**C170 – Data Management – Applications**

**Task 1**

**Zachary Hoppock**

**Western Governors University**

**March 23, 2023**

VHT2 Task 1: Normalization and Database Design

1. **Construct a normalized physical database model to represent the ordering process for Nora’s Bagel Bin by doing the following:**
   1. Complete the second normal form (2NF) section of the attached “Nora’s Bagel Bin Database Blueprints” document by doing the following:
      1. *Assign each attribute from the 1NF table into the correct 2NF table.*
      2. *Describe the relationship between the two pairs of 2NF tables by indicating their cardinality in each of the dotted cells: one-to-one (1:1), one-to-many (1:M), many-to-one (M:1), or many-to-many (M:M).*

*Diagram, table

Description automatically generated*

* + 1. *Explain how you assigned attributes to the 2NF tables and determined the cardinality of the relationships between your 2NF tables.*
       - For the Bagel table, the Primary Key would need to be Bagel ID as it will be referenced in the Bager Order Line Item table. Each Bagel ID is unique and would need to be given a name, a description, and a set price. All of these attributes rely solely on the Primary Key.
       - For the Bagel Order Line Item table, a composite Primary Key is already set in place, with each member of the key being a foreign key referencing respective tables. For each Bagel Order ID, you can have many different Bagel IDs for it. You would also need to specify the quantity of that Bagel ID you will be ordering.
       - For the Bagel Order table, the Primary Key would need to be Bagel Order ID as it will be referenced in the Bagel Order Line Item table. For each order, you would put down the date it was ordered, the first and name of the customer who placed the order, the street address, city, state, and zip code for the order to be delivered to, the phone number of the customer who made the order, the delivery fee of the order, and any special notes to be included on the order. All of these attributes rely solely on the Primary Key at this time.
       - For the cardinality of the Bager Order and Bagel Order Line Item tables, one order can have many line items included on it, but many line items can only belong to one order. This is due to the composite primary key in which each line item has different bagel IDs that would belong to one order ID. Therefore, a maximum of many ordered bagels would belong to one order. You would not include ordered bagels on any other order as they would already belong to one order.
       - For the cardinality of the Bagel Order Line Item and Bagel tables, one line item belongs to one type of bagel, but a type of bagel can belong to many line items. The bagel ID on a line refers to the details of a specified type of bagel alone, however the type of bagel can be included on an order multiple times if needed. For example, this could happen if a customer orders *n* amount of type *x* bagel and some type *y* bagels, but near the end of placing the order realizes they want *m* more of type *x* bagel along with their type *y* bagels.
  1. Complete the third normal form (3NF) section of the attached “Nora’s Bagel Bin Database Blueprints” document by doing the following:
     1. *Assign each attribute from your 2NF "Bagel Order" table into one of the new 3NF tables. Copy all other information from your 2NF diagram into the 3NF diagram.*
     2. *Provide each 3NF table with a name that reflects its contents.*
     3. *Create a new field that will be used as a key linking the two 3NF tables you named in part A2b. Ensure that your primary key (PK) and foreign key (FK) fields are in the correct locations in the 3NF diagram.*
     4. *Describe the relationships between the 3NF tables by indicating their cardinality in each of the dotted cells: one-to-one (1:1), one-to-many (1:M), many-to-one (M:1), or many-to-many (M:M).*

*Diagram

Description automatically generated with medium confidence*

* + 1. *Explain how you assigned attributes to the 3NF tables and determined the cardinality of the relationships between your 3NF tables.*
       - For the Bagel table, the Primary Key would need to be Bagel ID as it will be referenced in the Bager Order Line Item table. Each Bagel ID is unique and would need to be given a name, a description, and a set price. All of these attributes rely solely on the Primary Key.
       - For the Bagel Order Line Item table, a composite Primary Key is already set in place, with each member of the key being a foreign key referencing respective tables. For each Bagel Order ID, you can have many different Bagel IDs for it. You would also need to specify the quantity of that Bagel ID you will be ordering.
       - For the Catering Order table (originally named the Bagel Order table but renamed to identify what kind of order the bagel shop makes), the Primary Key would need to be Bagel Order ID as it will be referenced in the Bagel Order Line Item table. For each order, you would put down the customer ID of the customer who placed the order, the date it was ordered, the delivery fee of the order, and any special notes to be included on the order. All of these attributes rely solely on the Primary Key. The customer ID attribute is set up as a Foreign Key and will reference details from the Customer table.
       - For the Customer table, the Primary Key would need to be Customer ID as it will be referenced in the Catering Order table. Each entry will list a customer’s first and last name, the street address, city, state, and zip code of the customer, and the phone number of the customer. All of these attributes rely solely on the Primary Key.
       - For the cardinality of the Catering Order and Bagel Order Line Item tables, one order can have many line items included on it, but many line items can only belong to one order. This is due to the composite primary key in which each line item has different bagel IDs that would belong to one order ID. Therefore, a maximum of many ordered bagels would belong to one order. You would not include ordered bagels on any other order as they would already belong to one order.
       - For the cardinality of the Bagel Order Line Item and Bagel tables, one line item belongs to one type of bagel, but a type of bagel can belong to many line items. The bagel ID on a line refers to the details of a specified type of bagel alone, however the type of bagel can be included on an order multiple times if needed. For example, this could happen if a customer orders *n* amount of type *x* bagel and some type *y* bagels, but near the end of placing the order realizes they want *m* more of type *x* bagel along with their type *y* bagels.
       - For the cardinality of the Catering Order and Customer tables, many orders can be made by one customer and one customer can place up many orders. You cannot have multiple customers on one order otherwise the Catering Order table would need a composite Primary Key made that includes the customer ID, which would then require the order details to be placed into it’s own table for normality. This is not the case though as you wouldn’t have multiple customers ordering the same items on one order so the many to one relationship works.
  1. Complete the "Final Physical Database Model" section of the attached “Nora’s Bagel Bin Database Blueprints” document by doing the following:
     1. *Copy the table names and cardinality information from your 3NF diagram into the “Final Physical Database Model” and rename the attributes.*
     2. *Assign one of the following five data types to each attribute in your 3NF tables: CHAR(), VARCHAR(), TIMESTAMP, INTEGER, or NUMERIC(). Each data type must be used at least once.*

*See next page for table.*

A picture containing table

Description automatically generated

1. **Create a database using the attached "Jaunty Coffee Co. ERD" by doing the following:**
   1. Develop SQL code to create each table as specified in the attached “Jaunty Coffee Co. ERD” by doing the following:
      1. *Provide the SQL code you wrote to create all the tables.*

CREATE DATABASE jaunty\_coffee\_co;

USE jaunty\_coffee\_co;

CREATE TABLE coffee\_shop (

shop\_id INT,

shop\_name VARCHAR(50),

city VARCHAR(50),

state CHAR(2),

PRIMARY KEY (shop\_id)

);

CREATE TABLE supplier (

supplier\_id INT,

company\_name VARCHAR(50),

country VARCHAR(30),

sales\_contact\_name VARCHAR(60),

email VARCHAR(50) NOT NULL,

PRIMARY KEY (supplier\_id)

);

CREATE TABLE employee (

employee\_id INT,

first\_name VARCHAR(30),

last\_name VARCHAR(30),

hire\_date DATE,

job\_title VARCHAR(30),

shop\_id INT,

PRIMARY KEY (employee\_id),

FOREIGN KEY (shop\_id) REFERENCES coffee\_shop(shop\_id)

);

CREATE TABLE coffee (

coffee\_id INT,

shop\_id INT,

supplier\_id INT,

coffee\_name VARCHAR(30),

price\_per\_pound NUMERIC(5, 2),

PRIMARY KEY (coffee\_id),

FOREIGN KEY (shop\_id) REFERENCES coffee\_shop(shop\_id),

FOREIGN KEY (supplier\_id) REFERENCES supplier(supplier\_id)

);

* + 1. *Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.*

See next page for screenshot.

*Graphical user interface, application

Description automatically generated*

* 1. Develop SQL code to populate each table in the database design document by doing the following:
     1. *Provide the SQL code you wrote to populate the tables with at least three rows of data in each table.*

INSERT INTO coffee\_shop

VALUES (1, 'Coffee Bean Needs', 'Miami', 'FL'),

(2, 'Java Needs', 'Tampa Bay', 'FL'),

(3, 'Mocha Lot, Eh?', 'Atlanta', 'GA');

INSERT INTO employee

VALUES (11, 'James', 'Brendon', '2015-04-12', 'Manager', 1),

(12, 'Madison', 'Sauls', '2015-04-14', 'Barista', 1),

(13, 'Jeremy', 'Elbertson', '2015-05-02', 'Barista', 1),

(21, 'Monique', 'Mendez', '2016-11-23', 'Manager', 2),

(22, 'Thomas', 'Sholts', '2016-11-23', 'Barista', 2),

(23, 'Timothy', 'Morris', '2016-11-28', 'Barista', 2),

(31, 'Velma', 'Dinklidge', '2016-12-01', 'Manager', 3),

(32, 'Fred', 'Jonesy', '2016-12-10', 'Barista', 3),

(33, 'Madison', 'Lillith', '2017-01-09', 'Barista', 3);

INSERT INTO supplier

VALUES (100, 'Miami Blend Specialty', 'United States', 'Dillon Michaels', 'dmichaels@mblends.com'),

(200, 'Scottsdale Java', 'Ireland', 'Sam Smith', 'samsmith@scottsdalej.com'),

(300, 'Southern Hospitality', 'United States', 'Terry Grimes', 'terrygrimes@sohos.com');

INSERT INTO coffee

VALUES (101, 1, 100, 'Ocean Breeze Latte', 1.75),

(102, 1, 100, 'Mocha Sea, Mocha Do', 1.95),

(201, 2, 100, 'Latte On The Rocks', 1.55),

(202, 2, 300, 'Fresh Blend Coffee', 2.75),

(301, 3, 200, 'Top Of the Morning', 1.35),

(302, 3, 300, 'Dark Blend Coffee', 2.65);

* + 1. *Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.*

*A picture containing text

Description automatically generated*

* 1. Develop SQL code to create a view by doing the following:
     1. *Provide the SQL code you wrote to create your view. The view should show all of the information from the “Employee” table but concatenate each employee’s first and last name, formatted with a space between the first and last name, into a new attribute called employee\_full\_name.*

CREATE VIEW employee\_info AS

SELECT

employee\_id,

CONCAT(first\_name, ' ', last\_name) AS employee\_full\_name,

hire\_date,

job\_title,

shop\_id

FROM

employee;

* + 1. *Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.*

*Graphical user interface, text, application, email

Description automatically generated*

* 1. Develop SQL code to create an index on the coffee\_name field by doing the following:
     1. *Provide the SQL code you wrote to create your index on the coffee\_name field from the “Coffee” table.*

CREATE INDEX coffee\_name\_idx

ON coffee (coffee\_name);

* + 1. *Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.*

*Graphical user interface, text, application

Description automatically generated*

* 1. Develop SQL code to create an SFW (SELECT–FROM–WHERE) query for any of your tables or views by doing the following:
     1. *Provide the SQL code you wrote to create your SWF query.*

SELECT

supplier\_id,

coffee\_name,

price\_per\_pound

FROM

coffee

WHERE

price\_per\_pound < 2;

* + 1. *Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.*

*Graphical user interface, text, application, email

Description automatically generated*

* 1. Develop SQL code to create a query by doing the following:
     1. *Provide the SQL code you wrote to create your table joins query. The query should join together three different tables and include attributes from all three tables in its output.*

SELECT

shop\_name, city, state, -- from coffee\_shop table

coffee\_name, price\_per\_pound, -- from coffee table

company\_name AS supplier\_name, country AS supplier\_country -- from supplier table

FROM coffee\_shop

INNER JOIN coffee ON coffee\_shop.shop\_id = coffee.shop\_id

INNER JOIN supplier ON coffee.supplier\_id = supplier.supplier\_id

ORDER BY shop\_name;

* + 1. *Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.*

*Graphical user interface, text, application, email

Description automatically generated*