

**DATABASE**

**SPECIFICATIONS**

*Next Gen Restaurant*

*Nick Smailer, nrs5434@psu.edu*

**School of Graduate Professional Studies**

Information Science Department

IN SC 521 - Introduction to Database Concepts

October 15, 2023

# Document Control

## Work carried out by:

|  |  |  |
| --- | --- | --- |
| **Name** | **Email Address** | **Other** |
| **Nick Smailer** | **nrs5434@psu.edu** |  |
|  |  |  |
|  |  |  |

## Revision Sheet

|  |  |  |
| --- | --- | --- |
| **Release No.** | **Date** | **Revision Description** |
| **1.0** | **8/29/2023** | Input my information and completed milestone 1 |
| **1.1** | **9/14/2023** | Edited milestone 1 and completed milestone 2 |
| **1.2** | **10/1/2023** | Edited the language of milestone 1, edited the ERD for milestone 2, edited assumptions and constraints for milestone 2, completed milestone 3. |
| **1.3** | **10/15/2023** | Revised my diagrams from Milestones 2 and 3, as well as functional dependencies from Milestone 3. Completed normalization to 3NF through decomposition for the tables in my diagram. |
| **1.4** | **10/29/2023** | Added SQL code for the creation of my tables along with the amount of records per table. I also added a visual of the physical model as it appears in SQL. |
| **1.5** | **11/29/2023** | Added queries to return useful information from the database, finalized document for final submission. |
|  |  |  |

**DATABASE SPECIFICATIONS**

**TABLE OF CONTENTS**

Document Control i

Work carried out by: i

Revision Sheet i

Milestone 1: Data Requirements 1

System Name or Title 1

Core requirements 1

Milestone 2: Conceptual Design 2

Diagram 2

Assumptions and Constraints 3

Milestone 3: Logical Design 4

Entity Relationship Diagram 4

Assumptions and Constraints 5

Milestone 4: Normalization and 12

Milestone 5: Physical Design 12

Assumptions and Constraints 12

Naming Conventions 12

Tables 13

**Examples of values** 13

**Notes** 13

Milestone 6: SQL queries 29

# Milestone 1: Data Requirements

## System Name or Title

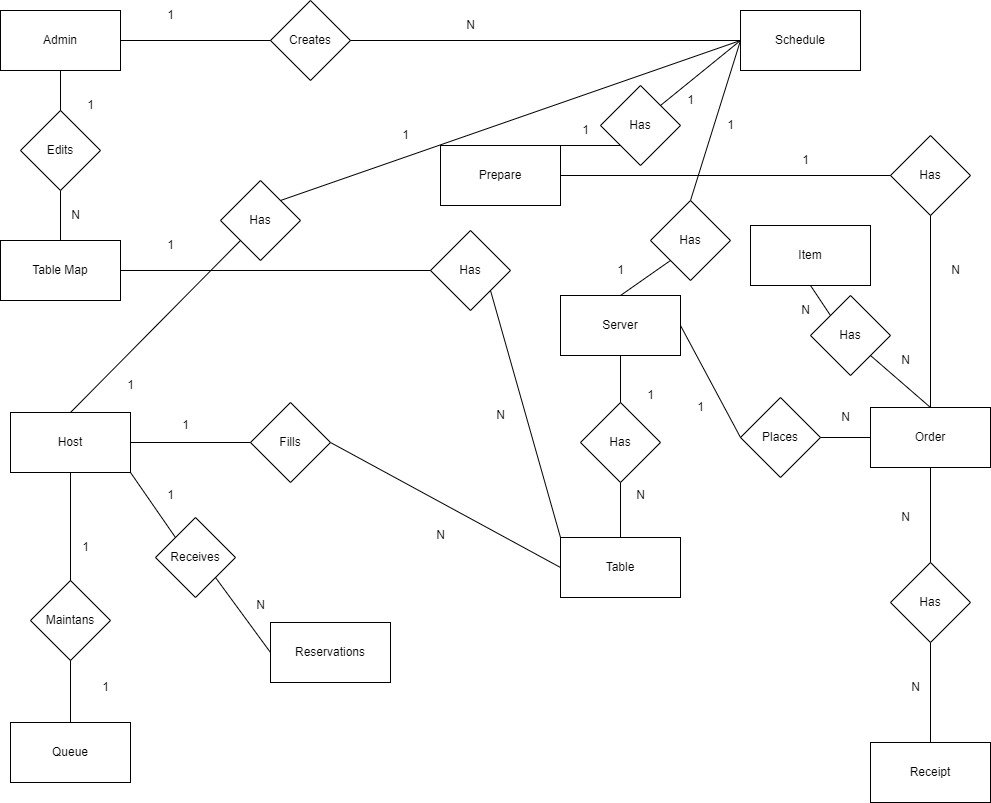
Next Gen Restaurant

## Core requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Requirement | Referenced page in SRS | Referenced Section in SRS | Referenced Paragraph in Section |
| 1 | Store a table map which only admins can make edits to. | 10 | 3.5.2 | 3.5.2.1 |
| 2 | Store the order of customers waiting for tables and store online reservations. | 10 | 3.5.3 | 3.5.3.1 |
| 3 | Store open table notifications for host | 11 | 3.5.3 | 3.5.3.5 |
| 4 | Store orders users enter into the system | 9 | 3.5.1 | 3.5.1.1 |
| 5 | Store checkout info (besides actual payment processing) | 9 | 3.5.1 | 3.5.1.4, 3.5.1.9 |
| 6 | Store preparation instructions for each item. | 9 | 3.5.1 | 3.5.1.2 |
| 7 | Store receipts for orders and store order history | 9 | 3.5.1 | 3.5.1.7, 3.5.1.8, 3.5.1.9 |
| 8 | Store schedules for users | 5 |  | 2 |
| 9 | Store default details for restaurant location, for example alcohol age requirements, sales tax, maximum bar tab, etc. | 8, 10 | 3.1, 3.5.1 | 2.4.1, 3.5.1.12 |
| 10 | Store KPIs for reporting and analytics. | 5 |  | 2 |

# Milestone 2: Conceptual Design

## Diagram



## Assumption & Constraints

To begin, the diagram above is in Chen notation. A few assumptions I made while creating the diagram above are as follows. First, to divide employees by role because different roles need access to different data, and one employee table connected to everything is messy. Since different users have different entities, I do not think there is a need for user authorization, as your user ID does that for you.

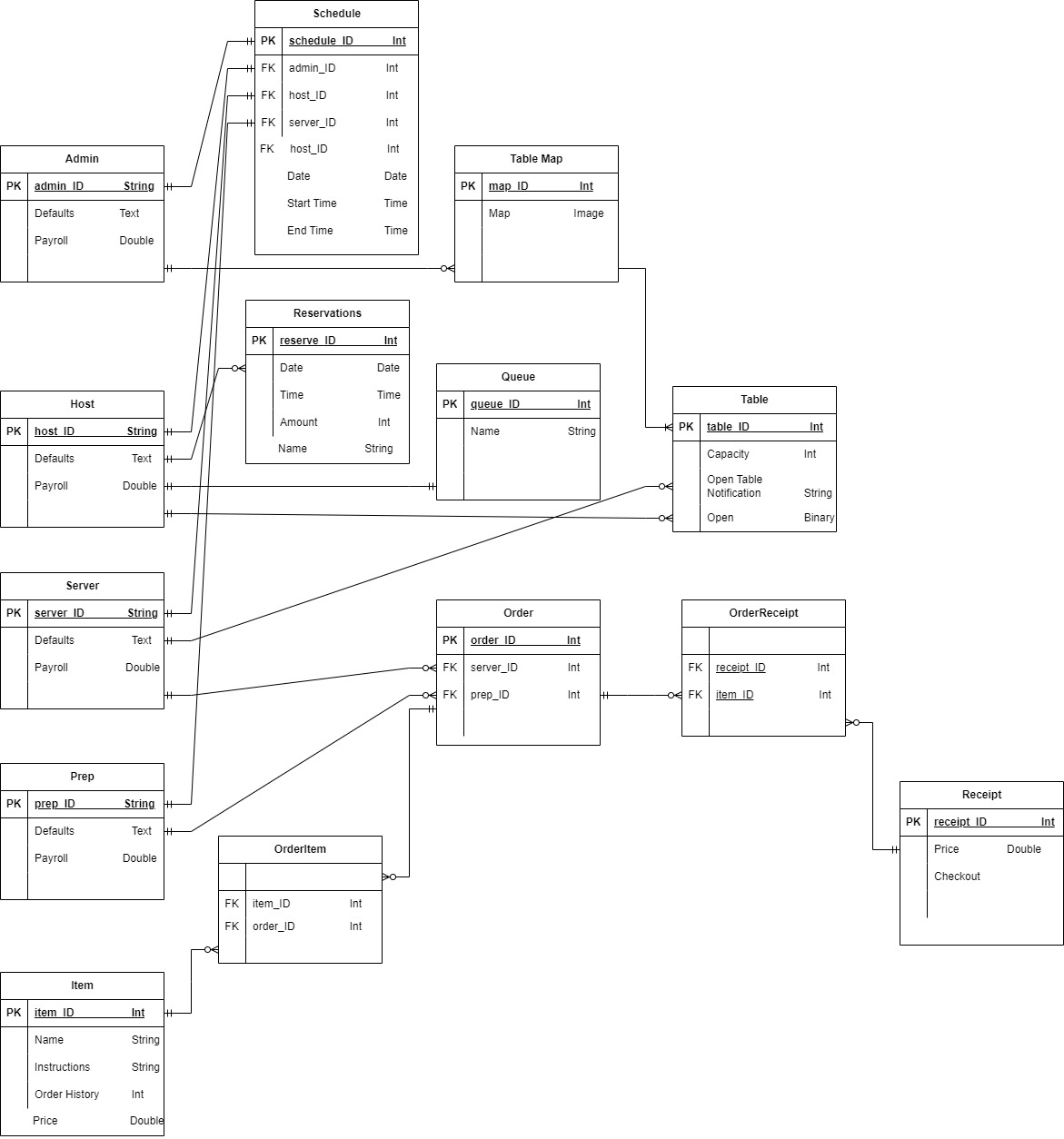
Next, there is no customer entity because the system does not need to store customer data. There is no way to retain/keep track of individual customers, and customers do not place orders, the servers will be the ones to enter them into the system thus making a customer entity unnecessary. Next, in the Table entity, table\_ID is an integer used to represent each table. This will be used to mark their unique locations on the map itself. While table\_ID is not stored in the Table Map entity, within the image that is stored for each map, the tables will have the table\_ID on them to illustrate where they should be placed. Next, Reservations and Queue are separate because Reservations have a higher priority than walk in customers, and by storing it this way I can create a unique record for each reservation containing date, time, and number of people. Instructions, order history, and reporting & analytics will all be stored on the order entity. The receipt entity contains checkout, as well. User defaults, containing information like minimum age to sell/consume alcohol, sales tax, minimum bar tab, etc. will be included as an attribute called “Defaults” given to all users. The relationship between order and receipts is many to many because many orders are on a receipt and there can be many receipts for an order (a table splits the bill).

The creation of the schedule entity added some complexity to my diagram, so I will explain. One admin creates many schedules. Each employee (host, server, prepare) has one schedule. Admins are able to manipulate the schedules of other users, regular users are not. While it looks a bit confusing currently, I believe the ERD in crows foot below will add some clarity visually. I will also explain the attributes more in depth below.

The main constraint I am worried about is whether or not I am getting to focused on the functionality of the system. Another constraint is the open table notification is currently only going to the host. The host can then inform the customer their table is ready, however the initial desire was to notify both the customer and the host. I am not sure exactly how a feature like this will work/what type of data it would be. I am thinking it is more of a function or feature than something that actually needs to be stored. Another difficulty I am running into is how and where to store reporting & analytics. This is a component that I want to include, and I think it may make sense to be its own table, as I cannot see it being atomic.

# Milestone 3: Logical Design

## Entity Relationship Diagram



## Assumptions and Constraints

## To begin, the ERD above is in crows foot notation. I had to adjust my conceptual model to include the entity created by the many to many relationships, so the shape is different from the conceptual model. The “Has” entity was created by the many to many relationships between Orders and Receipts. One thing I am still working out is scheduling. Under milestone 4 I go into more detail regarding this table.

A few assumptions are as follows. First, I assume defaults will store a long form text item containing different rules and laws. Next, the unique ID for every employee serves the same purpose as a user authorization, so user authorization is no longer required.

**Functional Dependencies:**

**Entity name:** Admin

**Attribute:** Initially, there were no attributes for Admin. Payroll, Defaults

**Keys:** None of the attributes above uniquely identify the others, so admin\_ID was created to be the primary key.

**Functional dependencies:** admin\_id 🡪 payroll, defaults

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Admin ID |  | Payroll, Defaults |

(admin\_ID)+=(admin\_ID, payroll, defaults)

**Entity name:** Host

**Attribute:** No descriptive attributes to begin with, Payroll and Defaults were added.

**Keys:** None of the attributes above uniquely identify the others, so host\_ID was created to be the primary key.

**Functional dependencies:** host\_ID 🡪 Payroll, Defaults

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Host ID |  | Payroll, Defaults |

(host\_ID) +=(host\_ID, payroll, defaults)

**Entity name:** Server

**Attribute:**  No descriptive attributes to begin with, Payroll and Defaults were added.

**Keys:** None of the attributes above uniquely identify the others, so server\_ID was created to be the primary key.

**Functional dependencies:** server\_ID 🡪 Payroll, Defaults

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Server ID |  | Payroll, Defaults |

(server\_ID) +=(server\_ID, payroll, defaults)

**Entity name:** Table

**Attribute:** Capacity, Open Table Notification and Open were added as attributes.

**Keys:** Since none of these can uniquely identify a table, table\_ID was created to do so.

**Functional dependencies:** table\_ID 🡪 Capacity, Open Table Notification, Open

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Table ID |  | Capacity, Open, Open Table Notification |

(table\_ID) +=(table\_ID, open, capacity, open table notification)

**Entity name:** Prep

**Attribute:** No descriptive attributes to begin with. Payroll and Defaults were added as attributes.

**Keys:** None of the attributes above uniquely identify the others, so Prepare\_ID was created to be the primary key.

**Functional dependencies:** Prep\_ID 🡪 Payroll, Defaults

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | prep ID |  | Payroll, Defaults |

(Prepare\_ID) +=(Prepare\_ID, payroll, defaults)

**Entity name:** Queue

**Attribute:** There were no attributes to begin with, so Walk in Order, Open Table Notification, and Online Reservation were added.

**Keys:** None of the attributes listed can uniquely identify the rest, so customer\_ID was created.

**Functional dependencies:** customer\_ID 🡪 Name

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Customer ID |  | Name |

(customer\_ID) +=(customer\_ID, Name)

**Entity name:** Item

**Attribute:** The following attributes were added to Item: Name, Price, Instructions, Order History

**Keys:** None of the listed attributes can be used to uniquely identify the rest, so order\_ID was created.

**Functional dependencies:** item\_id 🡪 Order Details, Order History, Price

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Item ID |  | Name, Instructions, Order History, Price |

(item\_ID) +=(item\_ID, name, instructions, order history, price)

**Entity name:** Receipt

**Attribute:** Price, Checkout, Items, customer\_ID, and order\_ID were added as attributes of Order.

**Keys:** customer\_ID and order\_ID are both foreign keys, receipt\_ID was added to identify the unique combination of all attributes

**Functional dependencies:** receipt\_ID 🡪 Price, Checkout. customer\_ID, item\_ID

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Receipt ID |  | Price, Checkout, Customer ID, Item ID |

(receipt\_ID) +=(receipt\_ID, customer\_ID, order\_ID, price, checkout)

**Entity name:** Table Map

**Attribute:** No descriptive attributes to begin with, Map was added as an attribute.

**Keys:** Map would be considered as a key, so map\_ID was created to be the key for this entity.

**Functional dependencies:** map\_ID 🡪 Map

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Map ID |  | Map |

(map\_ID) +=(map\_ID, map)

**Entity name:** OrderReceipt

**Attribute:** No descriptive attributes were provided, and none are required, but since this entity was created by the many to many relationship between order and receipts, both order\_ID and receipt\_ID are added as attributes.

**Keys:** Neither of the attributes are considered the key. Since this entity resulted from a conceptual relationship a key is not essential.

**Functional dependencies:** None

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
| Order\_ID, Receipt\_ID |  |  |  |

**Entity name:** Schedule

**Attribute:** None were provided initially, admin\_ID, host\_ID, server\_ID, prep\_ID, date, start time, and end time were added.

**Keys:** admin\_ID, host\_ID, server\_ID and prep\_ID are all foreign keys, and will be used in querying to return the individuals schedule, but they do not uniquely identify all other attributes, so schedule\_ID was created.

**Functional dependencies:** schedule\_ID 🡪 admin\_ID, host\_ID, server\_ID, prep\_ID, date, start time, end time

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Schedule\_ID |  | admin\_ID, host\_ID, server\_ID, prep\_ID, Date, Start Time, End Time |

(schedule\_ID) +=(schedule\_ID, admin\_ID, host\_ID, server\_ID, prep\_ID, date, start time, end time)

**Entity name:** Reservations

**Attribute:** No attributes were provided to begin with, date, time, amount, and name were created to describe the reservation.

**Keys:** None of the attributes above serve to uniquely identify the reservation, so reservation\_ID was created.

**Functional dependencies:**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | reserveID |  | Date, Time, Name, Amount |

(reserveID) +=(reserveID, date, time, name, amount)

**Entity name:** OrderItem

**Attribute:** No descriptive attributes were provided, and none are required, but since this entity was created by the many to many relationship between order and receipts, both order\_ID and item\_ID are added as attributes.

**Keys:** Neither of the attributes are considered the key. Since this entity resulted from a conceptual relationship a key is not essential.

**Functional dependencies:** None

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
| order\_ID, item\_ID |  |  |  |

**Entity name:** Order

**Attribute:** No descriptive attributes were provided initially, server­\_ID and prep\_ID were added to identify those who make and serve the order.

**Keys:** None of the attributes above can be used to uniquely identify an order, so order\_ID was created to be the key.

**Functional dependencies:** order\_ID 🡪 prep\_ID, server\_ID

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute not in FD | Attributes only on Left | Attributes in both | Attributes only on Right |
|  | Order\_ID |  | Prep\_ID, server\_ID |

(order\_ID) +=(order\_ID, prep\_ID, server\_ID)

# Milestone 4: Normalization AND

# Milestone 5: Physical Design

## Assumptions and Constraints

In the creation of my current model as well as the normalization below, I made a few assumptions. One assumption is that the attribute “Defaults”, which all users have, stores a document containing rules and regulations about the restaurant. On the “Table” table there are attributes called Open Table Notification and Open. These are to be updated by the server and host. When a host fills a table, Open becomes 1, when a server empties the table Open is changed to 0. Server should also send the Open Table Notification to the host after cleaning table.

A few constraints- one table that is a bit confusing in regards to both functional dependencies and normalization is Schedule. The primary key is schedule\_ID, and each user has their own value that increments daily storing an individual daily schedule. For example, one user may have values 1 – 3500 representing the possibility for about 10 years worth of unique schedules. The next user may have values 3501 – 7000 and so on. Foreign keys of admin\_ID, host\_ID, server\_ID, and prep\_ID are included in the Schedule entity, but not used to return the actual schedule. These are stored to show who’s schedule it is if another user were querying it. That is to say, the date and time are not functionally dependent on the user ID. Additionally, only one of those attributes will be filled per schedule, for example if it were a host’s schedule, host\_ID would have their respective ID, but the other ID attributes would be NULL. I am still thinking about and refining this entity, but currently this is how I am moving forward.

A few assumptions I am making are that checkout will be included in the payment processing which is not required by our system. The checkout information will be included is the price as well as the items within the order. Items can be queried using orderID or receiptID. I am also assuming users can use queries to create the KPIs mentioned in the requirements using the order history attribute.

Some of the tables have less than 20 records which I will now explain. Admin only has 6, this is because I do not think there should be a high number of administrators. The rest of the employee tables (Host, Server, and Prep) all have less than 20 as well, but the total number of employees in the system is over 40. Table Map has 5, this is because I did not think a restaurant would have 20 different table layouts options.

## Naming Conventions

Most of my names match what I called them in the previous milestones, but one notable difference is that in the physical implementation my primary keys appear as adminID while in the normalization they may appear as admin\_id. That applies to all my tables, I corrected some but not all. I also had to change the Table entity to Seats because Table and Tables were reserved words in Oracle. Additionally, the Queue entity was named Waiting in the physical implementation because Queue was a reserved word. Also, any attribute which had type specified as String became VARCHAR2. I believe I changed them, but the data type for the defaults stored by users was changed to type BLOB and the name of the attribute was changed to Rules. Finally, all attributes that had type Double became Number.

## Tables

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Admin | | | | |
|  | **Description** | An admin is one type of employee. This position is for users like managers who have different abilities than regular users within the system. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **Admin\_id** | ID of an admin, every user has a unique ID. | | String | a1298 | The letter a followed by 4 digits 1-10. |
|  | **Defaults** | Contains info like rules and regulations for the restaurant. | | Text | “Minimum age to consume alcohol: 21…” | This will be long form sentences stored together. |
|  | **Payroll** | The amount an individual is getting paid for the period. | | Double | 1,243.58 | Must be a positive, two decimal places. |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **admin­\_id → Defaults** | | | | |
|  | **Candidate keys** | **admin­\_id → Payroll** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | adminID | | | | |
|  | **Foreign Keys** | **-** | | | | |
|  | **SQL Code** | CREATE TABLE Admin (  adminID VARCHAR2(5) NOT NULL,  CONSTRAINT format\_check CHECK (REGEXP\_LIKE(adminID, '^a\d{4}$')),  Defaults BLOB,    Payroll NUMBER NOT NULL,  PRIMARY KEY (adminID)  ); | | | | |
|  | **Count of records in the table** | **6** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Host | | | | |
|  | **Description** | Host is another type of employee. This position is for users who will be servicing online reservations and walk in customers. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **host\_id** | ID of a host, every user has a unique ID. | | String | h5432 | The letter h followed by 4 digits 1-10. |
|  | **Defaults** | Contains info like rules and regulations for the restaurant. | | Text | “Minimum age to consume alcohol: 21” | This will be long form sentences stored together. |
|  | **Payroll** | The amount an individual is getting paid for the period. | | Double | 1,243.58 | Must be a positive, two decimal places. |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **hostID → Defaults**  **hostID → Payroll** | | | | |
|  | **Candidate keys** | hostID | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | hostID | | | | |
|  | **Foreign Keys** | **-** | | | | |
|  | **SQL Code** | CREATE TABLE Host (  hostID VARCHAR2(5) NOT NULL,  CONSTRAINT form\_check2 CHECK (REGEXP\_LIKE(hostID, '^h\d{4}$')),  Defaults BLOB,    Payroll NUMBER NOT NULL,  PRIMARY KEY (hostID)  ); | | | | |
|  | **Count of records in the table** | **9** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Server | | | | |
|  | **Description** | A server is another type of employee. This position is for users who service with the customers at tables. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **server\_id** | ID for a server, every user has a unique ID. | | String | s1234 | The letter s followed by 4 digits 1-10. |
|  | **Defaults** | Contains info like rules and regulations for the restaurant. | | Text | “Minimum age to consume alcohol: 21” | This will be long form sentences stored together. |
|  | **Payroll** | The amount an individual is getting paid for the period. | | Double | 1,243.58 | Must be a positive, two decimal places. |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **server\_id → Defaults**  **server\_id → Payroll** | | | | |
|  | **Candidate keys** | **server\_id** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique value | | | |
|  | **2NF** | **Yes** | A single attribute is the key for this table | | | |
|  | **3NF** | **Yes** | All non-prime attributes depend on primary key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | serverID | | | | |
|  | **Foreign Keys** | **-** | | | | |
|  | **SQL Code** | CREATE TABLE Server (  serverID VARCHAR2(5) NOT NULL,  CONSTRAINT form\_check3 CHECK (REGEXP\_LIKE(serverID, '^s\d{4}$')),  Defaults BLOB,    Payroll NUMBER NOT NULL,  PRIMARY KEY (serverID)  ); | | | | |
|  | **Count of records in the table** | **15** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Prep | | | | |
|  | **Description** | Prep is the final staff position. Prep encompasses all employees who fulfill customer orders. What this means is both kitchen staff and bar staff. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **prep\_id** | ID used for employees who prepare customer orders | | String | p9999 | The letter p followed by 4 digits 1-10. |
|  | **Defaults** | Contains info like rules and regulations for the restaurant. | | Text | “Minimum age to consume alcohol: 21” | This will be long form sentences stored together. |
|  | **Payroll** | The amount an individual is getting paid for the period. | | Double | 1,243.58 | Must be a positive, two decimal places. |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **prep\_id → Defaults**  **prep\_id → Payroll** | | | | |
|  | **Candidate keys** | prepID | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | prepID | | | | |
|  | **Foreign Keys** | **-** | | | | |
|  | **SQL Code** | CREATE TABLE Prep (  prepID VARCHAR2(5) NOT NULL,  CONSTRAINT form\_check4 CHECK (REGEXP\_LIKE(prepID, '^p\d{4}$')),  Defaults BLOB,    Payroll NUMBER NOT NULL,  PRIMARY KEY (prepID)  ); | | | | |
|  | **Count of records in the table** | **13** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Table Map | | | | |
|  | **Description** | Stores different maps of tables at the restaurant and data pertaining to them. Admin can swap between or customize maps within. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **mapID** | Uniquely identifies each different map option | | Integer | Any positive value | Cannot be null |
|  | **adminID** | Stores the admin who created specific table map | | Varchar | a1000 |  |
|  | **Map** | A picture of the table set up. | | Blob | Picture of table layout | Cannot be null |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **mapID → Map**  **mapID→ adminID** | | | | |
|  | **Candidate keys** | **mapID** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | mapID | | | | |
|  | **Foreign Keys** | adminID | | | | |
|  | **SQL Code** | CREATE TABLE Table\_Map (  mapID Integer NOT NULL,  CONSTRAINT m\_id CHECK (mapID > 0),  map\_pic BLOB,  adminID VARCHAR2(5),  FOREIGN KEY (adminID) REFERENCES Admin (adminID),  PRIMARY KEY (mapID)  ); | | | | |
|  | **Count of records in the table** | **5** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Waiting | | | | |
|  | **Description** | Stores order of walk-in customers who want a table. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **customerID** | Identifies each customer. Increments by 1 per customer. Represents their respective place in the queue. | | Integer | Any positive value | Cannot be null |
|  | **hostID** | Stores the host who helped the customer | | Varchar | h1000 |  |
|  | **Name** | Name of the customer | | Image | Picture of table layout | Cannot be null |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **customer\_ID → Name**  **customer\_ID → hostID** | | | | |
|  | **Candidate keys** | **customer\_id** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | customerID | | | | |
|  | **Foreign Keys** | hostID | | | | |
|  | **SQL Code** | CREATE TABLE Waiting (  customerID INTEGER NOT NULL,  CONSTRAINT c\_id CHECK (customerID > 0),  customer\_name VARCHAR(50),  hostID VARCHAR2(5),  FOREIGN KEY hostID REFERENCES Host (hostID),  PRIMARY KEY (customerID)  ); | | | | |
|  | **Count of records in the table** | **20** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Seats | | | | |
|  | **Description** | The actual tables in the restaurant. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **tableID** | Unique value assigned to each table in the restaurant | | Integer | Any value above 0 | Non null |
|  | **serverID** | Foreign key used to identify server who served table | | Varchar2 | s1000 |  |
|  | **Capacity** | Amount of people who can fit at the specific table | | Integer | 5 | Must be greater than 0 |
|  | **Open Table Notification** | Stores a message to be sent to host, sent by server when table is empty | | Varchar2 | “Table 10 is now open” | Host must then inform customer next in queue. |
|  | **Open** | Denotes whether or not a table is occupied | | Integer | 1 or 0 | 0 means empty, 1 means full |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **table\_id → capacity** | | | | |
|  | **Candidate keys** | **table\_id → open table notification** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | tableID | | | | |
|  | **Foreign Keys** | serverID | | | | |
|  | **SQL Code** | CREATE TABLE Seats (  tableID INTEGER NOT NULL,  CONSTRAINT t\_id CHECK (tableID > 0),  Max\_Amount INTEGER NOT NULL,  serverID VARCHAR2(5),  is\_Open NUMBER NOT NULL,  CONSTRAINT check\_Open CHECK (is\_Open IN (0, 1)),  Table\_Notif VARCHAR2(250),  FOREIGN KEY (serverID) REFERENCES Server (serverID),  PRIMARY KEY (tableID)  ); | | | | |
|  | **Count of records in the table** | **20** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Order | | | | |
|  | **Description** | This table stores customer orders as they are entered by servers | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **orderID** | Unique value assigned to each order placed in the system | | Integer | 985 | Cannot be null |
|  | **serverID** | ID of server who places and serves order | | Varchar2 | s1000 | Cannot be null |
|  | **prepID** | ID of employee who prepares the order | | Varchar2 | p1000 | Cannot be null |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **order\_id → server\_ID** | | | | |
|  | **Candidate keys** | **order\_id → prep\_ID** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | orderID | | | | |
|  | **Foreign Keys** | serverID, prepID | | | | |
|  | **SQL Code** | CREATE TABLE Orders (  orderID INTEGER NOT NULL,  CONSTRAINT o\_ID CHECK (orderID >0),  serverID VARCHAR2(5),  prepID VARCHAR2(5),  PRIMARY KEY (orderid),  FOREIGN KEY (serverID) REFERENCES Server (serverID),  FOREIGN KEY (prepID) REFERENCES Prep (prepID)  ); | | | | |
|  | **Count of records in the table** | **20** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Receipt | | | | |
|  | **Description** | Stores the total price of the order as well as checkout proceedings minus the actual payment processing. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **receipt\_id** | Stores each receipt using a unique value | | Integer | Between 1 and 999999999 | Non null |
|  | **price** | Stores the total price of the order by totaling items | | Double | 67.88 | Totals prices from items\_IDs |
|  | **checkout** |  | |  |  | This attribute needs further development |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | **receipt\_id → price** | | | | |
|  | **Candidate keys** | **receipt\_id → checkout** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | receiptID | | | | |
|  | **Foreign Keys** | **-** | | | | |
|  | **SQL Code** | CREATE TABLE Receipt (  receiptID INTEGER NOT NULL,  CONSTRAINT r\_check CHECK (receiptID > 0),  Price NUMBER NOT NULL,  PRIMARY KEY (receiptID)  ); | | | | |
|  | **Count of records in the table** | **20** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Schedule | | | | |
|  | **Description** | This table stores schedule data for each individual employee | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **schedule\_ID** | Unique schedule ID. Each employee has their own which increments daily. | | Integer | Anything above 0 | Cannot be null |
|  | **admin\_ID** | If employee in question is admin, this field will store their ID, blank otherwise. | | Integer | Must be valid admin\_ID | 0 if user is not admin |
|  | **prep\_ID** | If employee in question is prep, this field will store their ID, blank otherwise. | | Integer | Must be valid prep\_ID | 0 if user is not prep |
|  | **server\_ID** | If employee in question is server, this field will store their ID, blank otherwise. | | Integer | Must be valid server\_ID | 0 if user is not server |
|  | **Date** | One date | | Date | 10/10/23 |  |
|  | **Start Time** | Time the shift begins | | Time | 1:00 PM |  |
|  | **End Time** | Time the shift ends | | Time | 8:00 PM |  |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | Schedule\_ID → admin\_ID  Schedule\_ID → host\_ID  Schedule\_ID → server\_ID  Schedule\_ID → Start Time  Schedule\_ID → End Time | | | | |
|  | **Candidate keys** | **scheduleID** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | Every cell contains a unique value | | | |
|  | **2NF** | **Yes** | One attribute is the key for this table | | | |
|  | **3NF** | **Yes** | All non-prime attributes depend on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | scheduleID | | | | |
|  | **Foreign Keys** | adminID, hostID, serverID, prepID | | | | |
|  | **SQL Code** | CREATE TABLE SCHEDULE  (  SCHEDULEID NUMBER(\*, 0) NOT NULL,  ADMINID VARCHAR2(5 BYTE),  HOSTID VARCHAR2(5 BYTE),  SERVERID VARCHAR2(5 BYTE),  PREPID VARCHAR2(5 BYTE),  DATE\_START TIMESTAMP(6) NOT NULL,  DATE\_END TIMESTAMP(6) NOT NULL,  PRIMARY KEY (scheduleID) | | | | |
|  | **Count of records in the table** | **20** | | | | |

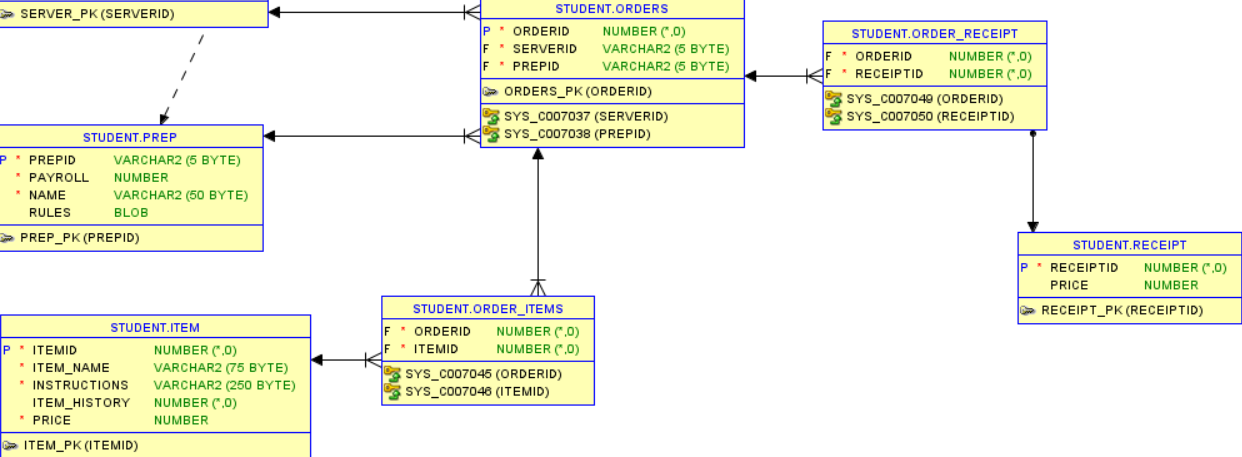
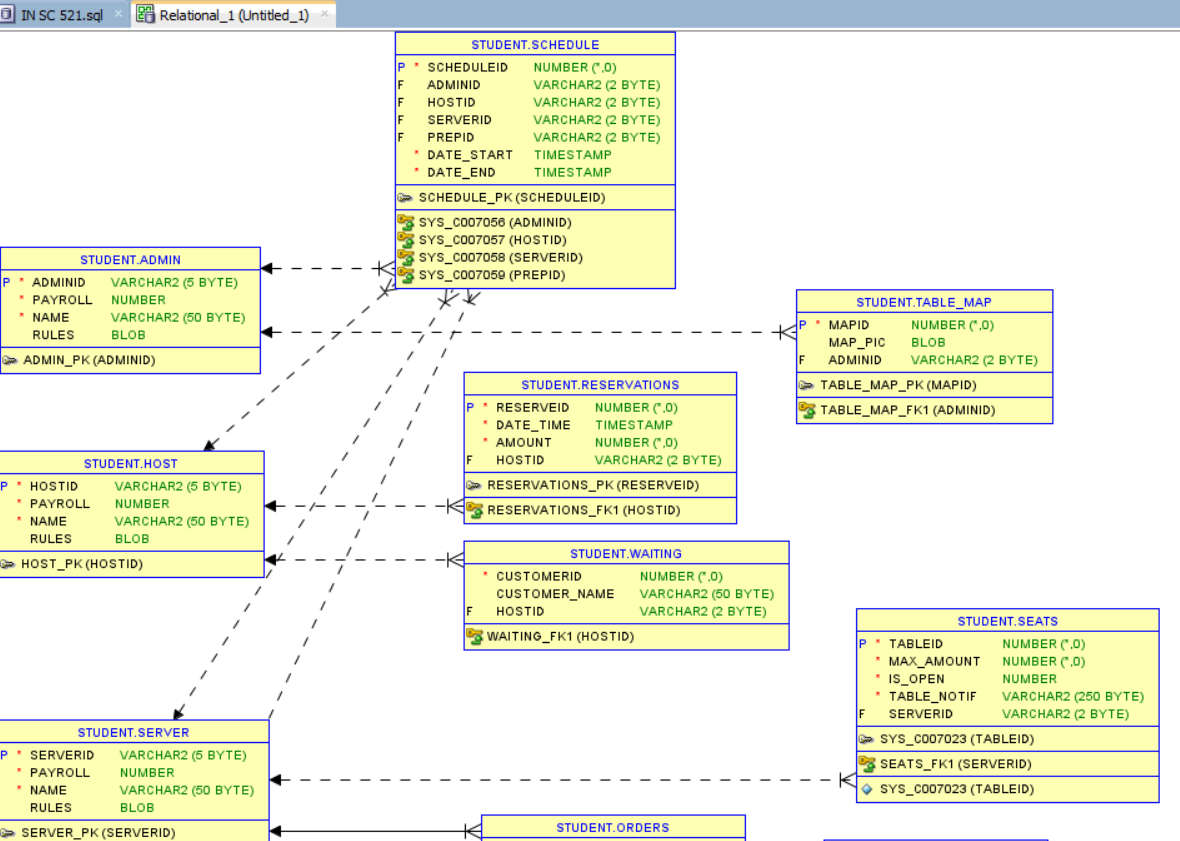
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Reservations | | | | |
|  | **Description** | This table stores the online reservation data. | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **reserveID** | Unique value assigned to an online reservation used to identify it | | Integer | Must be greater than 1, never repeated | Cannot be null |
|  | **Name** | Name of party making reservation | | String | John | Cannot be null |
|  | **Amount** | Number of people in the reservation | | Integer | 10 | Cannot be null |
|  | **Date** | Date of the reservation | | Date | 10/10/2023 | Cannot be null |
|  | **hostID** | Foreign key used to track who seats the customer | | Varchar | h1000 |  |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | reserveID → name  reserveID → amount  reserveID → date  reserveID → time  reserveID → hostID | | | | |
|  | **Candidate keys** | **reserveID** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is a single attribute | | | |
|  | **3NF** | **Yes** | All the non-prime attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | reserveID | | | | |
|  | **Foreign Keys** | hostID | | | | |
|  | **SQL Code** | CREATE TABLE Reservations (  reserveID INTEGER NOT NULL,  date\_time TIMESTAMP NOT NULL,  Amount INTEGER NOT NULL,  hostID VARCHAR2(5),  FOREIGN KEY  PRIMARY KEY (reserveID)  ); | | | | |
|  | **Count of records in the table** | **20** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | Item | | | | |
|  | **Description** | This table stores the items that can be ordered in the system | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **item\_ID** | Unique value assigned to a menu item used to identify it | | Integer | Must be greater than 1, never repeated | Cannot be null |
|  | **Name** | Name of the item | | String | Cheeseburger | Cannot be null |
|  | **Price** | How much the item costs | | Double | 10.99 | Cannot be null |
|  | **Instructions** | Step by step guide to make the item | | String | “First get the bun….” | Cannot be null |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | item\_id → name  item\_id → price  item\_id → instructions  item\_id → order history | | | | |
|  | **Candidate keys** | **Item\_id** | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The key for this table is the single attribute item\_ID | | | |
|  | **3NF** | **Yes** | All the non-key attributes depend on the key and only on the key | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | itemID | | | | |
|  | **Foreign Keys** | **-** | | | | |
|  | **SQL Code** | CREATE TABLE Item (  itemID INTEGER NOT NULL,  CONSTRAINT i\_ID CHECK (itemID >0),  item\_name VARCHAR(75),  Instructions VARCHAR(250),  item\_history INTEGER,  Price NUMBER,  PRIMARY KEY (itemID)  ); | | | | |
|  | **Count of records in the table** | **20** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | OrderItems | | | | |
|  | **Description** | This table links the Order and Items tables because of their many to many relationship | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **Order\_ID** | Unique value used to identify specific order | | Integer | 2 | Cannot be null |
|  | **Item\_ID** | Unique value used to identify specific item | | Integer | 1001 | Cannot be null |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | CREATE TABLE order\_receipt (  orderID INTEGER NOT NULL,  itemID INTEGER NOT NULL,  FOREIGN KEY (orderID) REFERENCES Orders,  FOREIGN KEY (itemID) REFERENCES Item  ); | | | | |
|  | **Candidate keys** | None | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The two foreign keys make up a composite key | | | |
|  | **3NF** | **Yes** | No transitive dependency since table is only made up of foreign keys | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | None | | | | |
|  | **Foreign Keys** | orderID, itemID | | | | |
|  | **SQL Code** | CREATE TABLE ORDER\_ITEMS  (  ORDERID NUMBER(\*, 0) NOT NULL,  ITEMID NUMBER(\*, 0) NOT NULL  ) | | | | |
|  | **Count of records in the table** | **48** | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Name of the table*** | OrderReceipt | | | | |
|  | **Description** | This table links the order and receipt tables because of their many to many relationship | | | | |
|  | **Attribute** | **Description** | | **Type** | **Examples of values** | **Notes** |
|  | **Order\_ID** | Unique value assigned to an order | | Integer | 12 | Cannot be null |
|  | **Receipt\_ID** | Unique value assigned to a receupt | | Integer | 888 | Cannot be null |
|  | **Functional Dependencies and Keys** | | | | | |
|  | **Functional dependencies** | None | | | | |
|  | **Candidate keys** | None | | | | |
|  | **Normalization** | | | | | |
|  | **1NF** | **Yes** | All cells contain a unique atomic value | | | |
|  | **2NF** | **Yes** | The two foreign keys make up a composite key | | | |
|  | **3NF** | **Yes** | No transitive dependency since table is only made up of foreign keys | | | |
|  | **Physical Design** | | | | | |
|  | **Primary Key** | None | | | | |
|  | **Foreign Keys** | orderID, receiptID | | | | |
|  | **SQL Code** | CREATE TABLE ORDER\_RECEIPT  (  ORDERID NUMBER(\*, 0) NOT NULL,  RECEIPTID NUMBER(\*, 0) NOT NULL  ) | | | | |
|  | **Count of records in the table** | **22** | | | | |

Below are images of my physical model. I used two pictures to ensure the clarity of each table in the image.



# Milestone 6: SQL queries and

**Note**: I know the assignment called for 7 queries, I created 9 just because I felt like I needed a couple more to fully include all the operators required and return a variety of information.

|  |  |
| --- | --- |
| **Query 1** |  |
| **English version** | Return an individual’s schedules using their user ID |
| **Source for the query need in the SRS document** | SRS Document page 5, paragraph 2 |
| **SQL sentence** | SELECT  date\_start, date\_end  FROM  schedule  WHERE  adminid = 'a1000' |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 2** |  |
| **English version** | Return items ordered using the receipt |
| **Source for the query need in the SRS document** | SRS Document page 9, section 3.5.1, paragraphs 3.5.1.7-9 |
| **SQL sentence** | SELECT  item\_name  FROM  item  JOIN order\_items ON item.itemid = order\_items.itemid  WHERE  orderid = 1 |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 3** |  |
| **English version** | Return order totals for checkout, filtered by amount in this query |
| **Source for the query need in the SRS document** | SRS Document page 9, section 3.5.1, paragraph 3.5.1.1 |
| **SQL sentence** | SELECT  order\_receipt.orderid,  SUM(item.price) AS total\_price  FROM  item  JOIN  order\_items ON item.itemid = order\_items.itemid  JOIN  order\_receipt ON order\_items.orderid = order\_receipt.orderid  JOIN  receipt ON order\_receipt.receiptid = receipt.receiptid  GROUP BY  order\_receipt.orderid  HAVING  SUM(item.price) > 100; |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 4** |  |
| **English version** | Return orders entered by a specific user |
| **Source for the query need in the SRS document** | SRS Document page 9, section 3.5.1, paragraph 3.5.1.1 |
| **SQL sentence** | SELECT  item\_name  FROM  item  JOIN  order\_items ON item.itemid = order\_items.itemid  JOIN  orders ON order\_items.orderid = orders.orderid  WHERE  serverid = 's1000' |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 5** |  |
| **English version** | Return the server ID and prep ID for a specific order |
| **Source for the query need in the SRS document** | SRS Document page 9, section 3.5.1, paragraph 3.5.1.1 |
| **SQL sentence** | SELECT  serverid, prepid  FROM  orders  WHERE  orderid = 3; |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 6** |  |
| **English version** | Return date/time for reservations, searched by a variety of names using union in case of a situation where the customer forgot whose name the reservation is under |
| **Source for the query need in the SRS document** | SRS Document page 10, section 3.5.3, paragraph 3.5.3.1 |
| **SQL sentence** | SELECT  reserve\_name, date\_time  FROM  reservations  WHERE  (reserve\_name LIKE 'Jo%' OR reserve\_name LIKE 'Ry%'); |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 7** |  |
| **English version** | Return the amount of people in a reservation and the amount of seats at a table |
| **Source for the query need in the SRS document** | I don’t think this is specifically mentioned in the document, just something I thought would be useful and a good example of using UNION |
| **SQL sentence** | SELECT  amount  FROM  reservations  WHERE  reserveid = 1  UNION  SELECT  max\_amount  FROM  seats  WHERE  tableid = 1; |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 8** |  |
| **English version** | Return order instructions for an item |
| **Source for the query need in the SRS document** | SRS Document page 9, section 3.5.1, paragraph 3.5.1.2 |
| **SQL sentence** | SELECT  item\_name, instructions  FROM  item; |
| **Example of returned rows (cropped screen caption)** |  |

|  |  |
| --- | --- |
| **Query 9** |  |
| **English version** | Return all admin schedules at once |
| **Source for the query need in the SRS document** | SRS Document page 5, paragraph 2 |
| **SQL sentence** | SELECT  date\_start, date\_end, adminid  FROM  schedule  WHERE  adminid IN (SELECT adminid FROM admin); |
| **Example of returned rows (cropped screen caption)** |  |