Entities hold and attribute of Email | A Person Entity has an attribute |

## Major Decisions

### Issue 1: Distinction between types of staff

Analysing the given case study, it is prominent that there are 2 given types of staff members present in our database model. These 2 types of staff come to be “Management Staff” & “Waiter Staff”. There is a clear distinction between the types. The “Management Staff” has an elevated moderation status compared to the “Waiter Staff”. Therefore, there must be a clear way to distinguish between the two. Hence the implementation of 2 Junction tables that clarifies the staff members privileges. A “ManagementStaff” entity that contains two foreign keys a “restaurant\_restaurantid” & “staff\_staffid” and 1 primary key “managementstaffid” similarly the “Waiter” entities contain the same but a different primary key name, that being “waiterid”. Having these 2 entities allows to easily query data on staff members and their privileges. This also allows a great scale ability to the database model. Relationship between the two entities is also different. A Restaurant can have many waiters and there can also be multiple managers therefore we implemented that every Restaurant has 1 to many Managementstaff and where every restaurant can have 1 to many Waiters tending the franchise. With this relationship a staff member can be a manager to many restaurants.

### Issue 2: Table attributes duplicate values

Each “Table” (restaurant table) are described with the attributes “type of table” for the size of tables such as *2 ,4, 6 or 8 seater* tables and “location” such as *window or interior*. As these attributes are duplicated between tables, we can create two new table with an identifier each describing “type of table” and “location” respectively. By giving these attributes a table, we comply with the third normal form.

### Issue 3: Staff serving bookings

As a requirement from the case study, we must hold details about what staff member served a table. Therefore, an entity was created to hold this data. The “ServedBy” contains 2 Foreign keys, “booking\_bookingid” & “staff\_staffid”, then a primary key is created using both these foreign keys.

This booking then holds all data related to the table such as, location, time, number of guests, and customer who made the booking. This allows multiple staff members to tend to a table avoiding duplicating data save memory in the database meeting the requirements of the 3rd Normal Form.

### Issue 4: Table naming conflicts to SQL keywords

The name of the entities “Table” and “Management” are used in SQL keywords, this creates an error when the SQL statement is executed. Two solutions were considered to fix this issue. The first solution is to enclose the table names in `backticks`, this tells SQL that these are names and are not to be considered a part of the SQL syntax. The second solution is to rename these entity names to something that is not like SQL keywords. The second solution was picked as it is best for easier understanding of entity names and to make refactoring easier. We have renamed “Table” and “Management” to “Seating” and “ManagementStaff” respectively.

### Issue 5: Staff, Guest and Customer Tables

The three tables “Staff”, “Guest” and “Customer” each held common attributes between each other such as “Name”, “Phone Number” and “Email Address”. To comply with the 3rd normal form model, we created an extra entity called “Person” which contains the common attributes between the three entities. The attribute “StaffEmail” has been kept in the “Staff” entity as their work email and personal email can differ.

### Other issues

2 junction entities that detail the staff’s job was decided to be created to replace a single “TypeOfStaff” entity because there was a need to create a junction table to link staff and a restaurant anyways. It’s a relationship that allows us to easily identify the moderation status of each staff member while still complying to the 3rd normal form. It also allows ease of scalability for each restaurant.

The “Guest” entity within the case study specifies a “table number” as a foreign key linking between the “Guest” and booking, this has been changed to “Booking\_RestaurantBookingID” to be more consistent.

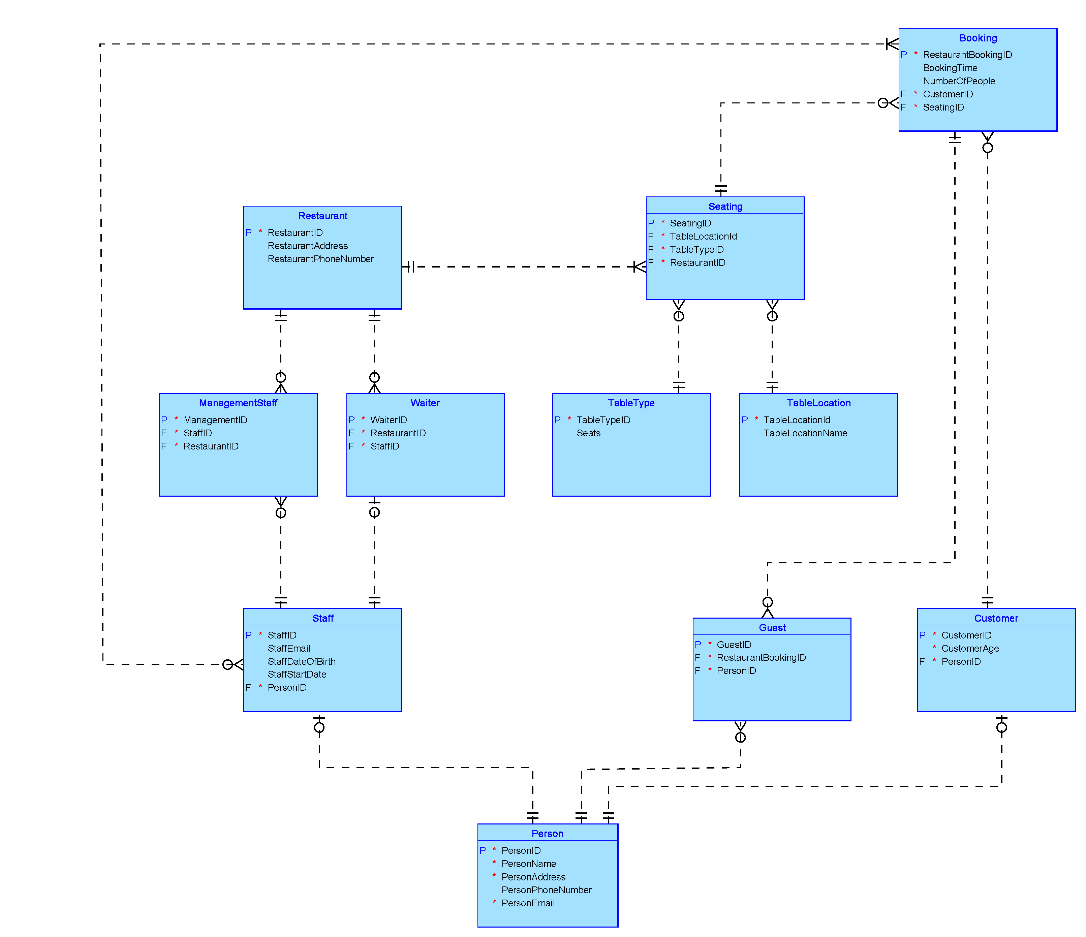
Creating a DML file a decision was made to opt out of adding the column names when Inserting data into the database, this was to reduce the horizontal screen real estate the query took. Instead, we opt for “INSERT INTO table\_name VALUES (x, y, z)”.

## Non-key constraints

|  |  |  |  |
| --- | --- | --- | --- |
| Constraint type | Table - field | Name given | Reason for introducing it |
| NOT NULL | Person - PersonName | personname\_nnull | Customer details must not be empty |
| NOT NULL | Person - PersonAddress | personaddress\_nnull | Customer details must not be empty |
| NOT NULL | Person - PersonEmail | personemail\_nnull | Customer details must not be empty |
| CHECK | Customer - CustomerAge | customerage\_chk | Customer age must be 18 or over |
| CHECK | Booking – NumberOfPeople | numberofpeople\_chk | A booking can only be made for up to 8 people |
| CHECK | Staff – StaffEmail | staffemail\_chk | All staff email must be the company domain ‘@burgershack.com’ |
| UNIQUE | Booking – (BookingTime, Seating) | booking\_bookingtime\_seating\_unq | Making sure 1 table can be taken at 1 Time |

# Present ERD

Screenshot of Logical Model



Screenshot of Physical Model

