Adeeb Shaban; 18110018

Database

Table of Contents

[Scenario: 2](#_Toc30526055)

[Business requirements: 2](#_Toc30526056)

[Employees: 3](#_Toc30526057)

[Guitars: 3](#_Toc30526058)

[Invoices 3](#_Toc30526059)

[Customers: 3](#_Toc30526060)

[Representing the database: 4](#_Toc30526061)

[ER Diagram: 4](#_Toc30526062)

[Mapping: 5](#_Toc30526063)

[Normalization: 6](#_Toc30526064)

[Physical Representation 6](#_Toc30526065)

[User privileges: 8](#_Toc30526066)

[Manager: 9](#_Toc30526067)

[Employees: 9](#_Toc30526068)

# Scenario:

Working as an assistant Database Developer for a large IT consultancy company. The company has been approached by a guitar store to expand to the growth for their products and services. And implementing a database for the store to record their data and analyze it. The guitar store will have all kinds of guitars (brands), the model of the guitar, etc. Guitar stores that has databases will grow further and will be managed more successfully.

# Business requirements:

First of all, we need to know what business requirements we need to start and building the system. Talking about the business requirements, in our case (guitar center) we need to look in deep details about **Employees, Guitars, Brands, Customers**. In another words, we have requirements to start building the database system as in we need employees to work in the guitar center, we need to know their positions e.g. sellers(finance), maintenance, etc. and guitars have different brands and models for each brand. And of course, the costumers that will buy the guitars]

.

**To start planning the scenario and the business requirements**, here are the scenario and the business requirements:

1. we have a guitar store, the guitar store has an **ID, name.**
2. We have a guitar store; the guitar store has many employees. Each employee has some attributes (**employee id**, **first name**, **last name, DOB (Date of Birth), one phone number, email, position** **of employee.**
3. the guitar store has many guitars, and they are split into two categories the owner needs to know about both types the guitars that are sold and guitars that are currently in store, each of those categories has an **ID**,**, price, brand ,the only difference if the guitar is sold the owner will also need to know about the invoice number.**
4. We have brands, each guitar has one brand, brands can contain more than one guitar, each brand has a **brand ID, brand name.**
5. the guitar store has many customers, each customer has a **customer id**, **first name, last name, DOB (Date of Birth), email and one phone number.**
6. We have invoices, customers could have many invoices, but each invoice has one customer, the owner wants to know the invoices number, its **ID, and the customer that it belongs to and one employee that prints it. But one employee can print more than one invoice.**

**NOTE:** But one guitar brand has many guitars. But one guitar doesn’t have many brands.

## Employees:

Employees are needed because they will be available in the store and they work as a finance. We need some details about these employees such as their **ID** for every employee, **first name**, **last name, DOB (Date of Birth), phone number (one phone number only), email, position**.

## Guitars:

Guitars are the products that will be on sale and that customers will be buying. We need some details about the guitars such as their **ID**, **brand** *(e.g. Fender).*

## Invoices

Invoices is needed because it’s the communication between the employee and the customer as in the employee will print what the guitar/guitars the customer bought, so the invoices will contain **invoice id, invoice number and invoice price.**

## Customers:

Customers is very important to be in the system, because they are the factor that buy all these listed items, and will be communicating with the employees to buy, maintain and get information. We should consider some details for customers such as: **ID** for each customer, **First name, last name,** their **email, DOB (Date of Birth)** and **one phone number.**

# Representing the database:

After determining the business requirements, we have some steps to do to start mapping and building the system. The step for determining the database business requirements is just the identification of what we will use and implement on the system, the next step is planning and analyzing these requirements.

## ER Diagram:

Entity Relation Diagram identified the relationship between all entitles and types of relationship e.g. one-many, many-many, etc.

**A close up of a logo

Description automatically generated**

## Mapping:

After knowing the relationship between entities and drawing the ER diagram, we have known all relations and how they are related to each other, we will start by a step called **mapping** which will help to build a database (being as reference) for all tables and attributes.

**Guitar\_store** (guitar\_store\_id, store\_name)

**In\_store\_guitar** (instore\_guitar\_id, guitar\_store\_id, brand\_id, price)

**Sold\_guitar** (sold\_guitar\_id, invoice\_id, guitar\_store\_id, brand\_id, price)

**Brand** (brand\_id, brand\_name, guitar\_store\_id)

**Customers** (customer\_id, customer\_first\_name, customer\_last\_name, customer\_email, customer\_DOB, customer\_phone\_number, guitar\_store\_id).

**Employee** (emp\_id, emp\_first\_name, emp\_last\_name, emp\_email, emp\_DOB, emp\_phone\_number, guitar\_store\_id).

**Invoice** (invoice\_id, emp\_id, customer\_id, invoice\_number, invoice\_price)

## Normalization:

There are three normalization rules that should be consider while doing the mapping and ERD.

**The first normal form** prohibits the repeating group of columns. Country of guitar was repeated so I made a table called country, **the second normal form** is the dependency on the primary key and the removal of partial dependencies while not needed. **The third normal form is that the non-key** columns are independent of each other. the three normalization steps didn’t affect my ERD because the planning was accurate.

As we know, when we have two tables and the relation is one-one or one-many or many-one, we can do a relationship between them by adding the one-many (primary key>> table) by adding a the primary key to the next table as a foreign key. Or many to one or one to one. But when we have a many to many relationships, we need make a new table to be between those many to many tables to make a relationship.

## Physical Representation

In the physical representation, I will be showing how the attributes of each entity (table) is represented in the physical way.

|  |  |
| --- | --- |
| Guitar\_store (table) | |
| guitar\_store\_id | **store\_name** | |
| Int (10) | Varchar (50) | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Employee (table) | | | | | |
| emp\_id | **emp\_first\_name** | | **emp\_last\_name** | **emp\_DOB** | **emp\_phone\_number** | **emp\_email** | **guitar\_store\_id** |
| Int (10) | Char (50) | | Char (50) | DATE | Varchar (50) | Varchar (50) | Int (10) |

|  |  |  |  |
| --- | --- | --- | --- |
| In\_store\_guitar (table) | | | |
| instore\_guitar\_id | Guitar\_Store\_id | brand\_id | Price |
| Int (10) | Int (10) | Int (10) | Decimal (13,2) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sold\_guitar (table) | | | | |
| sold\_guitar\_id | **invoice\_id** | **Guitar\_store\_id** | **Brand\_id** | **Price** |
| Int (10) | Int (10) | Int (10) | Int (10) | Decimal (13,2) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Customer (table) | | | | | |
| customer\_id | **customer\_first\_name** | **customer\_last\_name** | **customer\_DOB** | **customer\_phone\_number** | **customer\_email** | **guitar\_store\_id** |
| Int (10) | Char (50) | Char (50) | DATE | Varchar (50) | Varchar (50) | Int (10) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Invoice(table) | | | | |
| invoice\_id | **emp\_id** | **customer\_id** | **invoice\_number** | **invoice\_price** |
| Int (10) | Int (10) | Int (10) | Int (10) | Decimal (13,2) |

|  |  |  |
| --- | --- | --- |
| Brand (table) | | |
| brand\_id | **brand\_name** | **Guitar\_store\_id** |
| Int (10) | Char (50) | Int (10) |

## User privileges:

This step is about the security of the system (database) so non authorized people can’t alter (edit) anything in the database. This step is setting access to some certain people to add some certain data in the database. You can also set the access to view so the user can only view what you have determine for them to view. Setting privileges have many types to different users. For example, you want for the employees to only add some data in certain tables, you will set access for them to only do that. In our scenario we need the employees to add or delete purchased guitars and accessories, so the database will be up to date and simple.

When setting user privileges, we need to know what users are going to use the database to determine their privileges and what they will be having access to Here are the tables that show how the privileges will be divided:

### Manager:

The manager should have all the access to the database so he can manage it to its best.

|  |  |  |
| --- | --- | --- |
| User | Privileges | |
| Manager | **Entities** | **Access** |
| All entities | Full Access |
| Full Access |
| Full Access |
| Full Access |

As shown above, the manager has all access to the database (all tables).

### Employees:

#### Finance/customer service:

finance service will handle the customers while buying the guitar.

Here is the table of privileges for the finance employees

|  |  |  |
| --- | --- | --- |
| User | Privileges | |
| Finance | **Entities** | **Access** |
|
| Invoice | Insert, Delete, view. |
| Country | Insert, Delete, view. |
| Customer | Insert, Delete, view. |
| In store guitar | Insert, Delete, view. |
| Brand | Insert, Delete, view. |
| All other tables | No access |

As we can see the Finance will only has the access to the guitar, accessories table for inserting rows if some new guitars have arrived at the store.

Database Testing:

In this step we will test the business requirements through users and technical sides:

A screenshot of a computer screen

Description automatically generated

Here is the testing of the tables of the database

A screenshot of a computer

Description automatically generated

Here’s the customer table data

A screenshot of a computer

Description automatically generated

Here’s the brand table data

A screenshot of a computer

Description automatically generated

Here’s the employee table data

A screenshot of a computer

Description automatically generated

Here’s the store data

A screenshot of a computer screen

Description automatically generated

Here’s the instore table data

Database Maintenance:

One of its biggest advantages is that it is the quickest backup technique to repair the information; because the entirety is saved in one document. However, this technique will take more than the distance it desires and will feel the enterprise double the numerous for storage. Also, this technique requires quite a few time to back up the records. Therefore, we notion that this approach will in no way work in Miles; because they're nonetheless new and don’t have the budget to shop for database storages, when there are other beneficial methods. Although incremental backups run faster and takes much less garage than the overall backup, their healing method is complicated. If the backup runs on Saturday and a report is then updated Monday morning, should something show up to that record on Tuesday, one would need to get admission to the Monday nighttime backup to restore it. That why we’ve looked for every other backup approach one that has much less complicated recovery and has the same backup process as the incremental backup. This is wherein the differential backup comes in. This type of backup is similar to the incremental backup technique, in which its backups the information over the weekend doesn’t backup the information that has been already sponsored up or not updated, but this technique’s restoration procedure manner less complicated than the incremental. Differential backups don’t clean the archive bit. So, a record that is up to date after a ordinary backup might be archived each time a differential backup is run till the next normal backup runs and clears the archive bit.