|  |  |  |
| --- | --- | --- |
| Julie Sweany | | |
|  | | Paste in your answers for each step below in this column. Paste your actual SQL text, not a screen shot of it. Paste a screen shot of your Result Grid. Be sure to “Snip” your screen shots. |
|  | Final EER Diagram |  |
|  | Crayon Definitions | |  |  | | --- | --- | | **Entity Name** | **Crayon Definition** | | Customer | The Customer table contains records about people who make purchases in the shops. | | Shop | The Shop table contains records for each of the bakery’s retail locations. | | Sale | The Sale table contains records about each purchase of bakery product(s). | | SaleLineItem | The SaleLineItem table is a linking table which allows for the many-to-many relationship between the Sale and Product tables. | | Employee | The Employee table contains records for the people who work at the shops. | | Product | The Product table contains records for each of the products offered for sale in the shops. | | WorksIn | The WorksIn table is a linking table which allows for the many-to-many relationship between the Employee and Shop tables. | | Vendor | The Vendor table contains records for each of the vendors from which the bakery buys ingredients. | | Order | The Order table contains records for orders of ingredients that the bakery is purchasing from vendors. | | OrderLineItem | The OrderLineItem table is a linking table which allows for the many-to-many relationship between the Order and Ingredient tables. | | Ingredient | The Ingredient table contains records for each of the ingredients used in producing the products. | | IngredientRecipe | The IngredientRecipe table is a linking table which allows for the many-to-many relationship between the Ingredient and Recipe tables. | | Recipe | The Recipe table contains records for the recipe used to make each of the products. | | UnitOfMeasure | The UnitOfMeasure table contains records for different units of measure used to indicate amounts of ingredients. | |
|  | Mockarroo.com screen shot of your Customer table setup | **Mockaroo screen shot of your Customer table setup** |
| SELECT \* on each table and up to 10 records from the result grid of each | **SELECT \* statement and Result Grid from each table**  USE BrendasBakeries;  SELECT  \*  FROM  Customer;    USE BrendasBakeries;  SELECT  \*  FROM  Employee;    USE BrendasBakeries;  SELECT  \*  FROM  Ingredient;    USE BrendasBakeries;  SELECT  \*  FROM  IngredientRecipe;    USE BrendasBakeries;  SELECT  \*  FROM  `Order`;    USE BrendasBakeries;  SELECT  \*  FROM  OrderLineItem;    USE BrendasBakeries;  SELECT  \*  FROM  Product;    USE BrendasBakeries;  SELECT  \*  FROM  Recipe;    USE BrendasBakeries;  SELECT  \*  FROM  Sale;    USE BrendasBakeries;  SELECT  \*  FROM  SaleLineItem;    USE BrendasBakeries;  SELECT  \*  FROM  Shop;    USE BrendasBakeries;  SELECT  \*  FROM  UnitOfMeasure;    USE BrendasBakeries;  SELECT  \*  FROM  Vendor;    USE BrendasBakeries;  SELECT  \*  FROM  WorksIn; |
|  | Explanation | A **WHERE** and an **ORDER BY** clause.  Output the first and last names of all employees who have worked at the Downtown location and order by last name. |
| Result Grid |  |
| SQL | -- SP3.4 - Julie Sweany  -- Output the first and last names of all employees who have worked at the Downtown location and order by last name  USE BrendasBakeries;  SELECT DISTINCT  FirstName AS 'First Name', LastName AS 'Last Name'  FROM  Employee  JOIN  WorksIn ON Employee.ID = EmployeeID  WHERE  ShopID = 1  ORDER BY LastName; |
|  | Explanation | An **AND** or **OR** in the **WHERE** clause.  List the date, product, quantity purchased, and shop for all gingerbread cookies sold in the West side shop. |
| Result Grid |  |
| SQL | -- SP3.5 - Julie Sweany  -- Output information on all gingerbread cookie purchases made in the West side shop. Include date of sale, name of product, and quantity of cookies purchased.  -- Order results by date.  USE BrendasBakeries;  SELECT  Date AS 'Date of Sale',  Product.Name AS Product,  Quantity AS 'Number Purchased',  Shop.Name AS Shop  FROM  Sale  JOIN  SaleLineItem ON Sale.ID = SaleID  JOIN  Product ON ProductID = Product.ID  JOIN  Shop ON Sale.ShopID = Shop.ID  WHERE  ShopID = 2 AND ProductID = 8  ORDER BY Date; |
|  | Explanation | **GROUP BY**  For frequently ordered ingredients (more than 19 orders in this example), output the number of orders, ingredient name, unit of measure, lowest price, highest price and average price for each ingredient. Order by number of orders and then by ingredient. |
| Result Grid |  |
| SQL | -- SP3.6 - Julie Sweany  -- Output by ingredient, the number of orders, ingredient name, units, lowest price, highest price, and average price of frequently ordered ingredients.  -- Order results by number of orders and then by ingredient name.  USE BrendasBakeries;  SELECT  COUNT(\*) AS 'Number of Orders',  Ingredient.Name AS Ingredient,  UnitOfMeasure.Name AS Units,  MIN(Price) AS 'Lowest Price',  MAX(Price) AS 'Highest Price',  ROUND(AVG(Price), 2) AS 'Average Price'  FROM  OrderLineItem  JOIN  Ingredient ON IngredientID = Ingredient.ID  JOIN  UnitOfMeasure ON Ingredient.UnitOfMeasureID = UnitOfMeasure.ID  GROUP BY IngredientID  HAVING COUNT(\*) > 19  ORDER BY COUNT(\*) , Ingredient.Name; |
|  | Explanation | **INNER JOIN**  Output customers’ IDs, names, email addresses, phone numbers and the dates and locations of their purchases. Order by last names. |
| Result Grid |  |
| SQL | -- SP3.7 - Julie Sweany  -- Output customers' IDs, names, email addresses, and phone numbers and the dates and locations of their purchases.  -- Order results last name of customers.  USE BrendasBakeries;  SELECT  Customer.ID AS 'Customer ID',  CONCAT(LastName, ', ', FirstName) AS Name,  Email AS 'Email Address',  CONCAT('(',  LEFT(Phone, 3),  ') ',  SUBSTRING(Phone, 4, 3),  '-',  SUBSTRING(Phone, 7, 4)) AS 'Phone Number',  Date AS 'Purchase Date',  Shop.Name AS 'Purchase Location'  FROM  Customer  JOIN  Sale ON Customer.ID = CustomerID  JOIN  Shop ON Sale.ShopID = Shop.ID  ORDER BY LastName; |
|  | Explanation | **OUTER JOIN**  Output all ingredients and the recipes in which the ingredients are used. Include all ingredients, even those that are not used in recipes currently stored in the database. |
| Result Grid |  |
| SQL | -- SP3.8 - Julie Sweany  -- Output list of all ingredients and the recipes in which they are used. Include ingredients currently not included in recipes.  -- Order by ingredient.  USE BrendasBakeries;  SELECT  Ingredient.Name AS Ingredient,  Recipe.Name AS 'Used in recipe for:'  FROM  Recipe  JOIN  IngredientRecipe ON RecipeID = Recipe.ID  RIGHT JOIN  Ingredient ON IngredientID = Ingredient.ID  ORDER BY Ingredient.Name; |
|  | Explanation | **Roll Your Own query**  Output the total dollar amount of sales for each shop. Order results by total sales amount. |
| Result Grid |  |
| SQL | -- SP3.9 - Julie Sweany  -- Output total sales amount grouped by shop.  -- Order results by dollar amount of total sales.  USE BrendasBakeries;  SELECT  Shop.Name AS Shop, CONCAT('$', FORMAT(SUM(Quantity \* Price), 2)) AS 'Total Sales'  FROM  Shop  JOIN  Sale ON Shop.ID = ShopID  JOIN  SaleLineItem ON Sale.ID = SaleID  JOIN  Product ON Product.ID = ProductID  GROUP BY Shop  ORDER BY SUM(Quantity \* Price); |
|  | Explanation | **View**  The view is meant to provide easy access to information about recent orders with vendors. |
| SQL | -- SP3.10 - Julie Sweany  -- Output information (Date, OrderID, Employee Name, Vendor Name, Product Name, Quantity, Price, Unit of Measure, Total LIne Amount) for orders placed with vendors.  -- Order results by date, with most recent ones appearing first.  USE BrendasBakeries;  -- Create the view  CREATE OR REPLACE VIEW OrderingHistory AS  SELECT  Date,  `Order`.ID AS 'Order Number',  CONCAT(LastName, ', ', FirstName) AS 'Employee Placing Order',  Vendor.Name AS Vendor,  Ingredient.Name AS 'Item Ordered',  Quantity,  Price,  UnitOfMeasure.Name AS 'Per Unit',  CONCAT('$', FORMAT((Quantity \* Price), 2)) AS 'Total Line Amount'  FROM  Employee  JOIN  `Order` ON Employee.ID = EmployeeID  JOIN  Vendor ON VendorID = Vendor.ID  JOIN  OrderLineItem ON `Order`.ID = OrderID  JOIN  Ingredient ON IngredientID = Ingredient.ID  JOIN  UnitOfMeasure ON UnitOfMeasure.ID = Ingredient.UnitOfMeasureID  ORDER BY Date DESC;  -- Using the view  SELECT \* FROM orderinghistory; |
| Result Grid |  |
|  | Explanation | **Stored Procedure**  Create a stored procedure that takes in a Customer ID and returns the total dollar amount of that customer’s spending. |
| SQL | -- SP3-11 - Julie Sweany  -- Create procedure that takes in the customer id for a customer and returns the total amount of that customer's spending.  USE BrendasBakeries ;  -- Create stored procedure  DROP PROCEDURE IF EXISTS TotalCustomerSpending;  DELIMITER $$    CREATE PROCEDURE TotalCustomerSpending(IN CustomerID INT, OUT outTotalSpending DECIMAL(8,2))  BEGIN  DECLARE TotalSpending DECIMAL(8,2) ;    SET TotalSpending =  (SELECT SUM(Quantity \* Price)  FROM  Product  JOIN SaleLineItem ON Product.ID = ProductID  JOIN Sale ON Sale.ID = SaleID  JOIN Customer ON Customer.ID = Sale.CustomerID  WHERE CustomerID = Customer.ID);    SET outTotalSpending = TotalSpending ;    END $$  DELIMITER ;  -- Run the stored procedure    CALL TotalCustomerSpending(35, @outTotalSpending) ;  SELECT CONCAT("$", @outTotalSpending) AS 'Total Spending'; |
| Result Grid |  |
|  | Result Grid | **INSERT Trigger** |
| SQL | -- SP3\_12 - Julie Sweany  -- Create before insert trigger to make sure email is long enough to be valid and contains no blank spaces  USE BrendasBakeries;  DROP TRIGGER IF EXISTS Email\_Insert;  DELIMITER $$  CREATE TRIGGER Email\_Insert  BEFORE INSERT ON Customer  FOR EACH ROW  BEGIN  IF LENGTH(NEW.Email) > 5 THEN  SET NEW.Email = REPLACE(NEW.Email, ' ', '');  END IF;  END $$  DELIMITER ;  -- Insert statement using trigger  USE BrendasBakeries;  INSERT INTO Customer (FirstName, LastName, Email, Phone) values ('Jack', 'Smithers', 'js @bren da s.com', '5555555552');  -- Select statement to show inserted row  USE BrendasBakeries;  SELECT  LastName, FirstName, Email  FROM  Customer  WHERE  ID = last\_insert\_id(); |
| Explanation | The created before insert trigger checks to make sure incoming Email string is long enough to be a valid email address and it removes any extra spaces, if they are present, before inserting the new row. |
|  | Result Grid | **UPDATE Trigger** |
| SQL | -- SP3\_13 - Julie Sweany  -- Create trigger to make sure there is an "@" and a "dot" in the email address. If either is missing, it displays an error message.  DROP TRIGGER IF EXISTS Email\_Update;  DELIMITER $$  CREATE TRIGGER Email\_Update  BEFORE UPDATE ON Customer  FOR EACH ROW  BEGIN  IF LOCATE('@', NEW.Email, 1) = 0 OR LOCATE('.', NEW.Email, 1) = 0 THEN  SIGNAL SQLSTATE 'HY000'  SET MESSAGE\_TEXT = 'Email address must contain "@" and a "dot".';  END IF;  END $$  DELIMITER ;  -- Update statements using trigger  USE BrendasBakeries;  UPDATE Customer  SET  Email = 'LelahEmail@isp.com'  WHERE  ID = 1;  UPDATE Customer  SET  Email = 'TrumannAddress'  WHERE  ID = 2;  UPDATE Customer  SET  Email = 'Daryle@companycom'  WHERE  ID = 3;  -- Select statements to show updated emails  USE BrendasBakeries;  SELECT  ID AS 'Customer ID', Email  FROM  Customer  WHERE  ID IN (1 , 2, 3); |
| Explanation | Created a trigger to make sure updated email addresses contain an “@” and a “dot”. If either is missing, an error message appears in the results. |
|  | Result Grid | **DELETE Trigger** |
| SQL | -- SP3.14 - Julie Sweany  -- Write an after delete trigger to create record of all deletes from customer table  DROP TRIGGER IF EXISTS Customer\_After\_Delete;  DELIMITER $$  CREATE TRIGGER Customer\_After\_Delete  AFTER DELETE ON Customer  FOR EACH ROW  BEGIN  INSERT INTO SysLog  (TableName, Message, Created) VALUES  ( 'Customer', CONCAT('Deleted: Customer ID: ', OLD.ID, ', ', CONCAT(OLD.FirstName, ' ', OLD.LastName)), NOW());  END $$  DELIMITER ;  -- Create new customers with no sales  insert into Customer (FirstName, LastName, EMail, Phone) values ('Sarah', 'Smith', 'ssmith@isp.com', '2165556534');  insert into Customer (FirstName, LastName, EMail, Phone) values ('Eduardo', 'Egghead', 'EduardoEgghead@college.edu', '5555551862');  insert into Customer (FirstName, LastName, EMail, Phone) values ('Joe', 'Worthington', 'jw123@company.com', '4405552226');  -- Delete customers  DELETE FROM Customer  WHERE  ID IN (107 , 108, 109);  -- Output records from SysLog  SELECT  \*  FROM  SysLog  ORDER BY ID DESC; |
| Explanation | Create trigger to add record to SysLog file every time a customer is deleted from the Customer table. |
|  | Summary | **What is the most important thing you personally learned in this project?**  I now have a better understanding of the complexity involved in database design and the importance of design phase. Even with the benefit of being provided with the steps for Phase 3 beforehand, it was hard to foresee everything that would be needed and to get all the small details right.  **If you had to do this over again, what would you do differently?**  I think that if I had sought out more information about database design (especially real-world examples) and asked more questions before beginning the project, that would have made things easier. Also, I should have made more of an effort to try to get realistic data from Mockaroo, limiting the data to a very typical range, rather than trying to accommodate for unusual circumstances. |