TriannaCorp Data Solutions

Tom Frederick • Donnie Long • John Moeller Tara Mysak • Jennifer Paine

**TEAM A**  MIS 531 – University of Arizona, Spring 2017

Table of Contents

[Chapter 1: Requirements Analysis 3](#_Toc481342696)

[Executive Summary 3](#_Toc481342697)

[Corporate Overview 3](#_Toc481342698)

[Products 3](#_Toc481342699)

[Human Relations 4](#_Toc481342700)

[Marketing/Sales 4](#_Toc481342701)

[Financial 5](#_Toc481342702)

[Information Technology 5](#_Toc481342703)

[Other Needs 5](#_Toc481342704)

[Solution 5](#_Toc481342705)

[Chapter 2: Conceptual Schema 7](#_Toc481342706)

[Data Dictionary 8](#_Toc481342707)

[Chapter 3: Relational Schema 14](#_Toc481342708)

[Strong Entity Classes, Sub-Classes, Multi-valued attributes 14](#_Toc481342709)

[Other Relationships, Weak Entity Classes 16](#_Toc481342710)

[Data Dictionary 17](#_Toc481342711)

[Chapter 4: Queries and Views 25](#_Toc481342712)

[Chapter 5: Triggers and Procedures 31](#_Toc481342713)

[Procedure 1 31](#_Toc481342714)

[Procedure 2 35](#_Toc481342715)

[Chapter 6: Web Interface Walkthrough 38](#_Toc481342716)

[Reports and Views 38](#_Toc481342717)

[Data Management 39](#_Toc481342718)

[Neat! 39](#_Toc481342719)

[Chapter 7: Conclusions, Challenges, and Next Steps 41](#_Toc481342720)

[Methodology 41](#_Toc481342721)

[Current Challenges 41](#_Toc481342722)

[Advantages 41](#_Toc481342723)

[Next Steps 42](#_Toc481342724)

[Estimated Costs and Schedule 43](#_Toc481342725)

[Final thoughts 44](#_Toc481342726)

[Bibliography 45](#_Toc481342727)

[CHAPTER 3 APPENDIX A 46](#_Toc481342728)

[CHAPTER 3 APPENDIX B 58](#_Toc481342729)

## Chapter 1: Requirements Analysis

### Executive Summary

TriannaCorp will require a centralized IT database system to manage information across their organization. Centralizing their enterprise data into a RDBMS allows for improved efficiency as a single-point-of-knowledge is used to authoritatively store and quickly view important information. This is contrasted with spreadsheet or other file-based systems, which are more difficult to share as the number of stakeholders increase. Furthermore, the RDBMS would serve as part of a comprehensive online e-tailing system in a way that their current data management regime couldn’t. Finally, TriannaCorp could continue to build valuable features on top of the database system through use of queries, triggers and procedures, and by connecting it to other application systems, like ASP.NET.

### Corporate Overview

TriannaCorp, headquartered in San Diego with main branches in forty states, manufactures health-care products. They are considering a centralized IT system to better take advantage of current technology including a centralized relational database management system (RDBMS). They seek improved sales forecasting, additional sales vectors, and inventory management and are looking into the various enterprise resource planning options.

Among the challenges facing TriannaCorp are their diverse offerings and salesforce dispersed across 40 states. This requires the ability to deploy teams of well-trained customer service representatives who are both responsive and efficient. A data solution needs to take into account how employees are managed and customers are serviced. Following are the company’s needs grouped into general areas.

### Products

TriannaCorp has four main product groups known as the C3D: cosmetic, dental, dermal and diabetic. They are diverse in terms of the markets that are served and the requirements for manufacture. The cosmetics product group currently handles products that do not naturally fall into one of the other core groups. Pharmaceutical products (those may be in the diabetic or dermal product groups) require FDA compliance.

Products within the product groups are divided into product lines. The dermal and dental product groups are the largest of the product groups and have several products within each of their product lines. Database requirements regarding products require ease of retrieving information about product lines and products within each product line that have the highest profitability and volume. It is also important to be able to assess how each product line is being managed. There are a number of brands; each product is associated with a particular brand.

Pricing information needs to be easily updated and be able to be accessed frequently. This should be organized in a flexible way that takes into account the nuanced pricing practices in TriannaCorp. States with the highest sales volumes have premium pricing to enable the better customer service that clients in those states receive. In addition, clients may have discounts associated with them and there may be line item discounts for specific orders. This pricing information should be able to be grouped by product line or brand.

### Human Relations

Requirements for managing employee info would include standard demographic details, salary or wages, training, and assignment by product and location. This can be updated throughout an employee’s tenure with the company. TriannaCorp’s emphasis on lifelong employment means that employee history is of particular importance. This will allow managers to make informed decisions as to how quickly employees are developing, and who may need additional coaching, feedback or responsibilities.

Employee training is also tracked. Training courses are planned and sessions of those courses are delivered by employee instructors to other employees on an annual basis, in a retreat-like atmosphere. Training serves to improve product knowledge, allowing for employees to specialize.

TriannaCorp is organized into both main and local branches with each main branch servicing five to twenty-five local branches. Each branch has one to three managers and a number of client service reps. Main branches have marketing, finance, human resources, IT, and other administrative employees. One or more main-branch nodes form a region.

### Marketing/Sales

Client service representatives, or CSRs, typically visit their clients on a bi-monthly basis, sometimes in teams. Teams of CSRs are assigned to clients by geographical area. During each visit CSRs market new products and take orders.

Every two years the company runs at least one promotion for every product which vary slightly by state. There may be up to forty different promotions for a product running at a point in time managed by brand managers.

Marketing and sales requirements will involve utilizing client data as well as product and employee data about CSRs in particular. Additionally, financial data from order history and pricing will be useful. Together, this will help in the design of promotional strategies and to maintain workload balance and the assignment of employees.

Clients may report incidents involving their CSRs. The incidents are investigated by managers, and may result in a client meeting that may or may not require the CSR’s presence. The CSR may be re-assigned pending a manager’s investigation.

### Financial

Beside the normal cost tracking and budgeting, TriannaCorp must maintain product base costs, per-client and per-state discounts, and order history.  The requirements for the finance department would be to maintain data necessary to support Human Relations (salaries and payroll) and marketing (pricing, billing, and annual sales reports).

### Information Technology

This department must manage the data solution, so their engagement is critical to success. Because the current systems are varied and not centralized, there will be migration issues due to platform compatibility and data organization and type. This is one of the challenges that development of a new RDBMS.

The IT department has generated a core conceptual schema which is mapped to the local sub-schemas. Requirements in this area would utilize the concepts the IT department has put forward on a platform that is maintainable and organized in a way that is adaptable and can evolve as needs change.

### Other Needs

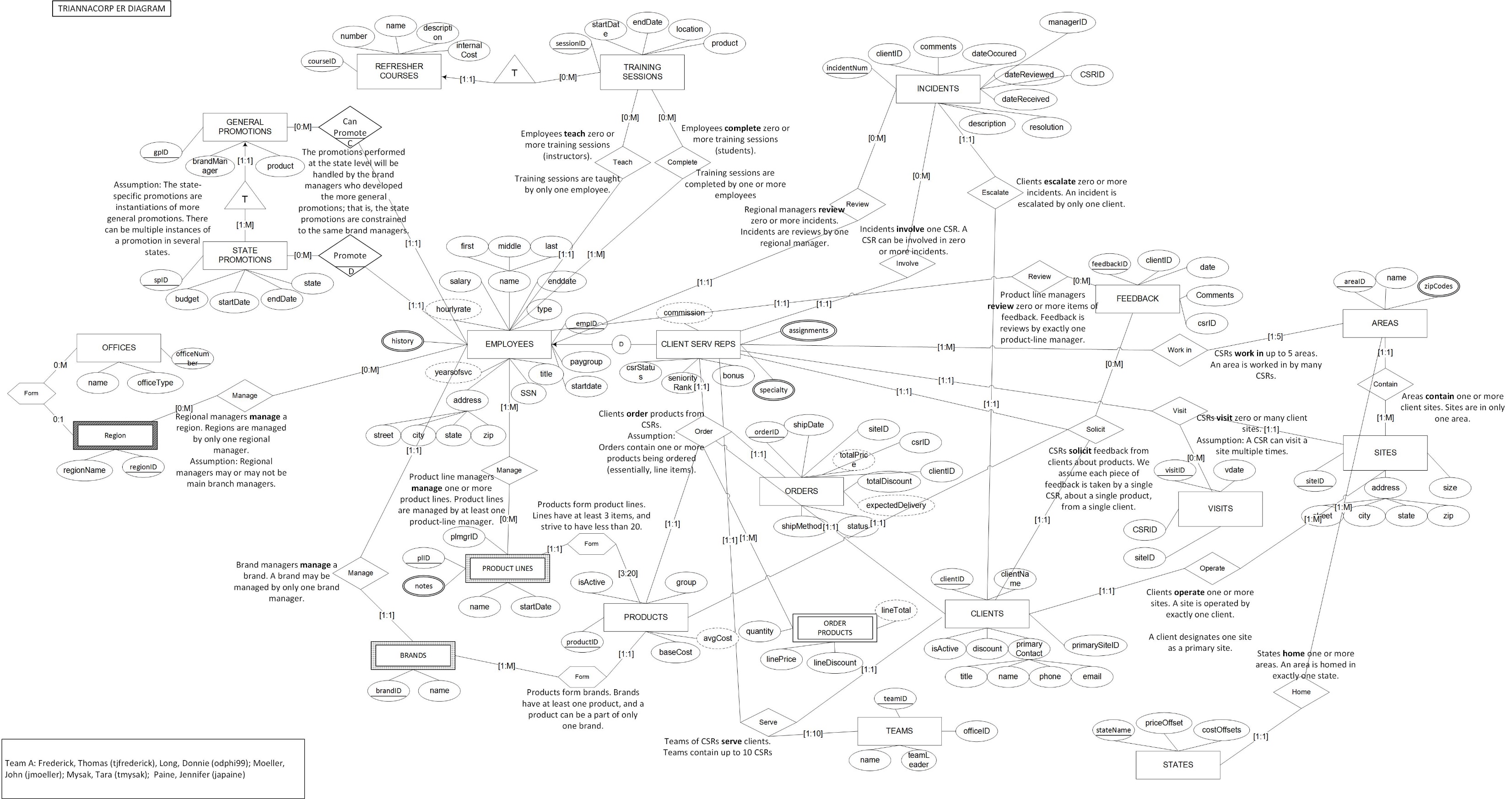
TriannaCorp should consider tracking history of product, client, orders and personnel records. These include orders placed but later cancelled, or not fulfilled. Currently product and order history are in a proprietary database that will not easily be integrated into a new centralized data system. Integrating this history is planned for a later implementation. Client history is in paper files that has been maintained by CSRs and could be transferred into the enterprise database system.

### Solution

A centralized RDBMS that ties all departments and locations together is the ideal approach to addressing the needs of TriannaCorp. It would have the advantage of replacing disparate systems with a comprehensive and flexible single system. This allows integration of data that is usable across the organization. As the departments that have maintained their own proprietary systems know, it is very difficult to share their information with their fellow employees in other departments. When there are those needs, they need to dedicate staff time to collating the data and exporting in a format usable by the other area. Thus, it would be advantageous to allow users in other departments access to data that they need for their areas’ own needs. For instance, the billing information maintained by finance can be accessed by marketing to target clients. Reporting can be customized in a way that information is input and shared across the enterprise, creating efficiencies and therefore, profitability.

This will enable responsiveness to changing market needs and customer demands that TriannaCorp is dedicated to fulfilling. Due to their decentralized structure, it has been a challenge for the main office to be aware of all the activities occurring across divisions and in the branch offices. A centralized data system that integrates all data into one system will enable the head office to efficiently access data it needs about product and employee performance and then make informed decisions about the direction in which to move the company.

## Chapter 2: Conceptual Schema



### Data Dictionary

| Schema Construct | Construct Description | Other Information |
| --- | --- | --- |
| AREA | Entity class - to model the CSR assignments.  They can be assigned up to five. |  |
| areaID | Primary key | Cannot be null |
| name | Name, based on counties |  |
| BRANDS | Entity class |  |
| brandID | Identifying Attribute | Cannot be null |
| brandName | Brand Name |  |
| bmgrID | Brand Manager ID | EmpID of Brand Managers |
| CLIENTS | Entity Class to model company customers |  |
| clientID | Identitying attribute | Cannot be null |
| clientName | Client Name |  |
| primaryContact | Multi-valued attribute Includes name, title, and contact information |  |
| isActive | Y or N |  |
| discount | Base Client Discount |  |
| primarySiteID | Main client office |  |
| EMPLOYEES | Entity Class to model the company employees and attributes. |  |
| emplID | Primary Key | Cannot be null. Unique for each employee. |
| name | Multi-valued attribute that includes first, middle, last |  |
| address | includes street, city, state, zip | For contacting employees and sending pay checks. |
| type | Broad classifications of employees | Based on employee function |
| title | Current job title |  |
| startDate | Start date of current title and salary | (change the diagram) |
| endDate | Term date; can be null | Cannot be earlier than start date |
| yearsofsvc | Derived from start date |  |
| birthDate | Date | For tracking demographical information |
| paygroup | Salary or wage; temporary employees are wage. |  |
| salary | Current Salary |  |
| hourlyrate | derived attribute | derived from salary for permanent employees. Otherwise, the hourly wage for temporary employees. |
| ssn | Social security number | Needed for tax purposes. Must be kept secure. Must be unique. |
| history | Multi-valued attribute that includes start and end dates, title, and salary information. | For tracking past employee history in detail. |
| CLIENT SERVICE REPRESENTIVES | Sub-class of employees |  |
| CSRstatus | Trainee or Full-time employee | CSR's need to visit 10 to 20 clients in the first year to move from trainee to full-time. |
| seniorityRank | Based on how long the CSR has been with the company and used to assign commission. |  |
| bonus | Sales bonus | Based on seniority |
| commission | Derived from performance and sales. |  |
| specialty | Product groups- Multivalued | CSR's can have one or two, and occassionally up to three product line specialities currently. Employees may have training in all four groups. |
| Assignments | Multi-valued for tracking client and area assignments. | CSR's can have 20 to 40 clients and no more than five areas and can be reassigned after 20 to 36 months. |
| FEEDBACK | Entity Class |  |
| feedbackID | identitying attribute | Cannot be null |
| date | Date the feedback was left |  |
| comments | Contact information for the person who left the feedback |  |
| CSRID | Links back to Employees |  |
| productID | Links back to Products |  |
| clientID | Links back to Clients |  |
| Schema Construct | **Construct Description** | **Other Information** |
| GENERAL PROMOTIONS | Typing Class | One to five promotions for every product every two years. Up to 40 at any one time. |
| gpID | Identifying Attribute | Cannot be null |
| brandManagerID | EmpID of Brand Managers, Foreign Key to Brands |  |
| Schema Construct | **Construct Description** | **Other Information** |
| INCIDENTS | Entity Class, to track unhappy clients and the resolutions |  |
| incidentNumber | Identifying Attribute | Cannot be null |
| dateOccurred | Date of the complaint |  |
| dateReceived | Date company received the complaint |  |
| dateReviewed | Date the complaint was reviewed by a manager |  |
| description | Client description of the incident |  |
| comments | Manager and employee comments |  |
| resolution | Outcome of the review |  |
| managerID | Identifying Attribute, Manager assigned to case |  |
| CSRID | Identifying Attribute, CSR involved |  |
| clientID | Identifying Attribute, Client involved |  |
| OFFICES | Entity Class, aggregates to regions, for modeling the physical offices. | A region is formed by one or more offices. Branch managers can service as regional managers. |
| officeID | Identifying Attribute | Cannot be null |
| officeName | Name based on general location | Name is used more commonly than number. |
| OfficeType | Branch, Main or Head Offices |  |
| ORDERS | Entity class for customer orders |  |
| orderID | Identifying Attribute | Cannot be null |
| CSRID | CSR who took the order |  |
| clientID | Customer who made the order |  |
| shipDate | Ship Date of order |  |
| expectedDeliveryDate | Derived from client and item discounts | Used to track rush orders |
| shipMethod | Shipping company used |  |
| totalDiscount | Derived from client and item discounts |  |
| totalPrice | The order total |  |
| siteID | Shipping address, links back to Sites |  |
| orderStatus | Tracks if the order is in process, completed, shipped, cancelled |  |
| orderDate | Date the order was placed |  |
| ORDER\_PRODUCTS | Weak entity class to model order line numbers |  |
| orderID | Composite indentifying attribute | Cannot be null |
| productID | Composite indentifying attribute | Cannot be null |
| ItemDiscount | Additional discounts |  |
| quantity | Total orders |  |
| lineTotal | Total of price and discounts |  |
| Schema Construct | **Construct Description** | **Other Information** |
| PRODUCT LINES | Composite entity class formed by products | Company has at least 10 CSR's for each line. Generally five to ten products per line, but at least three. |
| plID | Identifying Attribute | Cannot be null |
| name | Product Line Name | 1 or 2 lines managed by PL managers; no fewer than 3 brands per line; > 20 products too much |
| startDate | Inception Date |  |
| notes | Multi-valued attribute |  |
| plmID | Foreign Key Employee | ID of product line manager. Each line has one manager. Managers can work with more than one of the smaller lines. |
| PRODUCTS | Entity clases to model product information | 1 product-line/PL manager usually though sometimes 2 product lines/PL manager |
| productID | Identifying Attribute | Cannot be null |
| baseCost | Cost of item before state adjustments | Used to derive average cost |
| productName | Product Name |  |
| isActive | Product is production and has not been discontinued |  |
| group | Group that product belongs to | Cosmetic, Dental, Dermal, Diabetic |
| avgCost | Derived attribute | Derived from base cost and state price offset. |
| REFRESHER\_COURSES | Entity that is a type of training session | Required. Ensures CSR's stay current on product information. |
| courseID | Identifying Attribute | Cannot be null |
| courseName | Course name |  |
| productID | Product for training |  |
| description | Description of the training materials |  |
| internalCost | Cost of training |  |
| REGIONS | Aggregate Class of Offices |  |
| regionID | Identifying Attribute | Cannot be null |
| officeID | Refers back to Offices |  |
| managerID | Refers back to Employees, regional manager |  |
| SITES | Entity for modeling client sites |  |
| siteID | Identifying Attribute | Cannot be null |
| clientID | Refers back to the customers |  |
| siteAddress | multi-valued attribute that includes street, city, state, and zip |  |
| stateID | Refers back to States |  |
| STATES | Entity for modeling states for current and future business |  |
| StateID | Identifying Attribute | Two character abreviation, cannot be null |
| stateName | State Name |  |
| priceOffSet | Differential pricing by state |  |
| regionID | Refers back to Regions |  |
| STATE PROMOTIONS | Typing entity from GENERAL PROMOTIONS |  |
| spID | Identifying Attribute | Cannot be null |
| budget | Allocated budget | at least $10,000. |
| startDate | start of promotion |  |
| endDate | end of promotion |  |
| stateID | Refers back to States | promotions vary by state. |
| Training Sessions | Typing Entity for modeling product training |  |
| courseID | Identifying Attribute, links to refresher courses | Cannot be null |
| sesionID | Identifying Attribute |  |
| employeeID | Employee leading the training |  |
| productID | Product for training |  |
| location | Training site |  |
| startdate | Start date of the training |  |
| enddate | End date of the training |  |
| Teams | Entity for modeling sales teams | Up to ten CSRS. |
| teamID | Identifying Attribute | Cannot be null |
| name | Team name |  |
| teamLeader | refers back to Client Services Representitives | Lead CSR for the team. |
| OfficeID | Refers back to Offices | Assigned branch office. |
| VISITS | Relationship for CSR visits to clients |  |
| visitID | Identifying Attribute | Cannot be null |
| CSRID | Refers to Employees, the CSR who makes the visit |  |
| siteID | Refers to customer site | Larger sites may have several CSR's and more frequent visits. |
| vDate | Date of visit | Bi-monthly average. |

## Chapter 3: Relational Schema

## Strong Entity Classes, Sub-Classes, Multi-valued attributes

**AREAS** (areaID, areaName)

**CLIENTS** (clientID, clientName, isActive, discount, commissionRate, pcTitle, pcName, pcPhone, pcEmail, primarySiteID)

Foreign key (primarySiteID) references SITES(siteID)

**EMPLOYEES** (empID, firstname, middlename, lastname, SSN, streetname, city, state, zip, hiredate, termdate, type, title, paygroup, yearsofsvc, salary)

**CLIENT\_SERV\_REPS** (csrID, seniorityRank, csrStatus, bonus, commission, teamID)

Foreign key (csrID) references EMPLOYEES (empID)

Foreign key (teamID) references TEAMS (teamID)

**FEEDBACK** (feedbackID, comments, fDate, csrID, productID)

Foreign key (csrID) references CLIENT\_SERV\_REPS (csrID)

**GENERAL\_PROMOTIONS** (gpID, productID , brandManagerID)

Foreign key (brandManagerID) references EMPLOYEES (empID)

Foreign key (productID) references PRODUCTS (productID)

**STATE\_PROMOTIONS** (spID, stateID, startDate, endDate, budget, gpID)

Foreign key (stateID) references STATES (stateID)

Foreign key (empID) references EMPLOYEES (empID)

Foreign key (gpID) references GENERAL PROMOTIONS (gpID)s

**INCIDENTS** (incidentNum, dateOccured, dateReceived, dateReviewed, description, comments, resolutions, managerID, csrID, clientID)

Foreign key (managerID) references EMPLOYEES(empID)

Foreign key (csrID) references CLIENT\_SERVICE\_REPRESENTATIVES(empID)

Foreign key (clientID) references CLIENTS(clientID)

**NOTES** (plID, note)

Foreign key (pName) references PRODUCT\_LINES (pName)

**OFFICES** (officeNumber, officetype, name)

**ORDERS** (orderID, clientID, csrID, shipDate, expectedDelivery, shipMethod, isActive, totalDiscount, totalPrice, productID)

Foreign key (clientID) references CLIENTS (clientID)

Foreign key (csrID) references CLIENT\_SERV\_REPS (csrID)

Foreign key (productID) references PRODUCTS (productID)

**PRODUCTS** (productID, productName, isActive, avgCostbaseCost, group)

**RESOLUTION\_MEETINGS** (rmID, incidentNum, rmDate, isCSRPresent, isCSRReassigned)

Foreign key (incidentNum) references INCIDENTS(incidentNum)

**REFRESHER\_COURSES** (courseID, courseNum, courseName, description, internalCost)

**SITES** (siteID, clientID, size, street, city, state, zip)

Foreign key (clientID) references CLIENTS (clientID)

**STATES** (stateID, costOffsets, priceOffset, productID, clientID)

Foreign key (productID) references PRODUCTS (productID)

Foreign key (clientID) references CLIENTS (clientID)

**TEAMS** (teamID, leadID, teamName, siteID)

Foreign key (leadID) references EMPLOYEES (empID)

Foreign key (siteID) references SITES (siteID)

**TRAINING\_SESSIONS** (sessionID, startDate, endDate, sLocation, productID, instructionID, courseID)

Foreign key (productID) references PRODUCTS(productID)

Foreign key (instructorID) references EMPLOYEES(empID)

Foreign key (courseID) references REFRESHER\_COURSES(courseID)

**ZIPCODES** (zipcode, areaID)

Foreign key (areaID) references AREAS (areaID)

### Other Relationships, Weak Entity Classes

**AREAS\_CSRS** (areaID, csrID)

Foreign key (csrID) references EMPLOYEES (empID)

Foreign key (areaID) references AREAS (areaID)

**BRANDS** (brandID, brandName, brandManagerID)

Foreign key (brandManagerID) references EMPLOYEES (empID)

**BRANDS\_FORMS** (brandID, productID)

Foreign key (brandID) references BRANDS (brandID)

Foreign key (productID) references PRODUCTS (productID)

**ORDER\_PRODUCTS** (productID, orderID, quantity, linePrice, lineDiscount, lineTotal)

Foreign key (productID) references PRODUCTS (productID)

Foreign key (orderID) references ORDERS (orderID)

**PRODUCT\_LINES** (plID, plname, startDate)

**PRODUCT\_LINES\_FORM** (productID, pName)

Foreign key (productID) references PRODUCTS (productID)

Foreign key (pName) references PRODUCT\_LINES (pName)

**REGIONS** (regionID, regionname, managerID)

Foreign key (managerID) references EMPLOYEES (empID)

**REGIONS\_FORMS** (regionID, officeNumber)

Foreign key (regionname) references REGIONS

Foreign key (officeNumber) references OFFICES

**SESSIONS\_STUDENTS** (sessionID, studentID)

Foreign key (sessionID) references TRAINING SESSIONS (sessionID)

Foreign key (studentID) references EMPLOYEES (empID)

**VISITS**(visitID, csrID, clientID, date)

Foreign key (csrID) references CLIENT\_SERV\_REPS

Foreign key (clientID) references CLIENTS

### Data Dictionary

| Schema Construct | Data Type | Constraint |
| --- | --- | --- |
| AREAS | Relation representing the entity class AREAS | |
| ▪ areaID | varchar2(4) | Primary Key |
| ▪ areaName | varchar2(30) | Unique |
| ▪ stateID | varchar2(2) | Foreign Key references STATES |
| *FD*: areaID → areaID, areaName | | |
| *FD*: areaName → areaID, areaName | | |
|  |  |  |
| AREAS\_CSRS | Relation representing the entity class AREAS\_CSRS | |
| ▪ areaID | varchar2(4) | Foreign Key references AREAS |
| ▪ csrID | varchar2(4) | Foreign Key references CLIENT\_SERV\_REPS |
| Primary Key Constraint: areaID, csrID | | |
| *FD*: areaID, csrID → areaID, csrID | | |
|  |  |  |
| BRANDS | Relation representing the entity class BRANDS | |
| ▪ brandID | char(5) | Primary Key |
| ▪ brandName | varchar2(25) | Unique |
| ▪ brandManagerID | char(4) | Foreign Key references EMPLOYEES |
| *FD*: brandID → brandID, brandName, brandManagerID | | |
| *FD*: brandName → brandID, brandName, brandManagerID | | |
|  |  |  |
| BRANDS\_FORMS | Relation representing the entity class BRANDS\_FORMS | |
| ▪ brandID | char(5) | Foreign Key referernces BRANDS |
| ▪ productID | varchar2(5) | Foreign Key referernces PRODUCTS |
| Primary Key Constraint: brandID, productID | | |
| *FD:* brandID, productID → brandID, productID | | |
|  |  |  |
| CLIENT\_SERV\_REPS | Relation representing the entity class CLIENT\_SERV\_REPS | |
| ▪ csrID | char(4) | Foreign Key, references EMPLOYEES |
| ▪ seniorityRank | varchar2(10) |  |
| ▪ csrStatus | varchar2(10) |  |
| ▪ commission | number(2) | Expressed as percentage |
| ▪ bonus | number(5) | Expressed in dollars |
| ▪ teamID | char(4) | Foreign Key, references TEAMS |
| Primary Key Constraint: csrID | | |
| *FD*: csrID → csrID, seniorityRank, csrStatus, commission, bonus | | |
| *FD*: csrID, teamID → csrID, seniorityRank, csrStatus, commission, bonus, teamID | | |
|  |  |  |
| CLIENTS | Relation representing the entity class CLIENTS | |
| ▪ clientID | char(5) | Primary Key |
| ▪ clientName | varchar2(30) | Unique |
| ▪ IsActive | char(1) |  |
| ▪ discount | number(5,4) |  |
| ▪ commissionRate | number(4,2) |  |
| ▪ pcTitle | varchar2(40) |  |
| ▪ pcName | varchar2(40) |  |
| ▪ pcPhone | number(10) |  |
| ▪ pcEmail | varchar2(40) |  |
| ▪ primarySiteID | char(4) | Foreign Key references SITES |
| *FD*: clientID → clientID, clientName, IsActive, discount, commissionRate, pcTitle, pcName, pcPhone, pcEmail, primarySiteID | | |
| *FD*: clientName→ clientID, clientName, IsActive, discount, commissionRate, pcTitle, pcName, pcPhone, pcEmail, primarySiteID | | |
|  |  |  |
| CSR\_ASSIGNMENTS | Relation representing the entity class CSR\_ASSIGNMENTS | |
| ▪ csrID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ start\_date | date |  |
| ▪ end\_date | date |  |
| ▪ areaID | varchar2(4) | Foreign Key references AREAS |
| ▪ clientID | char(5) | Foreign Key references CLIENTS |
| Primary Key Constraint: csrID, areaID, clientID | | |
| *FD*: csrID, areaID, clientID → csrID, areaID, clientID, start\_date, end\_date | | |
|  |  |  |
| CSR\_SPECIALITY | Relation representing the entity class CSR\_SPECIALITY | |
| ▪ csrID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ pl\_ID | char(5) | Foreign Key references PRODUCT LINES |
| Primary Key Constraint: csrID, pl\_ID | | |
| *FD*: csrID, pl\_ID → csrID, pl\_ID | | |
|  |  |  |
| EMPLOYEE\_HISTORY | Relation representing the entity class EMPLOYEE\_HISTORY | |
| ▪ empID | char(4) | Foreign Key, references EMPLOYEES |
| ▪ start\_date | date |  |
| ▪ end\_date | date |  |
| ▪ title | varchar2(25) |  |
| ▪ salary | number(8,2) |  |
| Primary Key Constraint: empID, start\_date | | |
| *FD*: empID, start\_date → empID, start\_date, end\_date, title, salary | | |
|  |  |  |
| EMPLOYEE\_TRAJECTORY\_REPORT | Relation representing the entity class EMPLOYEE\_HISTORY | |
| ▪ empID | char(4) | Foreign Key, references EMPLOYEES |
| ▪ numRaises | number(38,0) |  |
| ▪ dayssincelastraise | number(38,0) |  |
| ▪ currenttitle | varchar2(25) |  |
| ▪ currentSalary | number(8,2) |  |
| ▪ lastIncreasePercentage | number(6,2) |  |
| ▪ varianceagainstpeerspercentage | number(6,2) |  |
| Primary Key Constraint: empID | | |
| *FD*: empID,→ empID, numRaises, dayssincelastraise, currenttitle, currentSalary, lastincreasePercentage, varianceagainstpeerspercentage | | |
|  |  |  |
| EMPLOYEES | Relation representing the entity class EMPLOYEES | |
| ▪ empID | char(4) | Primary Key |
| ▪ firstName | varchar2(25) |  |
| ▪ middleName | varchar2(25) |  |
| ▪ lastName | varchar2(25) |  |
| ▪ SSN | char(9) | Unique |
| ▪ streetName | varchar2(50) |  |
| ▪ city | varchar2(25) |  |
| ▪ state | varchar2(2) |  |
| ▪ zip | char(5) |  |
| ▪ hiredate | date |  |
| ▪ termdate | date |  |
| ▪ type | varchar2(25) |  |
| ▪ title | varchar2(25) |  |
| ▪ paygroup | varchar2(25) |  |
| ▪ salary | decimal(8,2) |  |
| *FD:*empID → empID, firstName, middleName, LastName, SSN, streetName, city, state, zip, hiredate, termdate, type, title, paygroup, salary | | |
| *FD*: SSN → empID, firstName, middleName, LastName, SSN, streetName, city, state, zip, hiredate, termdate, type, title, paygroup, salary | | |
|  |  |  |
| FEEDBACK | Relation representing the entity class FEEDBACK | |
| ▪ feedbackID | char(5) | Primary Key |
| ▪ comments | varchar2(1000) |  |
| ▪ fDate | date |  |
| ▪ csrID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ productID | char(5) | Foreign Key references PRODUCTS |
| ▪ clientID | char(5) | Foreign Key references CLIENTS |
| *FD*: feedbackID → feedbackID, comments, fDate, csrID, productID, clientID | | |
|  |  |  |
| GENERAL\_PROMOTIONS | Relation representing the entity class GENERAL\_PROMOTIONS | |
| ▪ gpID | char(6) | Primary Key |
| ▪ productID | char(5) | Foreign Key references PRODUCTS |
| ▪ brandManagerID | char(4) | Foreign Key references EMPLOYEES |
| *FD*: gpID → gpID, productID, brandManagerID | | |
|  |  |  |
| INCIDENTS | Relation representing the entity class INCIDENTS | |
| ▪ incidentNum | char(5) | Primary Key |
| ▪ dateOccured | date |  |
| ▪ dateReceived | date |  |
| ▪ dateReviewed | date |  |
| ▪ description | varchar2(1000) |  |
| ▪ comments | varchar2(1000) |  |
| ▪ resolutions | varchar2(500) |  |
| ▪ managerID | char(4) | Foreign Key references EMPLOYEES |
| ▪ csrID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ clientID | char(5) | Foreign Key references CLIENTS |
| *FD*: incidentNum → incidentNum, dateOccured, dateReceived, dateReviewed, description, comments, resolutions, managerID, csrID, clientID | | |
|  |  |  |
| OFFICES | Relation representing the entity class OFFICES | |
| ▪ officeNum | varchar2(3) | Primary Key |
| ▪ officeType | varchar2(10) |  |
| ▪ officeName | varchar2(30) | Unique |
| ▪ regionID | varchar2(3) | Foreign Key references REGIONS |
| *FD*: officeNum → officeNum, officeType, officeName, regionID | | |
| *FD*: officeName → officeNum, officeType, officeName, regionID | | |
|  |  |  |
| ORDER\_PRODUCTS | Relation representing the entity class ORDER\_PRODUCTS | |
| ▪ orderID | char(5) | Foreign Key references ORDERS |
| ▪ productID | char(5) | Foreign Key references PRODUCTS |
| ▪ item\_Discount | number(2,2) |  |
| ▪ quantity | number(3) |  |
| ▪ line\_total | number(6,2) |  |
| Primary Key Constraint: orderID, productID | | |
| *FD*: orderID, productID → orderID, productID, lineDiscount, TotalTotal | | |
|  |  |  |
| ORDERS | Relation representing the entity class ORDERS | |
| ▪ orderID | char(5) | Primary Key |
| ▪ clientID | char(5) | Foreign Key references CLIENTS |
| ▪ csrID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ shipDate | date |  |
| ▪ expectedDelivery | date |  |
| ▪ shipMethod | varchar2(10) |  |
| ▪ totalDiscount | number(2) |  |
| ▪ totalPrice | number(6,2) |  |
| ▪ siteID | char(4) | Foreign Key references SITEID |
| ▪ orderStatus | varchar2(20) |  |
| ▪ orderDate | date |  |
| *FD*: orderID → orderID, clientID, csrID, shipDate, expectedDelivery, shipMethod, IsActive, lineDiscount, TotalTotal | | |
|  |  |  |
| PRODUCT\_LINES | Relation representing the entity class PRODUCT\_LINES | |
| ▪ pl\_ID | char(5) | Primary Key |
| ▪ plName | varchar2(30) | Unique |
| ▪ startDate | date |  |
| *FD*: pl\_ID → pl\_ID, plName, startDate | | |
| *FD*: plName → pl\_ID, plName, startDate | | |
|  |  |  |
| PRODUCT\_LINES\_ARE\_MANAGED | Relation representing the entity class PRODUCT\_LINES\_ARE\_MANAGED | |
| ▪ pl\_ID | char(5) | Foreign Key references PRODUCT\_LINES |
| ▪ pl\_managerID | char(4) | Foreign Key references EMPLOYEES |
| ▪ startDate | date |  |
| Primary Key Constraint: pl\_ID, pl\_managerID | | |
| *FD*: pl\_ID , pl\_managerID→ pl\_ID, pl\_managerID | | |
|  |  |  |
| PRODUCT\_LINES\_FORM | Relation representing the entity class PRODUCT\_LINES\_FORM | |
| ▪ pl\_ID | char(5) | Foreign Key references PRODUCT\_LINES |
| ▪ productID | char(5) | Foreign Key references PRODUCTS |
| Primary Key Constraint: pl\_ID, productID | | |
| *FD*: pl\_ID, productID → pl\_ID, prodcutID | | |
|  |  |  |
| PRODUCT\_LINES\_NOTES | Relation representing the entity class PRODUCT\_LINES\_NOTES | |
| ▪ pl\_ID | char(5) | Foreign Key references PRODUCT\_LINES |
| ▪ note | varchar2(1000) |  |
| Primary Key Constraint: plID | | |
| *FD:* plID → pl\_ID, note | | |
|  |  |  |
| PRODUCTS | Relation representing the entity class PRODUCTS | |
| ▪ productID | char(5) | Primary Key |
| ▪ prodName | varchar2(25) | Unique |
| ▪ isActive | char(1) |  |
| ▪ baseCost | number(6,2) |  |
| ▪ prodGroups | varchar2(25) |  |
| *FD*: productID → productID, prodName, isActive, baseCost, prodGroups | | |
| *FD*: productName → productID, prodName, isActive, baseCost, prodGroups | | |
|  |  |  |
| REFRESHER\_COURSES | Relation representing the entity class REFRESHER\_COURSES | |
| ▪ courseID | char(5) | Primary Key |
| ▪ courseNum | number(5) |  |
| ▪ courseName | varchar2(50) |  |
| ▪ description | varchar2(250) |  |
| ▪ internalCost | number(6,2) |  |
| *FD*: courseID → courseID, courseNum, courseName, description, internalCost | | |
|  |  |  |
| REGIONS | Relation representing the entity class REGIONS | |
| ▪ regionID | varchar2(3) | Primary Key |
| ▪ regionName | varchar2(3) | Unique |
| ▪ managerID | char(4) | Foreign Key references EMPLOYEES |
| *FD*: regionID,→ regionID, regionName, managerID | | |
| *FD*: regionName → regionID, regionName, managerID | | |
|  |  |  |
| RESOLUTION\_MEETINGS | Relation representing the entity class RESOLUTION\_MEETINGS | |
| ▪ rmID | char(5) | Primary Key |
| ▪ incidentNum | char(5) | Foreign Key references INCIDENTS |
| ▪ rmDate | date |  |
| ▪ isCSRpresent | char(1) |  |
| ▪ isCSRreassigned | char(1) |  |
| *FD*: rmID → rmID, incidentNum, rmDate, isCSRpresent, isCSRreassigned | | |
|  |  |  |
| SESSIONS\_STUDENTS | Relation representing the entity class SESSIONS\_STUDENTS | |
| ▪ sessionID | char(5) | Foreign Key references TRAINING\_SESSIONS |
| ▪ studentID | char(4) | Foreign Key references EMPLOYEES |
| Primary Key Constraint: sessionID, studentID | | |
| *FD*: sessionID, studentID → sessionID, studentID | | |
|  |  |  |
| SITES | Relation representing the entity class SITES | |
| ▪ siteID | char(4) | Primary Key |
| ▪ clientID | char(5) | Foreign Key references CLIENTS |
| ▪ siteStreet | varchar2(25) |  |
| ▪ siteCity | varchar2(25) |  |
| ▪ siteState | varchar2(2) |  |
| ▪ siteZip | char(5) |  |
| ▪ siateID | varchar(2) | Foreign Key references STATES |
| *FD*: siteID → siteID, clientID, size, siteStreet, siteCity, siteState, siteZip | | |
|  |  |  |
| STATE\_PRODUCTS | Relation representing the entity class STATE\_PRODUCTS | |
| ▪ stateID | varchar2(2) | Foreign Key references STATES |
| ▪ productID | char(5) | Foreign Key references PRODUCTS |
| Primary Key Constraint: stateID, productID | | |
| *FD*: stateD, productID → stateID, productID | | |
|  |  |  |
| STATE\_PROMOTIONS | Relation representing the entity class STATE\_PROMOTIONS | |
| ▪ spID | char(6) | Primary Key |
| ▪ stateID | varchar2(2) | Foreign Key references STATES |
| ▪ startDate | date |  |
| ▪ endDate | date |  |
| ▪ budget | number(10,2) |  |
| ▪ gpID | char(6) | Foreign Key references GENERAL\_PROMOTIONS |
| *FD*: spID → spID, stateID, startDate, endDate, budget, gpID | | |
|  |  |  |
| STATES | Relation representing the entity class STATES | |
| ▪ stateID | varchar2(2) | Primary Key |
| ▪ stateName | varchar(15) | Unique |
| ▪ priceoffset | number(2,2) |  |
| ▪ regionID | varchar2(3) | Foreign Key references REGIONS |
| *FD*: stateID → stateID, stateName, priceoffset, regionID | | |
| *FD*: stateName → stateID, stateName, priceoffset, regionID | | |
|  |  |  |
| TEAMS | Relation representing the entity class TEAMS | |
| ▪ teamID | char(4) | Primary Key |
| ▪ leadID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ teamName | varchar2(25) | Unique |
| ▪ siteID | char(4) | Foreign Key,references SITES |
| *FD*: teamID → teamID, leadID, teamName | | |
| *FD*: teamName → teamID, leadID, teamName | | |
|  |  |  |
| TRAINING\_SESSIONS | Relation representing the entity class TRAINING\_SESSIONS | |
| ▪ sessionID | char(5) | Primary Key |
| ▪ startDate | date |  |
| ▪ endDate | date |  |
| ▪ slocation | varchar2(50) |  |
| ▪ productID | char(5) | Foreign Key references PRODUCTS |
| ▪ instructorID | char(4) | Foreign Key references EMPLOYEES |
| ▪ courseID | char(5) | Foreign Key references REFRESHER\_COURSES |
| *FD*: sessionID → sessionID, startDate, endDate, slocation, productID, instructorID, courseID | | |
|  |  |  |
| VISITS | Relation representing the entity class VISITS | |
| ▪ visitID | char(5) | Primary Key |
| ▪ csrID | char(4) | Foreign Key references CLIENT\_SERV\_REPS |
| ▪ siteID | char(4) | Foreign Key references SITES |
| ▪ vDate | date |  |
| *FD:* visitiID → visitID, csrID, siteID, vDate | | |

## Chapter 4: Queries and Views

---------------------------------------------

-- BRANDNAMES\_COSMETICS --

---------------------------------------------

/\* The query below displays the name of brands that  
carry 2 or more cosmetic products. The number of   
products, the name of the brand manager and their   
official title is also displayed. The list is put   
in decending order by number of cosmetic products.  
\*/  
CREATE VIEW BRANDNAMES\_COSMETICS AS  
SELECT DISTINCT  
 B.BRANDNAME AS "Brand Name",  
 COUNT(P.PRODGROUPS) AS "Number of Cosmetic Products",  
 E.FIRSTNAME || ' ' || E.LASTNAME AS "Brand Manager",  
 E.Title AS "Title"  
FROM   
 BRANDS B,   
 BRANDS\_FORMS BF,   
 PRODUCTS P,   
 EMPLOYEES E  
WHERE  
 B.BRANDID = BF.BRANDID  
AND BF.PRODUCTID = P.PRODUCTID  
AND B.BRANDMANAGERID = E.EMPID  
AND PRODGROUPS = 'Cosmetic'  
GROUP BY B.BRANDNAME,   
 E.FIRSTNAME,   
 E.LASTNAME,   
 E.TITLE  
HAVING   
 COUNT(P.PRODGROUPS) > 1  
ORDER BY   
 COUNT(P.PRODGROUPS) DESC;  
  
----------------------------------------

-- CSR\_BEHAVIOR\_STAND --

----------------------------------------

/\* The query below displays the CSR ID, the name of the  
CSR, the number of incidents since January 1, 2017, the   
behavior standard and the number of years as a CSR. Using   
a CASE, it displays the behavior standard of the CSR.   
\*/   
CREATE VIEW CSR\_BEHAVIOR\_STAND AS  
SELECT  
 C.CSRID AS "CSR ID",  
 FIRSTNAME || ' ' || LASTNAME AS "Customer Service Rep",  
 COUNT(I.CSRID) AS "Number of Incidents",  
 (CASE  
 WHEN COUNT(I.CSRID) >= 2 then 'Mediocre'  
 WHEN COUNT(I.CSRID) = 1 then 'Satifactory'  
 ELSE 'Outstanding'  
 END) AS "Behavior Standard",  
 TO\_CHAR((SYSDATE - HIREDATE)/365, '9.99') AS "Years as CSR"  
FROM  
 CLIENT\_SERV\_REPS C   
 JOIN EMPLOYEES ON C.CSRID = EMPID  
 LEFT JOIN INCIDENTS I ON C.CSRID = I.CSRID  
 AND I.DATEOCCURRED >= to\_date('01-Jan-17','DD-MON-YY')  
GROUP BY  
 C.CSRID,   
 FIRSTNAME,   
 LASTNAME,  
 HIREDATE  
ORDER BY  
 "Number of Incidents" DESC;   
  
---------------------------------------

-- CSRS\_OVER\_ASSIGNED --

---------------------------------------

/\*This query displays the CSR's who have five or more assignments and the states they're in, how their sales are affected. It uses the tables employees, areas,

areas\_csrs, and client\_serv\_reps via a natural join and uses an outer join to include total sales information. The results are filtered to only show CSRs with five

or more areas. A view has been created to show the results as needed to HR and managers for assignments.\*/

create view CSRs\_Over\_Assigned as

SELECT employees.EMPID,lastname, firstname, SENIORITYRANK, commission, sum(totalprice) "Sales", COUNT (DISTINCT AREAS\_CSRS.AREAID) "Areas Assigned", COUNT(DISTINCT STATEID) "States"

FROM EMPLOYEES join areas\_csrs

on employees.empid = areas\_csrs.CSR\_ID

join areas

on areas.AREAID = areas\_csrs.AREAID

join client\_serv\_reps

on areas\_csrs.CSR\_ID = client\_serv\_reps.CSRID

Left outer join orders

on orders.csrid = employees.empid

AND TYPE = 'Customer Service'

Having COUNT (DISTINCT AREAS\_CSRS.AREAID) >= 5

GROUP BY employees.EMPID,lastname, firstname,SENIORITYRANK, commission;

-----------------------------

-- PRODUCT\_PRICE --

-----------------------------

/\*This query calculates the product price for clients by taking the base price and calculating the state priceoffset and client discount to arrive

at the price before other discounts. It also calculates the percent increase on the base price. It joins together sites, states, clients, products, and state\_products using a natural join and is ordered by client.

A view created for this query, called Product\_price, will make this information available as needed for CSR's.\*/

create view product\_price as

select clientname,sites.stateid, Prodname "Product Name",cast(round(basecost,2) as DECIMAL(4,2)) "Basecost",round(SUM((basecost\*priceoffset)+(basecost\*discount)+basecost),2) "Final Price",

round(sum((((basecost\*priceoffset)+(basecost\*discount)+basecost)-basecost)/basecost),2) "% Increase"

from sites,state\_products,states, clients, products

where sites.stateid = state\_products.STATEID

and sites.stateid = states.stateid

and clients.clientid = sites.clientid

and products.productid = state\_products.productid

group by sites.StateID, basecost, Clientname, prodname

order by Clientname;

-------------------------

-- REGION\_INFO --

-------------------------

/\* The query below displays ALL region IDs and the the region  
names. It displays the number of offices and number of states  
in each region. Finally, it displays the full name of assigned  
region manager. If no manager has been assigned, then  
'to be determined' will show.  
\*/

CREATE VIEW REGION\_INFO AS  
SELECT  
 O.REGIONID AS "Region ID",  
 REGIONNAME AS "Region Name",  
 COUNT(distinct OFFICENAME) AS "Number of Offices",  
 COUNT(distinct STATEID) AS "Number of States",  
 (CASE  
 WHEN R.MANAGERID IS NULL then 'To Be Determined'  
 ELSE FIRSTNAME || ' ' || MIDDLENAME || ' ' || LASTNAME  
 END) AS "Region Manager Name"  
FROM REGIONS R  
 LEFT JOIN OFFICES O ON R.REGIONID = O.REGIONID  
 LEFT JOIN STATES S ON R.REGIONID = S.REGIONID  
 LEFT JOIN EMPLOYEES ON R.MANAGERID = EMPID  
GROUP BY  
 O.REGIONID,  
 REGIONNAME,  
 R.MANAGERID,  
 FIRSTNAME,  
 MIDDLENAME,  
 LASTNAME  
ORDER BY  
 O.REGIONID;

----------------------------------

-- TEAMNAMES\_SIZES --

----------------------------------

/\* The query below displays Team ID, Team Name,  
and the Team Lead. It also utilizes a subquery to display  
the size of the team (i.e.number of members). The query   
filters out any CSRs that have a termination date.  
\*/  
CREATE VIEW TeamNames\_Sizes AS  
SELECT T.TEAMID AS "Team ID",   
 T.TEAMNAME AS "Team Name",  
 E.FIRSTNAME || ' '|| E.LASTNAME AS "Team Lead",  
 (SELECT COUNT(TEAMID)  
 FROM CLIENT\_SERV\_REPS  
 WHERE CSRID IN (SELECT EMPID  
 FROM EMPLOYEES  
 WHERE TERMDATE IS NULL)   
 AND TEAMID = T.TEAMID) AS "Team Size"  
FROM TEAMS T  
 JOIN EMPLOYEES E ON T.LEADID = E.EMPID  
ORDER BY TEAMNAME;

-----------------------------------------------------------------------------------

-- Training Session extended details, with attendance figures --

-----------------------------------------------------------------------------------

/\* This SELECT query shows information about our training sessions, including their ID, their parent courses' name and description, the name of the product being taught, the dates and location of the session, and the name and title of the instructor. This is accomplished by joining together 4 tables. These are all natural joins.

In addition, we also show how many students completed each of these training sessions. This is accomplished with a subquery that counts the instances STUDENTID for the session in question. \*/

SELECT SESSIONID, COURSENAME, DESCRIPTION, PRODNAME, STARTDATE, ENDDATE, SLOCATION, FIRSTNAME, LASTNAME, TITLE, (

SELECT COUNT (STUDENTID) FROM SESSIONS\_STUDENTS WHERE SESSIONID = ts.SESSIONID) AS "Attendees"

FROM

REFRESHER\_COURSES JOIN TRAINING\_SESSIONS ts ON REFRESHER\_COURSES.COURSEID = ts.COURSEID JOIN EMPLOYEES ON ts.INSTRUCTORID = EMPLOYEES.EMPID JOIN PRODUCTS ON ts.PRODUCTID = PRODUCTS.PRODUCTID;

-------------------------------

-- V\_CLIENT\_SALES --

------------------------------

/\* This query pulls active clients and ranks them by their total sales and by their total orders based on the volume of items ordered. I used an outer join so that even clients who haven't purchased anything yet are listed.

I saved this query as a view so it will be easy for CSRs or other staff to quickly look up our top clients.\*/

create or replace view v\_client\_sales AS (

SELECT C.CLIENTNAME, to\_char(coalesce(sum(O.TOTALPRICE),0),'$999,999.99') as sales, rank () over (order by coalesce(sum(O.TOTALPRICE),0) desc)

as rank\_by\_sales, to\_char(coalesce(sum(OP.QUANTITY),0),'99999') as sales\_volume, rank () over (order by coalesce(sum(OP.QUANTITY),0) desc) as rank\_by\_volume

from clients c left outer join orders o

on (c.clientid=o.clientid) left outer join order\_products op on (o.orderid=op.orderid)

where C.ISACTIVE = 'Y' --and O.ORDERSTATUS != 'Canceled' -- if add this it excludes all clients who haven't ordered anything yet

group by c.clientname)

ORDER BY rank () over (order by coalesce(sum(O.TOTALPRICE),0) desc)

;

-------------------------------

-- VIEW\_AREA\_EXP --

-------------------------------

/\*  
This query aggregates CSR experience by area served but  
filters based on comparative average.It features subquery  
and aggregation and is sorted.  
\*/  
  
CREATE VIEW view\_area\_exp AS  
SELECT b.areaname AS "Area", to\_char(sum (months\_between(sysdate,hiredate)),999) AS "Total Experience"  
FROM areas\_csrs a, employees e, areas b  
WHERE a.csr\_id = e.empid  
AND a.areaid = b.areaid  
GROUP BY b.areaname  
HAVING AVG (months\_between(sysdate,hiredate)) < (SELECT   
 AVG(months\_between(sysdate,hiredate))  
 FROM employees)  
ORDER BY areaname;

----------------------------

-- VIEW\_CSR\_SAL --

---------------------------

/\*  
This query selects Customer Service Reps from the areas\_csrs  
table who support more than one area, and ranks them by salary  
from the employees table. It features subquery and aggregation.  
\*/  
  
CREATE VIEW view\_csr\_sal AS  
SELECT e.empid AS "Employee ID", e.firstname AS "First Name", e.lastname AS "Last Name", e.salary AS "Salary"  
FROM employees e  
WHERE e.empid IN  
(SELECT csr\_id  
FROM areas\_csrs  
GROUP BY csr\_id  
HAVING COUNT(csr\_id) > 1)  
ORDER BY e.salary;

## Chapter 5: Triggers and Procedures

### Procedure 1

The employee\_trajectory\_rpt\_proc stored procedure creates report information for each employee about their career trajectory with TriannaCorp.

As a "lifelong employer", we take a keen interest in making sure that our employees progress in their careers. This report could be used by HR and managers to evaluate employee progress and address equity issues

For each employee, we record:

- The number of raises they've received in their career

- The number of days (from today's date) that have passed since their last raise. If they haven't received a raise, this will be 0

- Their current title and salary

- The size of their last salary adjustment, expressed as a percentage against their previous salary

- The variance of their salary against peers with the same title. If the percentage is greater than 0, they make more than their peers; if less, they make less. The average is based on current salaries only, not historical salaries.

This procedure runs with no arguments, and generates a report entry for any employee for whom we have employee history data.

CREATE OR REPLACE PROCEDURE employee\_trajectory\_rpt\_proc

IS

------------------ DECLARE VARIABLES ---------------------------

-- VARIABLES FOR OUR EVENTUAL OUTPUT

--vo\_empid EMPLOYEE\_TRAJECTORY\_REPORT.EMPID%type;

vo\_numraises EMPLOYEE\_TRAJECTORY\_REPORT.NUMRAISES%type;

vo\_dayssinceraise EMPLOYEE\_TRAJECTORY\_REPORT.DAYSSINCELASTRAISE%type;

vo\_title EMPLOYEE\_TRAJECTORY\_REPORT.CURRENTTITLE%type;

vo\_salary EMPLOYEE\_TRAJECTORY\_REPORT.CURRENTSALARY%type;

vo\_increasepct EMPLOYEE\_TRAJECTORY\_REPORT.LASTINCREASEPERCENTAGE%type;

vo\_variancepct EMPLOYEE\_TRAJECTORY\_REPORT.VARIANCEAGAINSTPEERSPERCENTAGE%type;

-- INTERNAL HELPER VARIABLES

/\* Some dates for working with dates between stuff \*/

vv\_recentdate DATE;

vv\_recentsalary EMPLOYEE\_TRAJECTORY\_REPORT.CURRENTSALARY%type;

vv\_prevdate DATE;

vv\_prevsalary EMPLOYEE\_TRAJECTORY\_REPORT.CURRENTSALARY%type;

vv\_avgsalary EMPLOYEE\_TRAJECTORY\_REPORT.CURRENTSALARY%type; -- for storing the average salary, for all current salaries, of a given job title.

vv\_upsert INTEGER; -- For our "upsert" combo insert into/update pattern for error-proofing inserts/updates.

BEGIN

FOR xe IN (SELECT DISTINCT EMPID FROM EMPLOYEE\_HISTORY ORDER BY EMPID) LOOP -- for each employee who has a history, exclusively

-- Set some defaults for our variables

vo\_numraises := 0;

vo\_dayssinceraise := 0; -- This is reasonable if they've never got a raise from us.

vo\_title := '';

vo\_salary := 0;

vo\_increasepct := 0; -- This is reasonable if they've never got a raise.

vo\_variancepct := 1; -- This is reasonable if they have exactly the average.

-- Figure out number of raises, store in vo\_numraises

--- This is bad logic, because if they returned to the same salary later, it wouldn't be distinct... This will hold place until we can come back and do something more complicated.

SELECT COUNT(DISTINCT SALARY) INTO vo\_numraises FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID;

-- Figure out number of days since last salary increase, store in vo\_dayssinceraise

SELECT END\_DATE INTO vv\_recentdate FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID); -- date of last employee history record.

SELECT SALARY INTO vv\_recentsalary FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID); -- salary during the most recent employee record.

SELECT END\_DATE INTO vv\_prevdate FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND SALARY != vv\_recentsalary AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND END\_DATE < vv\_recentdate); -- date of salary change prior to the most recent record.

SELECT SALARY INTO vv\_prevsalary FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND SALARY != vv\_recentsalary AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND END\_DATE < vv\_recentdate); -- salary prior to the most recent record.

---- Let's do the salary calcs as well while we have the numbers.

-- Calculate last raise as a percentage of their previous salary, store in vo\_increasepct

--DBMS\_OUTPUT.PUT\_LINE(' xe.EMPID: ' || xe.EMPID || ' vv\_recentsalary: ' || vv\_recentsalary || ' vv\_recentdate: ' || vv\_recentdate || ' vv\_prevsalary: ' || vv\_prevsalary || ' vv\_prevdate: ' || vv\_prevdate);

IF(vv\_recentdate > vv\_prevdate) THEN -- then their most recent record is a raise.

SELECT TRUNC(sysdate) - vv\_recentdate INTO vo\_dayssinceraise FROM dual;

ELSE

SELECT TRUNC(sysdate) - vv\_prevdate INTO vo\_dayssinceraise FROM dual;

END IF;

---- Let's do the salary calcs as well while we have the numbers handy.

-- Calculate last raise as a percentage of their previous salary, store in vo\_increasepct

SELECT ROUND(((vv\_recentsalary / vv\_prevsalary)\*100)-100,1) INTO vo\_increasepct FROM dual;

-- Determine current title, store in vo\_title

SELECT TITLE INTO vo\_title FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID);

-- Determine current salary, store in vo\_salary

SELECT SALARY INTO vo\_salary FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE EMPID=xe.EMPID);

-- Calculate salary variance against peers (with same title), store in vo\_variancepct

--SELECT AVG(SALARY) FROM EMPLOYEE\_HISTORY eh WHERE TITLE='SR. BRANCH MANAGER' AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE TITLE='SR. BRANCH MANAGER' AND EMPID=eh.EMPID);

SELECT AVG(SALARY) INTO vv\_avgsalary FROM EMPLOYEE\_HISTORY eh WHERE TITLE=vo\_title AND END\_DATE = (SELECT MAX(END\_DATE) FROM EMPLOYEE\_HISTORY WHERE TITLE=vo\_title AND EMPID=eh.EMPID); -- We only want the latest salaries to compose the average.

--DBMS\_OUTPUT.PUT\_LINE(' vv\_avgsalary: ' || vv\_avgsalary);

SELECT ROUND((vo\_salary/vv\_avgsalary)\*100,0) INTO vo\_variancepct FROM DUAL;

--DBMS\_OUTPUT.PUT\_LINE(' xe.EMPID: ' || xe.EMPID || ' vo\_numraises: ' || vo\_numraises || ' vo\_dayssinceraise: ' || vo\_dayssinceraise || ' vo\_title: ' || vo\_title || ' vo\_salary: ' || vo\_salary || ' vo\_increasepct: ' || vo\_increasepct || ' vo\_variancepct: ' || vo\_variancepct); -- DEBUG OUTPUT!

SELECT COUNT(\*) INTO vv\_upsert FROM EMPLOYEE\_TRAJECTORY\_REPORT WHERE EMPID = xe.EMPID;

IF (vv\_upsert > 0) THEN -- row exists; do an update, not an insert

UPDATE EMPLOYEE\_TRAJECTORY\_REPORT SET NUMRAISES = vo\_numraises, DAYSSINCELASTRAISE = vo\_dayssinceraise, CURRENTTITLE = vo\_title, CURRENTSALARY = vo\_salary, LASTINCREASEPERCENTAGE = vo\_increasepct, VARIANCEAGAINSTPEERSPERCENTAGE = vo\_variancepct WHERE EMPID = xe.EMPID; -- UPDATE to an existing employee trajectory report record.

ELSE -- row does not exist; perform an insert.

INSERT INTO EMPLOYEE\_TRAJECTORY\_REPORT (EMPID, NUMRAISES, DAYSSINCELASTRAISE, CURRENTTITLE, CURRENTSALARY, LASTINCREASEPERCENTAGE, VARIANCEAGAINSTPEERSPERCENTAGE) VALUES(xe.EMPID, vo\_numraises, vo\_dayssinceraise, vo\_title, vo\_salary, vo\_increasepct, vo\_variancepct); -- INSERT a new record.

END IF;

END LOOP;

END employee\_trajectory\_rpt\_proc;

/

--SHOW ERRORS;

EXECUTE employee\_trajectory\_rpt\_proc;

### Procedure 2

/\*  
The Salary Compare Procedure is a tool for Human Services to quickly get a  
picture of comparative salaries by employee type. The purpose would be to  
rapidly evaluate compensation against competitors for advantage in hiring.  
Because salaries are largely dependent on time with the company in addition  
to position, a date can be entered to limit how far back to search by hiring  
date. The data is output to a file SalaryCompareReport, which is overwritten  
each time the proc is run. The number of employees sampled is also displayed.  
\*/  
  
  
CREATE OR REPLACE PROCEDURE   
 salary\_compare\_procedure   
 (temp\_year IN number)  
-- create proc with year input  
  
AS  
  
 temp\_type employees.type%type;  
 temp\_avg SalaryCompareReport.AVG\_Salary%type;  
 temp\_num SalaryCompareReport.Employee\_Count%type;  
--declare variables  
  
 CURSOR cursor1  
 IS  
 SELECT distinct type FROM employees;  
--collect employee types  
  
 CURSOR cursor2  
 IS  
 SELECT COUNT(type)   
 FROM employees   
 WHERE type = temp\_type  
 AND EXTRACT(YEAR FROM hiredate) >= temp\_year  
 GROUP BY type;  
--collect employee count by type & year  
   
 CURSOR cursor3  
 IS  
 SELECT AVG(salary)   
 FROM employees   
 WHERE type = temp\_type  
 AND EXTRACT(YEAR FROM hiredate) >= temp\_year;  
--calculate average salary by type & year  
  
 BEGIN  
  
 DELETE FROM SalaryCompareReport;  
--clear table  
  
 FOR temploop IN cursor1 LOOP  
 temp\_type := temploop.type;  
--loop through employee types  
  
 OPEN cursor2;  
 FETCH cursor2 INTO temp\_num;  
 IF temp\_num IS NULL THEN  
 temp\_num := 0;  
 END IF;  
 CLOSE cursor2;  
--store count, set to 0 if null  
  
 OPEN cursor3;  
 FETCH cursor3 INTO temp\_avg;  
 IF temp\_avg IS NULL THEN  
 temp\_avg := 0;  
 END IF;  
 CLOSE cursor3;  
--store sum, set to 0 if null  
  
 INSERT INTO SalaryCompareReport VALUES (temp\_type, temp\_year, temp\_avg, temp\_num);  
 END LOOP;  
--send values to report table  
   
 COMMIT;  
  
END salary\_compare\_procedure;  
/

## Chapter 6: Web Interface Walkthrough

To show the power of our enterprise database system, we have developed a demonstration website that TriannaCorp could use to interact our data and systems.

**Our web interface is available at this URL:**

[**http://mis-navys17-tma.catnet.arizona.edu/index.aspx**](http://mis-navys17-tma.catnet.arizona.edu/index.aspx)

Note: Because it is hosted on an internal-only network, this demo site does not contain any credential information. An access and authorization layer should be implemented as part of further development.

Index page

The site index page, at <http://mis-navys17-tma.catnet.arizona.edu/index.aspx> provides the default entry point for our users. Here users could navigate to the queries, reports and data entry pages that they require to perform their duties.

The links on these pages don’t demonstrate the full power or feature coverage of the system, but instead show may be possible with future investment.

We’ve split up our demo site into 2 main themes, Views/Reports and Data Management.

### Reports and Views

Our reports and views area is where we can show the power of our enterprise data system to present data in way that are valuable in making decisions. These pages are home to our queries, which extract data for presentation from our database. These pages could also display reporting functionality we have programmed into the database via procedures, using our business rules and goals as an organization.

We chose to demonstrate the following:

* Areas/experience - Displays a list of our areas and the number of person-months the CSRs who work in them possess.
* Brands with Cosmetics - Displays brands that carry two or more cosmetic products.
* Client Sales - Displays a ranking of clients by their total sales with us. We show clients even if we hadn’t closed a sale with them yet.
* Client Service Representatives Behavior Standards - Shows CSRs and their level of behavioral standard based on incidents reported.
* Client Service Representatives Overassigned - Presents a list of CSRs who are assigned to too many areas, the number of states they’re assigned to, and their sales.
* Employee Trajectory Report - Displays information about an employee’s progression through their career with the company, given their historical employee records.
* Product Price - On this page, we show each client’s discounts after taking in their client-specific discount and their by-state and by-product price offsets. We also calculate the percentage of price increase overall.
* Salary Comparison Report - A report of average salary by employee type, with some metadata about the extent of the calculations.
* Training Session information - On this page, we show Training Session information, and show details for which Refresher Course is being instantiated, which product the session is about, which employees are teaching it, their title.

### Data Management

In our data management area, users can interact directly with the system to perform “CRUD”, or Create, Read, Update or Delete, operations directly on our data. These pages are useful for providing simple methods for data entry and management, fixing errors with raw data, and to provide users with a simple and robust method for manual data loading. These may be most useful for the more elemental and basic units of data in our system.

In a production system, access to these tables would be bound to users with certain roles in our organization, and would fit into access layers and authorization schemes for the entire site. But in our demonstration system, these aspects are not yet implemented.

We have decided to show the following entities in our data management area:

* Clients - Our clients who we do business with.
* Employee History - Time-series records of employee title and salary information.
* Offices - For our local and main branch offices, and our headquarters.
* Order Products - The products that are in orders, with quantity and any applicable discounting.
* Product Lines - The lines that our products fit into.
* Regions - Geographic extents that we have offices in.
* States - States that our business operates in.

### Neat!

Be sure to check out the following impressive things in our system:

* Area/experience page: This output is generated by a View that is saved in our database.
* Client Sales page: This output is generated by a View that is saved in our database.
* CSR Behavior Standard page: This output is generated by a View that is saved in our database. The view uses a CASE statement.
* CSR Behavior Standards page. This output is generated by a View that is stored in our database.
* CSRs Overassigned: This output is generated by a View that is stored in our database.
* Employee Trajectory: This is created with a stored procedure that uses historical employee data.
* Order Products page: This provides for insertion and updating of a weak entity class.
* Product Price page: This is output generated by a View that is saved in our database.
* The employee trajectory report is produced by a stored procedure.
* The salary comparison report is produced by a stored procedure.
* In most of the report and views pages, we have worked to format output in more readable formats for columns that show currency or percentages. We have generally done this in the view, but for some outputs we have elected to do it in the web presentation layer.

## Chapter 7: Conclusions, Challenges, and Next Steps

### Methodology

For this project, the team met with local and main branch offices, as well as staff in the home office, to understand the needs for both everyday business and ongoing reporting and planning. Using their information, we were able to create a prototype schema for the company and populate with a small data set. The team also created triggers for ensuring identifier uniqueness and added constraints for referential integrity. We also created procedures to create specialized reports in an automated fashion. With a design based on stakeholder input, we created a basic website to demonstrate the ease of data entry and to display some frequently requested data and reports.

While excited about the possibilities for a more integrated data system, some expressed concerns about giving up control of “their” data and learning a new system. Because TriannaCorp has a dispersed organization, where offices maintain a great deal of autonomy, we expect these concerns will be company-wide. It is clear from the small data sample, we will also need to review data standards throughout our organization.

### Current Challenges

After speaking with employees at all levels, it is clear many hours are spent inputting data into a variety of systems, and validating submissions to main branches and to the home office, where additional time is needed to stitch information together into reports for financial and planning purposes. Anytime a person transfers to a new office, they need to be trained on the new data systems. Offices have differing formats and data entry standards, making it difficult to compile information into a coherent whole and errors are common. Management and client service representatives feel they do not have the information they need to make timely decisions for inventory and customer satisfaction. Managing paper copies and spreadsheets is difficult and time-consuming, with problems with version control and broken links. The differing systems also make IT support challenging and requires multiple types of hardware and software to maintain them. Finally, the company has years of records that will need to be transferred to any new data management system.

### Advantages

TriannaCorp prides itself on being a long-term employer and has an excellent institutional memory. This means people at all levels of the company have fulfilled many different jobs in different regions and have a solid knowledge of their systems and their needs. In essence, offices have tested many different data management systems on small scales. The team will be able to use that information to test them on a larger scale to find the best fit the company. In spite of general concerns regarding complexity and control, the team found people are enthusiastic about a system that will grow with the company and feel them to spend more time on core duties rather than paper management.

TriannaCorp already has a system in place for regular employee training and a culture of learning. We will be able to leverage the regular training and refresher courses for the initial rollout and later upgrades.

While there will be start-up costs involved with setup and training, the IT team believes the current personnel will be enough for ongoing maintenance and development. TriannaCorp will also be able to negotiate a better deal for a company-wide implementation, creating additional savings.

### Next Steps

The team believes the prototype is a success. Creating and implementing the full version will be a long-term project. Here is an overview of the steps we need to move forward:

1. **Company-wide needs and standards assessment:** While regions have different cultures and product needs, the team believes the basic reports for inventory, pricing, orders and personnel should be consistent. We will be able to use the prototype schema to scale to the entire company. As part of the on-going assessment, we can set data and formatting standards that will make the future adoption of specialized sales, inventory, and human resource systems easier. Some base systems may need to be retooled to better integrate into a company-wide solution.

Continued requirements gathering and project updates will help keep people invested in the final outcome.

1. **Schedule and budget:** In order to do this project correctly, a realistic timeline needs to be created. Historical information is important to the company, but the team believes focusing on the current and prior few years of data should be the focus. The team expects the full implementation take a year. A draft schedule and software quote are included in our next section. The team recommends obtaining at least two more quotes before deciding on a final vendor; we may want to engage a consultancy with experience in database implementations to review our plans and offer services. Current staff will need to be reassigned and additional short-term hiring may be needed for the project.

Digitizing and entering older records will be time consuming and expensive. After the most current information is entered, the board of directors and management will need to identify the next priorities for entry as time and budget permits.

1. **Model expansion and design:** The original entity model was created using a small sample of offices. Further revisions may be needed as it is expanded to the entire company.
2. **Implementation and testing:** Implementation of the database can be done in stages by region, with on-site testing by end users. Changes and updates can be implemented as each region is brought into the plan.
3. **Training:** As the system is implemented in stages, the company can take advantage of the training system in place for both the local IT teams and end users. A large part of training will be dedicated to helping employees recreate their regular reports in the new system. By doing this, we will be able to demonstrate the time-savings and responsiveness of the more centralized data management solution.
4. **Debriefing and reporting**: At the end of the project, the team plans to report back to the board of directors and the company as a whole on lessons learned and future updates.

### Estimated Costs and Schedule

#### **Estimated Costs** (from “Oracle Technology Global Price List”)

|  |  |
| --- | --- |
| **Oracle RDBMS Enterprise Edition** | $47,500.00 |
|  |  |
| **Enterprise Edition Options** |  |
| Partitioning | $2,530.00 |
| Real Application Testing | $2,530.00 |
| Database In-Memory | $5,060.00 |
|  |  |
| **Database Enterprise Management** |  |
| Diagnostics Pack | $7,500.00 |
|  |  |
| Secure Backup | $3,500.00 |
|  |  |
| **Application Server Products** |  |
| WebLogic Server Enterprise Edition | $25,000.00 |
|  |  |
| **Data Integration Technology** |  |
| Data Integration Suite | $70,000.00 |
|  |  |
| **Database Enterprise Management** |  |
| Diagnostics Pack | $7,500.00 |
| Tuning Pack | $5,000.00 |
|  |  |
| **Application Testing** |  |
| Load Testing Developer Edition | $1,760.00 |

#### Schedule

|  |  |
| --- | --- |
| **Requirements** |  |
| Interviews | 2 months |
| Site Visit | 2 months |
| **Conceptual Model** |  |
| Entity Relationship Diagram | 2-4 weeks |
| Relational Schema, Table Design | 3-4 weeks |
| **Hardware Procurement** |  |
| Installation | 2-4 weeks |
| Testing | 6-10 weeks |
| **Database Design** |  |
| Table Install | 1 week |
| SQL Install (queries, procs, etc) | 2-4 weeks |
| **Testing** |  |
| Data loading | 2 months |
| Test scripts & queries | 4 months |
| **Training** |  |
| End-user classes | 4 months |
| IT maintenance training | 2 weeks |

### Final thoughts

The team understands creating a database system for the company is a long-term project that will affect short-term sales, inventory and personnel activities but we believe it would be a valuable first step that will make the future goals easier and less costly to implement and manage.

If one thinks in terms of a pyramid, a database system would give the company a solid foundation managing current needs and future expansion. Employees company-wide will spend less time managing data and more time using it to make timely decisions for clients and new products.

## Bibliography

"Oracle Technology Global Price List" *Oracle Corporation*, 31 Mar. 2017, http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf.

Vock, J. (2017, April 11). *An Analytics Framework for Non-Experts*. Lecture presented at 2017 UAccess Symposium in University of Arizona, Tucson.

## CHAPTER 3 APPENDIX A

SQL Statements to Create Tables and Define Constraints

(in alphabetical order by table name)

---------------

-- AREAS --

---------------

create table AREAS (

areaID varchar2(4) NOT NULL,

areaName varchar2(30) NOT NULL,

stateID varchar2 (2) NOT NULL,

constraint areas\_pk PRIMARY KEY (areaID),

constraint areas\_unique UNIQUE (areaName));

----------------------

-- AREA\_CSRS --

----------------------

create table AREAS\_CSRS (

areaID varchar2(4) NOT NULL,

CSR\_ID varchar2(4) NOT NULL,

constraint areascsrs\_pk PRIMARY KEY (areaID,CSR\_ID),

constraint csr\_fk FOREIGN KEY (empID) REFERENCES employees(empID));

-----------------

-- BRANDS --

-----------------  
create table BRANDS  
(  
 brandID char(5) constraint brand\_pk PRIMARY KEY,  
 brandName varchar(25) NOT NULL,  
 brandManagerID char(4) NOT NULL,  
 constraint brands\_manager\_fk FOREIGN KEY (brandManagerID) references EMPLOYEES (empID),

constraint brand\_name\_unique UNIQUE (brandName)  
);

---------------------------

-- BRANDS\_FORM --

---------------------------  
create table Brands\_Form  
(  
 brandID char(5) NOT NULL,  
 productID varchar(5) NOT NULL,  
 constraint bf\_brand\_fk FOREIGN KEY (brandID) references BRANDS (brandID),  
 constraint bf\_prodcut\_fk FOREIGN KEY (productID) references PRODUCTS (productID),  
 constraint bf\_pk PRIMARY KEY (brandID, productID)  
);  
  
---------------------------------

-- CLIENT\_SERV\_REPS --

---------------------------------

create table CLIENT\_SERV\_REPS  
(  
 csrID char(4) constraint CSR\_PK PRIMARY KEY (csrID)

seniorityRank varchar(10),  
 csrStatus varchar(10),  
 commission number(2), /\* percentage \*/  
 bonus number(5), /\* Dollars \*/  
 teamID char(4),  
 constraint csr\_emp\_fk FOREIGN KEY (csrID) references Employees (empID),  
 constraint csr\_team\_fk FOREIGN KEY (teamID) references Teams (teamID)  
 );  
  
------------------  
-- CLIENTS --

------------------   
/\* Clients and related tables \*/  
CREATE TABLE CLIENTS (  
 clientID char(5) constraint cl\_clientid\_pk PRIMARY KEY,  
 clientName varchar2(30),  
 isActive char(1),  
 discount number(5,4),  
 commissionRate number(4,2),  
 pcTitle varchar2(40),  
 pcName varchar2(40),  
 pcPhone number(10),  
 pcEmail varchar2(40),  
 primarySiteID char(4),  
 constraint cl\_fk\_primarySiteID FOREIGN KEY (primarySiteID) references SITES (siteID),

constraint client\_unique UNIQUE (clientName)  
);

------------------------------

-- CSR\_ASSIGNMENTS --

------------------------------

create table csr\_assignments

(

csrID char(4) NOT NULL,

Start\_date date,

End\_date date,

areaID varchar2(4) NOT NULL,

clientID char(5) NOT NULL,

constraint csra\_pk PRIMARY KEY (csrID, areaID, clientID),

constraint csra\_areaID\_fk FOREIGN KEY (areaID) references AREAS (areaID),

constraint csra\_csrID\_fk FOREIGN KEY (csrID) references CLIENT\_SERV\_REPS (csrID),

constraint csra\_clientID\_fk FOREIGN KEY (clientID) references CLIENTS (clientID)

);

------------------------------

-- CSR\_SPECIALITY --

------------------------------

create table csr\_speciality

(

csrID char(4) not null constraint csrg\_csrID\_fk references Client\_Serv\_Reps (csrID),

PL\_ID char(5) not null constraint csrg\_PL\_ID\_fk references Product\_Lines (PL\_ID)

);

----------------------------------

-- EMPLOYEE\_HISTORY --

----------------------------------

Create table EMPLOYEE\_HISTORY

(

emplD char(4) constraint eh\_empID\_pk PRIMARY KEY NOT NULL,

Start\_date date,

End\_end date,

title varchar2(35),

salary decimal(8,2),

Constraint eh\_empid\_fk FOREIGN KEY(empID) references EMPLOYEES (empID)

);

--------------------------------------------------

-- EMPLOYEE\_TRAJECTORY\_REPORT --

--------------------------------------------------

CREATE TABLE EMPLOYEE\_TRAJECTORY\_REPORT (

empID char(4),

numRaises integer,

daysSinceLastRaise integer,

currentTitle varchar2(25),

currentSalary number(8,2),

lastIncreasePercentage number(6,2),

varianceAgainstPeersPercentage number(6,2),

constraint et\_empid\_fk FOREIGN KEY (empID) references EMPLOYEES (empID),

constraint et\_empid\_pk PRIMARY KEY (empID)

);

----------------------

-- EMPLOYEES --

----------------------

create table Employees   
(  
 empID char(4) constraint emp\_pk PRIMARY KEY,  
 firstName varchar(25),  
 middleName varchar(25),  
 lastname varchar(25),  
 SSN number(9) NOT NULL,  
 streetName varchar2(50),  
 city varchar2(25),  
 state varchar2(4),  
 zip number(5),  
 hiredate date,  
 termdate date,  
 type varchar2(25),  
 title varchar2(35),  
 paygroup varchar2(25),  
 salary number(8,2),

constraint emp\_ssn\_unique UNIQUE (SSN)  
 );

--------------------

-- FEEDBACK --

--------------------  
CREATE TABLE FEEDBACK (  
 feedbackID char(5) constraint fb\_feedbackid\_pk PRIMARY KEY,  
 comments varchar(1000),  
 fDate date,  
 csrID char(5),--NOT NULL,  
 productID char(5)--NOT NULL

clienID char (5),  
 constraint fb\_fk\_csrID FOREIGN KEY (csrID) references CLIENT\_SERV\_REPS (csrID),

constraint fb\_fk\_productID FOREIGN KEY (productID) references PRODUCTS (productID),  
 constraint fb\_fk\_clientID FOREIGN KEY (clientID) references CLIENTS (clientID)  
);

--------------------------------------

-- GENERAL PROMOTIONS --

--------------------------------------

create table General\_Promotions  
(  
gpID char(6) constraint gp\_pk PRIMARY KEY,  
productID char(5),  
brandmanagerID char(4),  
constraint gp\_emp\_fk FOREIGN KEY (brandmanagerID) references EMPLOYEES (empID)  
constraint gp\_product\_fk FOREIGN KEY (productID) references Products (productID));

---------------------

-- INCIDENTS --

---------------------  
CREATE TABLE INCIDENTS (  
 incidentNum char(5) constraint in\_incidentnum\_pk PRIMARY KEY,  
 dateOccurred date,  
 dateReceived date,  
 dateReviewed date,   
 desciption varchar2(1000),  
 comments varchar2(1000),  
 resolutions varchar2(500),  
 managerID char(5), --NOT NULL,  
 csrID char(5), --NOT NULL,  
 clientID char(5) NOT NULL,  
 constraint in\_managerid\_fk FOREIGN KEY (managerID) references EMPLOYEES (empID),  
 constraint in\_csrid\_fk FOREIGN KEY (csrID) references CLIENT\_SERV\_REPS (csrID),  
 constraint in\_clientid\_fk FOREIGN KEY (clientID) references CLIENTS (clientID)  
);  
  
-----------------

-- OFFICES --

-----------------

create table OFFICES (

officeNum varchar2(3),

officeType varchar2(10) NOT NULL,

officeName varchar2(30) NOT NULL,

regionID varchar2(3),

constraint offices\_pk PRIMARY KEY (officeNum),

constraint region\_fk FOREIGN KEY (regionID) REFERENCES regions(regionID),

constraint officename\_unique UNIQUE (officeName));

---------------------------------

-- ORDER\_PRODUCTS --

---------------------------------

CREATE TABLE ORDER\_PRODUCTS (  
 ORDERID CHAR(5 BYTE),

PRODUCTID CHAR(5 BYTE),   
 ITEM\_DISCOUNT NUMBER(2,2),  
 QUANTITY NUMBER(3,0),   
 LINE\_TOTAL(6,2),

constraint op\_pk PRIMARY KEY (orderID, productID),

constraint OP\_ORDERID\_fk FOREIGN KEY (ORDERID) references ORDERS (orderID),  
 constraint OP\_PRODUCT\_fk FOREIGN KEY (PRODUCTID) references PRODUCTS (productID);

-----------------

-- ORDERS --

-----------------  
DROP table ORDERS;  
  
CREATE TABLE ORDERS   
(  
orderID char(5) constraint ord\_orderid\_pk PRIMARY KEY,  
clientID char(5),  
csrID char(4),  
shipDate date,  
expectedDelivery date,  
shipMethod varchar2(10),  
totalDiscount number (2), --this is percentage  
totalPrice number(6,2),  
siteid char (4),

orderstatus varchar2(20),

orderdate date,

constraint ord\_csrid\_fk FOREIGN KEY (csrID) references CLIENT\_SERV\_REPS (reps),

constraint ord\_clientid\_fk FOREIGN KEY (clientID) references CLIENTS (clientID),

constraint or\_siteid\_fk FOREIGN KEY (siteID) references SITES (siteID)

);

-----------------------------

-- PRODUCT\_LINES --

-----------------------------  
CREATE TABLE Product\_Lines   
(  
pl\_ID char(5),  
plName varchar(30),  
startDate date,  
constraint product\_lines\_pk primary key (pl\_ID),

constraint plName\_unique UNIQUE (plName)  
);

------------------------------------------------------

-- PRODUCT\_LINES\_ARE\_MANAGED --

-----------------------------------------------------  
CREATE TABLE Product\_Lines\_Are\_Managed   
(  
pl\_ID char(5),  
PL\_managerID char(4),  
manage\_start date,

constraint plam\_pk PRIMARY KEY (pl\_ID, pl\_managerID),

constraint plam\_fk FOREIGN KEY (pl\_managerid) references EMPLOYEES (empID),  
constraint pl\_fk FOREIGN KEY (pl\_ID) references PRODUCT\_LINES (pl\_ID),  
);

---------------------------------------

-- PRODUCT\_LINES\_FORM --

---------------------------------------  
CREATE TABLE Product\_Lines\_Form   
(  
pl\_ID char(5),  
productID char(5),  
constraint plf\_pk PRIMARY KEY (pl\_ID, productID),

constraint prod\_fk FOREIGN KEY (productID) references PRODUCTS (productID),  
constraint pl\_fk FOREIGN KEY (pl\_ID) references PRODUCT\_LINES (pl\_ID),  
);

---------------------------------------

-- PRODUCT\_LINES\_NOTES --

---------------------------------------  
Create table PRODUCT\_LINES\_NOTES

(  
 pl\_ID char(5) NOT NULL,  
 note varchar(1000),  
 constraint note\_pl\_fk FOREIGN KEY (plID) references PRODUCT\_LINES (pl\_ID),  
 constraint note\_pk PRIMARY KEY (pl\_ID)  
);

---------------------

-- PRODUCTS --

---------------------  
CREATE TABLE products   
(  
productID char(5),  
prodName VARCHAR2(25),  
isActive char(1),  
baseCost number(6,2),  
prodGroups varchar(25),  
constraint products\_pk primary key (productID),

constraint officename\_unique UNIQUE (officeName),

constraint prodName\_unique UNIQUE (prodName)  
);

----------------------------------  
-- REFRESHER COURSES --

----------------------------------

/\* Refresher courses (typing), training sessions (instantiation), sessions-student (new WE) \*/  
CREATE TABLE REFRESHER\_COURSES (  
 courseID char(5) constraint rc\_courseid\_pk PRIMARY KEY,  
 courseNum number(5),  
 courseName varchar2(50),  
 description varchar2(250),  
 internalCost number(6,2),

);

-------------------

-- REGIONS --

-------------------

create table REGIONS (

regionID varchar2(3),

regionName varchar2(30) NOT NULL,

managerID varchar2(4),

constraint regions\_pk PRIMARY KEY (regionID),

constraint manager\_fk FOREIGN KEY (empID) REFERENCES employees(empID),

constraint regionname\_unique UNIQUE (regionName));

----------------------------------------

-- RESOLUTION\_MEETINGS --

----------------------------------------

CREATE TABLE RESOLUTION\_MEETINGS (  
 rmID char(5) constraint rm\_rmid\_pk PRIMARY KEY,  
 incidentNum char(5) NOT NULL,  
 rmDate date,  
 isCSRPresent char(1),  
 isCSRReassigned char(1),  
 constraint rm\_fk\_incidentNum FOREIGN KEY (incidentNum) references INCIDENTS (incidentNum)  
);

----------------------------------

-- SESSIONS\_STUDENTS --

----------------------------------  
create table Sessions\_Students  
(  
 sessionID varchar(5) NOT NULL,  
 studentID char(4) NOT NULL,

constraint ss\_pk PRIMARY KEY (sessionID, studentID),  
 constraint ss\_session\_fk FOREIGN KEY (sessionID) references TRAINING\_SESSIONS (sessionID),  
 constraint ss\_student\_fk FOREIGN KEY (studentID) references EMPLOYEES (empID)  
);  
  
--------------

-- SITES --

--------------  
create table Sites  
(  
 siteID char(4) constraint site\_pk PRIMARY KEY,  
 clientID char(5) NOT NULL,  
 siteStreet varchar2(25),  
 siteCity varchar2(25),  
 siteState varchar2(25),  
 siteZip char(5),  
 stateID varchar2(2),

constraint site\_client\_fk FOREIGN KEY (clientID) references CLIENTS (clientID),

constraint site\_state\_fk FOREIGN KEY (stateID) references STATES (stateID)  
);

-------------------------------

-- STATE\_PRODUCTS --

-------------------------------

CREATE TABLE STATE\_PRODUCTS

(

stateID varchar2(2),

productID char(5),

constraint st\_pro\_pk PRIMARY KEY (stateID, productID),

constraint sc2\_stateID\_fk FOREIGN KEY (stateID) references STATES (stateID),

constraint sc2\_stateID\_fk FOREIGN KEY (productID) references PRODUCTS (productID)

);

-----------------------------------

-- STATE\_PROMOTIONS --

-----------------------------------

create table State\_Promotions

(

spID char(6) constraint sp\_pk PRIMARY KEY,

stateID char(2),

startDate date,

endDate date,

budget number(10,2),

gpID char2(6),

constraint sp\_stateID\_fk FOREIGN KEY (stateID) references STATES (stateID),

constraint sp\_fk FOREIGN KEY (gpID) references General\_Promotions (gpID)

);

----------------

-- STATES --

----------------

CREATE TABLE STATES(

stateID varchar2(2) constraint states\_pk PRIMARY KEY,

stateName varchar2(15),

priceOffset number(2,2) /\*percentage\*/

regionID varchar2(3),

constraint st\_regionID\_fk FOREIGN KEY (regionID) references (regions),

constraint statename\_unique UNIQUE (stateName)

);

----------------------------

-- STATES\_CLIENTS --

----------------------------

CREATE TABLE STATES\_CLIENTS

(productID char(5),

clientID char(5),

constraint states\_CLIENT\_fk FOREIGN KEY (clientID) references CLIENTS,

constraint states\_PROC\_fk FOREIGN KEY (productID) references PRODUCTS);

---------------

-- TEAMS --

---------------

create table Teams  
 (  
 teamID char(4) CONSTRAINT team\_pk PRIMARY KEY,  
 leadID char(4),  
 teamName varchar(25),  
 siteID char (4),

constraint teamname\_unique UNIQUE (teamName),

constraint team\_fk FOREIGN KEY (leadID) references Employees (empID),

Constraint TEAM\_SITEID\_FK (siteID) references SITES (siteID)  
 );

--------------------------------

-- TRAINING SESSIONS --

--------------------------------  
CREATE TABLE TRAINING\_SESSIONS (  
 sessionID char(5) constraint ts\_sessionID\_pk PRIMARY KEY,  
 startDate date,  
 endDate date,  
 sLocation varchar2(50),  
 productID char(5), NOT NULL,  
 instructorID char(4), NOT NULL,  
 courseID char(5) NOT NULL,  
 constraint ts\_fk\_productID FOREIGN KEY (productID) references PRODUCTS (productID),  
 constraint ts\_fk\_instructorID FOREIGN KEY (instructorID) references EMPLOYEES (empID),  
 constraint ts\_fk\_courseID FOREIGN KEY (courseID) references REFRESHER\_COURSES (courseID));

---------------  
-- VISITS --

---------------  
create table Visits  
(  
 visitID char(5) constraint visit\_pk PRIMARY KEY,  
 csrID char(4) NOT NULL,  
 siteID char(4),  
 vDate date,  
 constraint visit\_csr\_fk FOREIGN KEY (csrID) references CLIENT\_SERV\_REPS (csrID),  
 constraint visit\_site\_fk FOREIGN KEY (siteID) references SITES (siteID)  
);

## CHAPTER 3 APPENDIX B

--Sequence-And-Trigger-Generation-code.sql

-- SEQUENCES --  
  
  
 CREATE SEQUENCE FEEDBACK\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE EMPLOYEES\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE CLIENTS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE RESOLUTION\_MEETINGS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE OFFICES\_sq -- slightly different to work with existing data  
 START WITH 1  
 MAXVALUE 99  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE ORDERS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE STATES\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE TRAINING\_SESSIONS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE INCIDENTS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE AREAS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE TEAMS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE GENERAL\_PROMOTIONS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE REGIONS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE REFRESHER\_COURSES\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE PRODUCT\_LINES\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE PRODUCTS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
  
 CREATE SEQUENCE STATE\_PROMOTIONS\_sq  
 START WITH 100  
 MAXVALUE 9999  
 INCREMENT BY 1  
 ;  
-- TRIGGERS FOR SEQUENCES --  
  
  
 CREATE OR REPLACE TRIGGER FEEDBACK\_sqt BEFORE INSERT ON FEEDBACK  
 FOR EACH ROW  
 BEGIN  
 SELECT FEEDBACK\_sq.NEXTVAL INTO :new.FEEDBACKID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER EMPLOYEES\_sqt BEFORE INSERT ON EMPLOYEES  
 FOR EACH ROW  
 BEGIN  
 SELECT EMPLOYEES\_sq.NEXTVAL INTO :new.EMPID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER CLIENTS\_sqt BEFORE INSERT ON CLIENTS  
 FOR EACH ROW  
 BEGIN  
 SELECT CLIENTS\_sq.NEXTVAL INTO :new.CLIENTID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER RESOLUTION\_MEETINGS\_sqt BEFORE INSERT ON RESOLUTION\_MEETINGS  
 FOR EACH ROW  
 BEGIN  
 SELECT RESOLUTION\_MEETINGS\_sq.NEXTVAL INTO :new.RMID FROM dual;  
 END;  
 /  
   
 -- OFFICES: slightly different to work with existing data  
 CREATE OR REPLACE TRIGGER OFFICES\_sqt BEFORE INSERT ON OFFICES  
 FOR EACH ROW  
 BEGIN  
 SELECT (‘o’||OFFICES\_sq.NEXTVAL) INTO :new.OFFICENUM FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER ORDERS\_sqt BEFORE INSERT ON ORDERS  
 FOR EACH ROW  
 BEGIN  
 SELECT ORDERS\_sq.NEXTVAL INTO :new.ORDERID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER STATES\_sqt BEFORE INSERT ON STATES  
 FOR EACH ROW  
 BEGIN  
 SELECT STATES\_sq.NEXTVAL INTO :new.STATEID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER TRAINING\_SESSIONS\_sqt BEFORE INSERT ON TRAINING\_SESSIONS  
 FOR EACH ROW  
 BEGIN  
 SELECT TRAINING\_SESSIONS\_sq.NEXTVAL INTO :new.SESSIONID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER INCIDENTS\_sqt BEFORE INSERT ON INCIDENTS  
 FOR EACH ROW  
 BEGIN  
 SELECT INCIDENTS\_sq.NEXTVAL INTO :new.INCIDENTNUM FROM dual;  
 END;  
 /  
   
-- This will use the same sequence as EMPLOYEES  
 CREATE OR REPLACE TRIGGER CLIENT\_SERV\_REPS\_sqt BEFORE INSERT ON CLIENT\_SERV\_REPS  
 FOR EACH ROW  
 BEGIN  
 SELECT EMPLOYEES.NEXTVAL INTO :new.CSRID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER AREAS\_sqt BEFORE INSERT ON AREAS  
 FOR EACH ROW  
 BEGIN  
 SELECT AREAS\_sq.NEXTVAL INTO :new.AREAID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER TEAMS\_sqt BEFORE INSERT ON TEAMS  
 FOR EACH ROW  
 BEGIN  
 SELECT TEAMS\_sq.NEXTVAL INTO :new.TEAMID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER GENERAL\_PROMOTIONS\_sqt BEFORE INSERT ON GENERAL\_PROMOTIONS  
 FOR EACH ROW  
 BEGIN  
 SELECT GENERAL\_PROMOTIONS\_sq.NEXTVAL INTO :new.GPID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER REGIONS\_sqt BEFORE INSERT ON REGIONS  
 FOR EACH ROW  
 BEGIN  
 SELECT REGIONS\_sq.NEXTVAL INTO :new.REGIONID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER REFRESHER\_COURSES\_sqt BEFORE INSERT ON REFRESHER\_COURSES  
 FOR EACH ROW  
 BEGIN  
 SELECT REFRESHER\_COURSES\_sq.NEXTVAL INTO :new.COURSEID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER PRODUCT\_LINES\_sqt BEFORE INSERT ON PRODUCT\_LINES  
 FOR EACH ROW  
 BEGIN  
 SELECT PRODUCT\_LINES\_sq.NEXTVAL INTO :new.PL\_ID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER PRODUCTS\_sqt BEFORE INSERT ON PRODUCTS  
 FOR EACH ROW  
 BEGIN  
 SELECT PRODUCTS\_sq.NEXTVAL INTO :new.PRODUCTID FROM dual;  
 END;  
 /  
   
  
 CREATE OR REPLACE TRIGGER STATE\_PROMOTIONS\_sqt BEFORE INSERT ON STATE\_PROMOTIONS  
 FOR EACH ROW  
 BEGIN  
 SELECT STATE\_PROMOTIONS\_sq.NEXTVAL INTO :new.SPID FROM dual;  
 END;  
 /