FoxCore Retail Database Design Project

Group 4

Vinutha Chelamallu

Hsiao Tien (Tammy) Chow

Somya Garg

Ta-Chin (Meek) Ho

Sai Pradeep Vallabhaneni

Stuart School of Business, Illinois Institute of Technology

MAX 506: Database Design and SQL

Dr. Dinakar Jayarajan

April 25, 2024

FoxCore Retail Database Design Report

In this report, we detail our creation and implementation of a database for FoxCore Retail - A company that specializes in selling unique items at events. The primary objective was to streamline their data collection and analysis process by replacing their current paper-based system. We discuss the design process, including the conceptual data model and the logical relational database model. Furthermore, we provide the necessary Data Definition Language (DDL) statements to establish the database tables, Data Manipulation Language (DML) statements for modifying and populating the database, and sample queries for data retrieval and analysis. The database structure is designed to optimize data storage, retrieval, and analysis, which will be helpful for sales tracking, inventory management, and event performance evaluation.

Conceptual Database Design

The conceptual data model is a foundation for the FoxCore Retail database, providing a holistic view of the entities and their interconnections. This model empowers FoxCore Retail to make informed decisions that optimize its business operations and bolster profitability. The following details the identified entities, their attributes, and the cardinalities of their relationships:

Entities and Attributes:

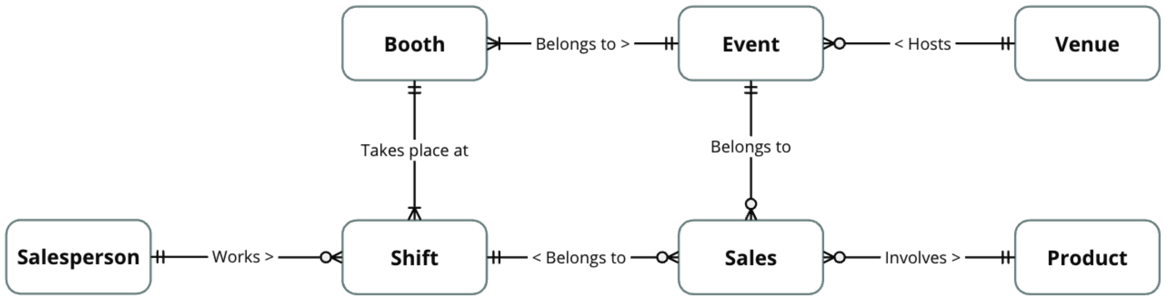
- **Event:** Details about sales events are recorded. Attributes include EventID (primary key), VenueID (foreign key referencing Venue Table), EventName, StartDate, EndDate, Description, and EventType (categorical with validations).
- **Venue:** Stores information about the locations where events are held. Attributes include VenueID (primary key), VenueName, StreetAddress, City, State, PostalCode, and Description.
- **Booth:** This entity captures details about the individual booths set up at events. Attributes include BoothID (primary key), EventID (foreign key referencing Event table), and Location.
- **Product:** Represents the various products sold by FoxCore Retail. Attributes include ProductID (primary key), ProductName, WholesaleCode, and MinSellingPrice.
- **Salesperson:** Stores information about the salespeople who work at events. Attributes include SalespersonID (primary key), FirstName, LastName, StreetAddress, City, State, PostalCode, and PhoneNumber (unique).
- **Sales:** Captures details about individual sales transactions. Attributes include SalesId (primary key), EventID (foreign key Event table), ShiftID (foreign key referencing Shift table), ProductID (foreign key referencing Product table), QuantitySold, SellingPrice, and SalesDate.

- **Shift:** Represents a salesperson's work schedule for a specific booth during an event. Attributes include ShiftID (primary key), SalespersonID (foreign key referencing Salesperson table), BoothID (foreign key referencing Booth table), StartTime, and EndTime.

Cardinalities and Relationships:
- A Venue can host multiple Events (one-to-many).
- An Event takes place at a single Venue (many-to-one).
- An Event can have various Booths (one-to-many).
- A Booth belongs to a single Event (many-to-one).
- A Product can be sold in multiple Sales transactions (one-to-many).
- A Sales transaction involves a single Product (many-to-one).
- A Salesperson can work multiple Shifts (one-to-many).
- A Shift is assigned to a single Salesperson (many-to-one).
- A Shift takes place at a single Booth (one-to-many).
- A Booth can have multiple Shifts (one-to-many).
- A Sales transaction belongs to a single Event (one-to-many).
- An Event can have various Sales transactions (many-to-one).
- A Sales transaction belongs to a single Shift (one-to-many).
- A Shift can have multiple Sales transactions (many-to-one).

Entity Relationship Diagram (ERD):

The following Entity-Relationship (ER) Diagram depicts this conceptual data model visually. The rectangles represent the entities, and the lines showcase the cardinalities of their relationships. This ER Diagram serves as a blueprint for the relational database design, ensuring data integrity and minimizing redundancy.



Logical (Relational) Database Model

The logical relational database model serves as a critical bridge between the conceptual data model and the physical implementation of the database. It translates the high-level entities and relationships identified in the conceptual model into a structure of well-defined tables with distinct columns and constraints. These tables and their interconnections form a comprehensive network for storing, organizing, and managing data efficiently.

Normalization and Data Integrity

The tables within this logical model adhere to normalization principles, likely following the Third Normal Form (3NF) or a higher level. Normalization minimizes data redundancy and ensures data integrity by eliminating the possibility of anomalies (errors) that can occur when data is inserted, deleted, or updated. Standard SQL data types, such as VARCHAR, DATE, and DECIMAL, are used throughout the tables to define the format and characteristics of the data stored within each column.

Normalized Schema Representation

This refined Entity-Relationship Diagram depicts a nromalized structure of the database. It incorporqates all the essential entities, their attributes, primary keys, foreign keys, and the cardinalities of the relationships between them. It also emphasizes the relationships between the entities, providing a clear understanding of how data elements interact within the database model.

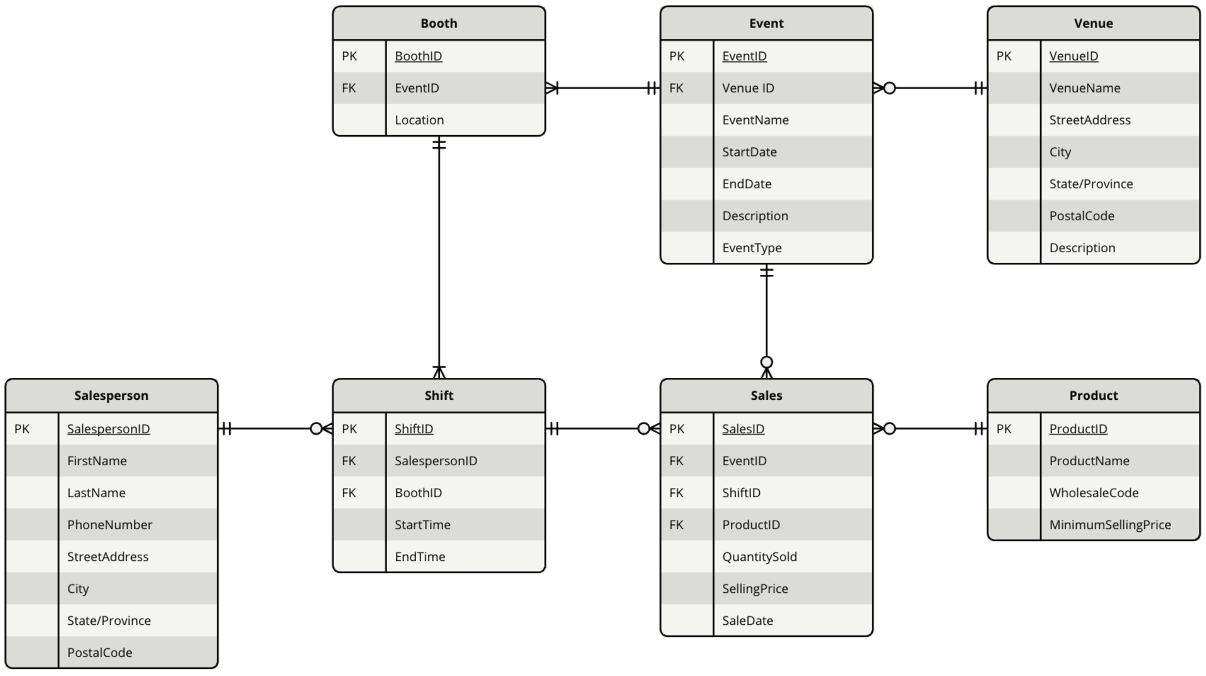


Table Definitions

The logical model comprises the following interrelated tables:

- **Venue Table**:
  - VenueID (VARCHAR(10), Primary Key, Not Null)

  - VenueName (VARCHAR(50), Not Null)
  - StreetAddress (VARCHAR(50), Not Null)
  - City (VARCHAR(50), Not Null)
  - State (CHAR(50), Not Null)
  - PostalCode (NUMERIC(5), Not Null)
  - Description (VARCHAR(255), Not Null)

- **Event Table**:
  - EventID (VARCHAR(10), Primary Key, Not Null)
  - VenueID (VARCHAR(10), Foreign Key referencing Venue(VenueID), Not Null)
  - EventName (VARCHAR(20), Not Null)
  - StartDate (DATE, Not Null)
  - EndDate (DATE, Not Null)
  - Description (VARCHAR(255))
  - EventType (VARCHAR(25), Check Constraint (EventType IN ('Trade Show,' 'Rib Fest,' 'Waterfront festival,' 'Music Festival,' 'Beer Festival,' 'Sporting Event,' 'Street Festival')))

- **Booth Table**:
  - BoothID (VARCHAR(10), Primary Key, Not Null)
  - EventID (VARCHAR(10), Foreign Key referencing Event(EventID), Not Null)
  - Location (VARCHAR(255), Not Null)

- **Product Table**:
  - ProductID (VARCHAR(10), Primary Key, Not Null)
  - ProductName (VARCHAR(50), Not Null)
  - WholesaleCode (DECIMAL(10, 2), Not Null)
  - MinSellPrice (DECIMAL(10, 2), Not Null)

- **Salesperson Table**:
  - SalespersonID (VARCHAR(10), Primary Key, Not Null)
  - FirstName (VARCHAR(20), Not Null)
  - LastName (VARCHAR(20), Not Null)
  - StreetAddress (VARCHAR(50), Not Null)
  - City (VARCHAR(50), Not Null)
  - State (CHAR(50), Not Null)
  - PostalCode (NUMERIC(5), Not Null)
  - PhoneNumber (NUMERIC(10), Not Null, Unique)

- **Shift Table**:
  - ShiftID (VARCHAR(10), Primary Key, Not Null)

- SalespersonID (VARCHAR(10), Foreign Key referencing Salesperson(SalespersonID), Not Null)
- BoothID (VARCHAR(10), Foreign Key referencing Booth(BoothID), Not Null)
- StartTime (TIMESTAMP, Not Null)
- EndTime (TIMESTAMP, Not Null)

- **Sales Table**:
  - SalesID (VARCHAR(10), Primary Key, Not Null)
  - EventID (VARCHAR(10), Foreign Key referencing Event(EventID), Not Null)
  - ShiftID (VARCHAR(10), Foreign Key referencing Shift(ShiftID), Not Null)
  - ProductID (VARCHAR(10), Foreign Key referencing Product(ProductID), Not Null)
  - QuantitySold (NUMERIC(10), Not Null)
  - SellingPrice (DECIMAL(10, 2), Not Null)
  - SaleDate (DATE, Not Null)

Relationships and Data Consistency

The relationships between tables are established through foreign keys. For instance, the EventID in the Booth table references the primary key of the Event table, ensuring that a booth can only be associated with a valid event. Similarly, foreign key relationships exist between the other tables, maintaining data consistency and preventing the creation of orphaned records (records with invalid references). This logical relational database model provides a solid foundation for the physical implementation of the FoxCore Retail database. By translating the conceptual data model into a structure of well-defined tables with enforced constraints, the logical model paves the way for efficient data storage, retrieval, and analysis, ultimately empowering FoxCore Retail to make informed business decisions.

Physical Database Design

The logical relational database model serves as a blueprint for the physical implementation of the database. This section delves into transforming the logical model's abstract structure into a concrete representation that leverages the specific features and functionalities of the chosen Database Management System (DBMS). The physical database design optimizes storage efficiency, query performance, and overall database maintenance.

Data Definition Language (DDL) Statements

The Data Definition Language (DDL) statements below are designed to create the requisite database tables founded on the logical (relational) database model.

```
CREATE TABLE Venue(
  VenueID VARCHAR(10) NOT NULL,
  VenueName VARCHAR(50) NOT NULL,
  StreetAddress VARCHAR(50) NOT NULL,
  City VARCHAR(50) NOT NULL,
  State CHAR(50) NOT NULL,
  PostalCode NUMERIC(5) NOT NULL,
  Description VARCHAR(255) NOT NULL,
  PRIMARY KEY (VenueID));

CREATE TABLE Event(
  EventID VARCHAR(10) NOT NULL,
  VenueID VARCHAR(10) NOT NULL,
  EventName VARCHAR(50) NOT NULL,
  StartDate DATE NOT NULL,
  EndDate DATE NOT NULL,
  Description VARCHAR(255),
  EventType VARCHAR(25),
  CHECK (Eventtype IN ('Trade Show', 'Rib Fest', 'Waterfront festival', 'Music Festival',
'Beer Festival', 'Sporting Event', 'Street Festival')),
  PRIMARY KEY (EventID),
  FOREIGN KEY (VenueID) REFERENCES Venue(VenueID)
    ON UPDATE CASCADE
    ON DELETE NO ACTION);

CREATE TABLE Booth(
  BoothID VARCHAR(10) NOT NULL,
  EventID VARCHAR(10) NOT NULL,
  Location VARCHAR(255) NOT NULL,
  PRIMARY KEY (BoothID),
  FOREIGN KEY (EventID) REFERENCES Event(EventID)
    ON UPDATE CASCADE
    ON DELETE NO ACTION);

CREATE TABLE Product(
  ProductID VARCHAR(10) NOT NULL,
  ProductName VARCHAR(50) NOT NULL,
  WholesaleCode DECIMAL(10, 2) NOT NULL,
  MinSellPrice DECIMAL(10, 2) NOT NULL,
  PRIMARY KEY (ProductID));

CREATE TABLE Salesperson(
  SalespersonID VARCHAR(10) NOT NULL,
  FirstName VARCHAR(20) NOT NULL,
  LastName VARCHAR(20) NOT NULL,
  StreetAddress VARCHAR(50) NOT NULL,
  City VARCHAR(50) NOT NULL,
  State CHAR(50) NOT NULL,
  PostalCode NUMERIC(5) NOT NULL,
  PhoneNumber NUMERIC(10) NOT NULL,
  PRIMARY KEY (SalespersonID),
  UNIQUE KEY (PhoneNumber));
```

```
CREATE TABLE Shift(
  ShiftID VARCHAR(10) NOT NULL,
  SalespersonID VARCHAR(10) NOT NULL,
  BoothID VARCHAR(10) NOT NULL,
  StartTime TIMESTAMP NOT NULL,
  EndTime TIMESTAMP NOT NULL,
  PRIMARY KEY (ShiftID),
  FOREIGN KEY (SalespersonID) REFERENCES Salesperson(SalespersonID)
  FOREIGN KEY (BoothID) REFERENCES Booth(BoothID));

CREATE TABLE Sales(
  SalesID VARCHAR(10) NOT NULL,
  EventID VARCHAR(10) NOT NULL,
  ShiftID VARCHAR(10) NOT NULL,
  ProductID VARCHAR(10) NOT NULL,
  QuantitySold NUMERIC(10) NOT NULL,
  SellingPrice DECIMAL(10, 2) NOT NULL,
  StartDate DATE NOT NULL,
  PRIMARY KEY (SalesID),
  FOREIGN KEY (EventID) REFERENCES Event(EventID)
  FOREIGN KEY (ShiftID) REFERENCES Shift(ShiftID)
  FOREIGN KEY (ProductID) REFERENCES Product(ProductID));
```

Data Manipulation Language (DML) Statements:

Using Data Manipulation Language (DML) statements is fundamental to inserting and managing data in a database. These statements enable efficient manipulation of data over time, providing an essential foundation for the effective storage and retrieval of information:

- **INSERT:** Insert new data into a table.
  - Example: `INSERT INTO Venue (VenueID, VenueName, StreetAddress, City, State, PostalCode, Description) VALUES ('V001', 'Convention Center', '123 Main St', Toronto, Ontario, '12345', 'Large venue for conventions and trade shows');`

- **UPDATE:** Modify existing data in a table.
  - Example: `UPDATE Event SET EventName = ‘FoxCore Summer Festival’ WHERE EventID = ‘E002’;`

- **DELETE:** Remove data from a table.
  - Example: `DELETE FROM Product WHERE ProductID = ‘P001’;`

Sample Queries:

Please take note that the FoxCore Retail database is a valuable resource that can be utilized to extract and scrutinize data using the exemplar queries listed below:

- **Find all events at a specific venue:**

```sql
SELECT EventName, StartDate, EndDate
FROM Event
WHERE VenueID = ‘V001’;
```

- **Find total sales for a product across all events along with their venues:**

```sql
SELECT v.VenueName, e.EventName, e.StartDate, e.EndDate, SUM(s.QuantitySold *
s.SellingPrice) AS TotalSales
FROM Event e
JOIN Sales s ON e.EventID = s.EventID
JOIN Venue v ON e.VenueID = v.VenueID
GROUP BY v.VenueName, e.EventName, e.StartDate, e.EndDate
ORDER BY v.VenueName, TotalSales DESC;
```

- **Find salespeople who worked on a particular event, as well as their total sales and commission for that event. Assuming that the commission is 5% of the total sales.**

```sql
SELECT s.FirstName, s.LastName, SUM(sa.QuantitySold * sa.SellingPrice) AS
TotalSales, SUM(sa.QuantitySold * sa.SellingPrice) * 0.05 AS Commission
FROM Salesperson s
INNER JOIN Shift sh ON s.SalespersonID = sh.SalespersonID
INNER JOIN Sales sa ON sh.ShiftID = sa.ShiftID
INNER JOIN Event e ON sa.EventID = e.EventID
WHERE e.EventID = ‘E002’
GROUP BY s.FirstName, s.LastName;
```

- **Find which product is sold by which salesperson at what quantity and price at a particular event and on which event date.**

```sql
SELECT p.productName, s.FirstName, s.LastName, sa.QuantitySold, sa.SellingPrice,
e.Eventname, sa.salesdate
FROM Salesperson s
JOIN Shift sh ON s.SalespersonID = sh.SalespersonID
JOIN Sales sa ON sh.ShiftID = sa.ShiftID
JOIN Event e ON sa.EventID = e.EventID
JOIN product p ON p.productID=sa.productID;
```

- **Find the month-on-month total sales.**

```sql
SELECT YEAR(s.SalesDate) AS SaleYear, MONTH(s.SalesDate) AS SaleMonth,
SUM(s.QuantitySold * s.SellingPrice) AS TotalSales
FROM Sales s
GROUP BY YEAR(s.SalesDate), MONTH(s.SalesDate)
ORDER BY SaleYear, SaleMonth;
```

- **Individual sales of different products.**

```sql
SELECT p.productid, p.productname, SUM(sa.QuantitySold * sa.SellingPrice) AS
TotalSales
FROM Sales sa
JOIN product p ON p.productID=sa.productID
JOIN event e ON e.eventID=sa.eventID
JOIN shift sh ON sh.shiftID= sa.shiftid
GROUP BY p.ProductID;
```

- **The total quantity of a product sold by a salesperson at an event**

```
SELECT sp.Firstname, sp.Lastname,e.eventname, p.productname,
SUM(s.quantitysold) AS Totalquantitysold
FROM sales s
JOIN Shift sh ON s.Shiftid=sh.shiftid
JOIN event e ON e.eventid=s.eventid
JOIN product p ON s.productid=p.productid
JOIN salesperson sp ON sh.salespersonid=sp.salespersonid
GROUP BY sp.salespersonid,e.eventid, p.productid;
```

- **Find the total sales of all products in events.**

```
SELECT e.EventName, p.ProductName, SUM(s.QuantitySold * s.SellingPrice) AS
TotalSales
FROM Sales s
JOIN Event e ON s.EventID = e.EventID
JOIN Product p ON s.ProductID = p.ProductID
GROUP BY e.EventName, p.ProductName
ORDER BY TotalSales DESC;
```

Conclusion

This report outlines the design of a relational database custom-built for Foxcore Retail. This database aims to provide a streamlined and effective method for storing, retrieving, and analyzing data, aiming to improve sales tracking, inventory management, and event performance evaluation. The report includes DDL and DML statements, offering a solid foundation for creating and manipulating the database. Furthermore, we have included sample queries that showcase how to extract valuable insights from the data to make informed decisions.

References

Neufeld, Derrick, et al. “Foxcore Retail (A): Designing a Database.” *Ivey Publishing - Ivey Business School*, 21 Dec. 2018, www.iveypublishing.ca/s/product/foxcore-retail-a-designing-a-database/01t5c00000CwoXf.