MIS 531: ENTERPRISE DATA MANAGEMENT

MIS 531 Team "SELECT the best FROM students" Project

Kirsten Fure

Derik Callahan

Julie Davis

Seth Quartey

Kyle Williams

MIS 531 Team “SELECT the best FROM students” Project

Table of Contents

Chapter 1: Requirements Analysis

Requirements Summary

TriannaCorp, a manufacturer of health-care products wants to develop a centralized database of all their data, especially to improve sales forecasting and inventory management.

The company is structured into main-branches (where each main branch has its own department). Each main branch supports 5-25 local branches, which each have their own unique branch number and branch name. One or more main-branches make a region. A region has one manager, who could be just a regional manager or could also perform as a main-branch manager also. There is a head office that oversees everything. Each branch has employees, including one to three managers and client services representatives. A branch could also have functional marketing managers, finance, human resources, IT and other administrative employees. For each employee, they need to store employee address data, contact data and SSN. Some employees are salaried, and some are hourly.

For client service representatives specifically, they need to store seniority rank, commission rate, actual sales, performances and status. They need to track how many representatives they have for each product group; they want at least 10 per product group. Each client service representative can support several product groups, and the system needs to track which date-specific product group training courses, annual courses and new product training sessions the reps have completed. For each course, they need to track the course number, name, description, internal cost, start date, end date, location, and employee who is the instructor. They also track if a customer service rep has been debriefed on a new product launch.

They need to store client information: ID, name, primary address, multiple location addresses, primary contact information (title, name, phone, email) and pricing discounts. They need to track which client-service representatives are supporting the clients. For large locations, they could have multiple client-service representatives (from all the product groups) which form a team of up to ten representatives where one is the lead. For the team, they store a team name and a work area. They track and report how many clients each client-service representative has and limit them to no more than five “areas”, which are based on driving distance. They track each time client service representatives visit a client site, so they can know how often clients are being visited. Also, first-year “trainee” CSRs are encouraged to visit 10 to 20 different clients or client sites during their first year. After they complete this, management upgrades them to “full-time” status CSRs.

Clients report problem incidents to regional management. For each problem incident, they store the manager, date of review, incident number, client, representative, date of incident, date complaint received, client description, discussion comments, actions, current status. They need to be able to track how many problem reports over a period of time and per client. If more than 5 incident reports are filed within the same month or two from the same client within a month, management needs to be alerted.

They store data on four main product groups, known as C3D, which are cosmetic, dental, dermal and diabetic. Each product is also associated with a brand. For each product and each state, they track average cost per unit and the list price. The company strictly does not change prices more than twice in any calendar year.

The cosmetic and dental groups also have product-lines, each with a line number, name, date started, and multiple notes. They track which and how many products are in each line. Each line has one or more product-line managers. Managers can handle more than one line. For each product in the line, they track profitability and volume.

For each product, they track promotions by state and usually store start date, end date, and a promotion budget. Many promotions can be running at the same time for the same product and promotion details vary by state. Promotions are managed by one brand manager.

Client Service representatives take orders, which can contain multiple products. For orders, they need to track discounts applied, shipping date, shipping method, expected delivery date and the order status (ex: if it was cancelled, not fulfilled, etc.). For each item in the order, they need to track the product, quantity, price, and discount. Client Service Representatives also save feedback from clients about the products.

They also want to track history of products, clients, orders and employee records. This may include status, start date, end date, and historical notes.

Assumptions

- Shipping date and shipping method are tracked at an order level, not for individual items/products.
- Employee address is their home address.
- A branch must belong to a region and cannot belong to more than one region.
- A combination of branches cannot make up more than one region.
- A personnel manager can manage at most one region.
- A region has at least one, and at most one, personnel manager.
- Only client service reps receive the quarterly bonuses based on actual sales and performances.
- A team may have only one, but must have at least one, client service representative.
- A combination of client service reps may form multiple teams (for different clients).

- Only one discount can be applied to a single order or product, but a discount may be applied to both an order and a product within the same order.
- A problem incident is related to a single client service representative. For example, if two client service reps were involved in the same problem, separate problem incidents would be created for each representative.
- A single combination of personnel managers, functional personnel, and client service reps can make up only one unique branch.
- Client sites are considered independent clients for purposes of counting how many “clients” a representative is assigned to at any given time.
- A single brand manager may or may not manage many product lines.
- A product is associated with a single product line and cannot be in more than one line at a time.
- A product is associated with a single product group and cannot be in more than one group at a time.
- Every order must be associated with one and only one client service rep.
- Any employee type can take courses.

User Roles - Data Requirements

HR/Payroll

- Reporting of employee addresses and contact information in order to mail checks.
- Employee data, address, contact information, SSN, hourly or salary.
- Reporting on employee seniority level, commission rate and actual sales for commissions payment to CSRs and commission rate assignment (based on seniority level).
- Actual sales and performance for client-service representatives for bonus assignments.
- Reporting of branch employees and their respective positions. This can help HR to know what positions are available and how many staff support each branch.
- Reporting on how many employees within each position, each branch and/or region.

Regional Management

- Problem incident report information - finding any potential problems/trends with certain products or certain clients. For example, more than five problem reports overall in a month is a problem, and if the same client has two or more in a month, management needs to take action.

- Tracking status and action items on problem incidents.
- Client information, such as a list of all their locations, addresses and current team assignments.
- Client-representative information such as how many areas each CSR is representing. (Note: Each representative should have five or less.)
- Reports of how many lines a product line manager is managing and how many products are within each line.

Client Service Representatives

- Client info such as pricing discount information.
- Problem incidents and status.
- Create orders and track order details such as shipping status and number of days from order placement to order delivery.
- Product information.
- Access to product pricing information for order placement.
- Enter Client Feedback on Products and run reporting for management.

Marketing

- Promotion data by state or current promotions for each product.
- Ordering and sales data by state in order to determine which states are profitable.
- Effectiveness of a promotion on sales.

Product Line Managers

- Average cost per unit for each product.
- Product pricing information.
- Reports on how many products are in the product line. (Note: A line should have no more than 20 products.)
- Reporting on the highest product profitability and highest volume by querying order information related to certain product lines within a certain time period.

- Customer feedback by product and finding trends or helpful information from end customers.

Personnel Management

- Client team assignment, showing which CSRs are on which team representing which client and location.
- Reporting on how many and which clients a CSR is representing. (Note: A CSR should represent between 20 and 40 clients.)
- Reporting on how many product groups each CSR represents.
- Reporting on how many CSRs are representing each group. (Note: A group should be represented by at least 10 CSRs.)
- Access to completed training course information for employees.
- Produce list of upcoming training courses, dates, descriptions, and locations.
- Reporting on first-year “trainee” CSRs and how many different unique client, or client sites, they are visiting. (Note: A CSR should visit 10 to 20 in the first year.) Then management can upgrade them to “full-time” status CSRs.
- Report how many times each client has had a visit from a CSR per month.
- Report how often CSR visits result in Orders.

Finance

- Auditing reports on order discounts and client pricing discounts to ensure valid discounts are being applied by CSRs.
- Auditing reports on product pricing change frequency. (Note: The price of a product should change no more than two times in a year.)

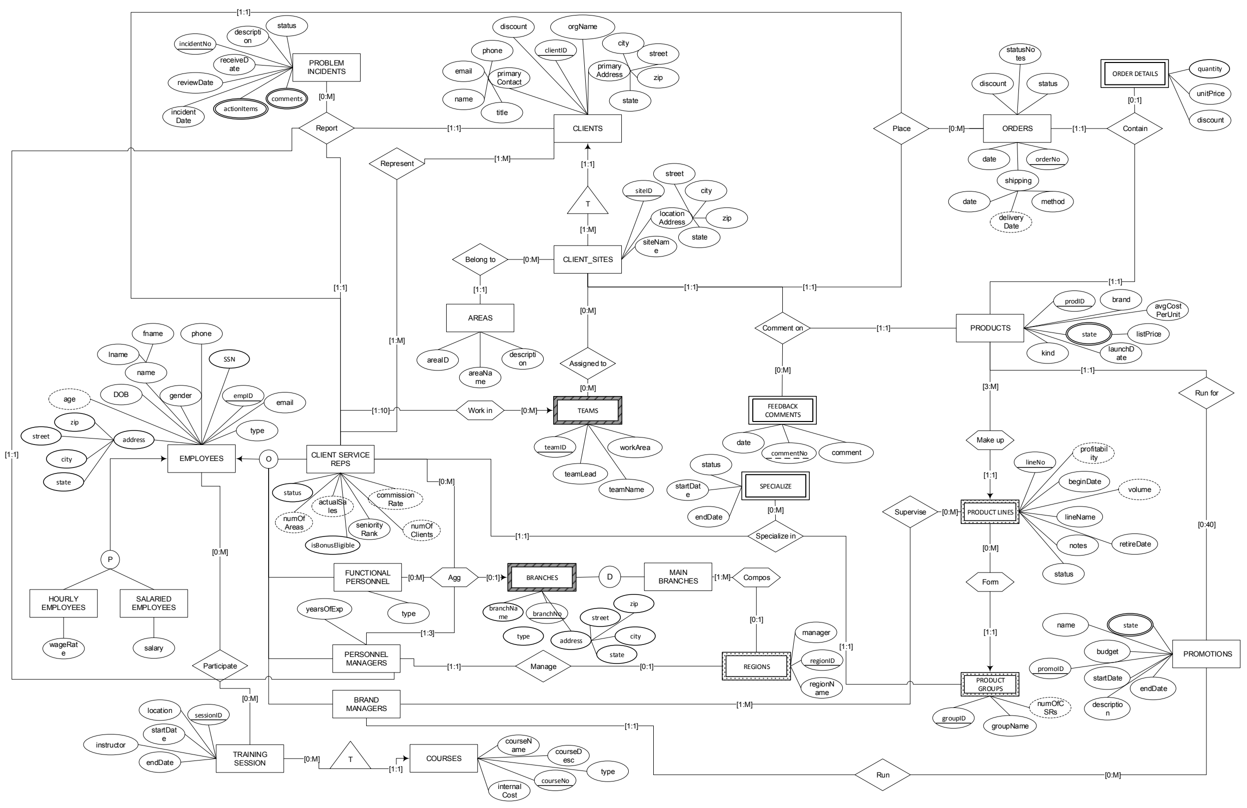
Advantages of Database vs. File System for these Users

- TriannaCorp employees from different branches and regions will have access to the same consistent and current information, including products, promotions, and orders.
- The database offers transaction technology so that the most accurate and current information is available to all users. For example, a user’s input will never be overwritten by someone else’s.
- The database offers customizable data security so that certain data can be available to certain users through their sign-on information.

- There will be less data redundancy saving employees time. They will only need to enter information one time in one place.
- Accurate and up-to-date ordering information will help with inventory tracking and financial forecasting.
- Analysis and business decisions can be performed quickly due to the data design. Management will be able to run a multitude of reports and analyze the business across product lines or for certain time periods or for certain states, etc. They will be able to analyze the effectiveness of promotions and analyze profitability.
- They can track problem incidents and provide timely customer service to their clients by noticing potential problem areas and rises in problem incidents among certain products or certain clients.

Chapter 2: Conceptual Schema

ER Diagram



Data Dictionary (Conceptual)

Schema Construct	Construct Description	Other Information
AREAS	Aggregate class to client sites Information	
area_id		Primary Key/Identifying Attribute Format char(5)
area_name		
area_description		
ASSIGNED TO	Relationship that models Teams assigned to client sites	

BRANCHES	Weak entity class to Employee functional personnel	
branchno		Primary Key/Identifying Attribute Format char(5)
branchName		
type		CHECK (type IN('local', 'main')
street		
city		
state		
zip		
main_branchno		Foreign key main_branchno references MAIN_BRANCHES branchno
BRAND MANAGER	Subclass to Employee Information	
empid		Foreign Key in EMPLOYEES
lineNo		Foreign Key in PRODUCT LINE
CLIENT SITES	Entity Class to client sites information	
siteID		Primary Key/Identifying Attribute Format char(8)
site_name		
street		
city		
state		
zip		
clientID		Foreign Key clientID references CLIENTS
area_ID		Foreign key area_ID references AREAS
CLIENTS	Entity Class to clients information	
clientID		Primary Key/Identifying Attribute Format char(10)
orgname		
contact_name		
contact_email		
contact_phone		

contact_title		
city		
street		
· City		
state		
zip		
corp_pricing_discount		
CLIENT SERVICE REPRESENTATIVE	Subclass to Employee Information	
empid		Foreign key empID references EMPLOYEES empID
csr_status		
comm_rate		
seniority_rank		(seniority_rank('Senior','Junior','Associate')
actual_sales		
numOfClients		
numofAreas		
COMMENT ON	Relationship that models feedback comments comment on Products by clients	
COMPOS	Relationship that models which branches belongs to regions	
CONTAIN	Relationship that models products contained in order details	
COURSES	Entity Class to courses information	
courseNo		Primary Key/Identifying Attribute Format char(6)

courseName		
courseDesc		
internal_cost		Cost in (US $)
course_type		
EMPLOYEES	Entity Class to employee information	
empID		Primary Key/Identifying Attribute Format char(10)
lname		
fname		
DOB		
phone		
SSN	Check and unique constraints(SSN should be unique)	
email		
gender		
street		
city		
state		
zip		
position		
branchNo		Foreign key branchNo references BRANCHES branchNo
FEEDBACK	Weak Entity Class to clients information	
siteid		foreign key siteID references CLIENT_SITES siteID
prodID		foreign key prodID references PRODUCTS prodID
commentNo		Primary Key/Identifying Attribute Format char(6)
comments		
feedback_date		
FORM	Relationship that models Product Line which forms	

	Product Group	
FUNCTIONAL MANAGERS	Subclass to Employee Information	
empID		Foreign key empdID references EMPLOYEES empID
mgr_type		
GROUP	Relationship that models client sites group by areas	
HOURLY EMPLOYEES	Subclass to Employee Information	
empID		Foreign key empID references EMPLOYEES empID
wage_rate		Rate in (US $)
MANAGE	Relationship that models which personnel managers mange regions	
MAKE UP	Relationship that models Product Line make up product group[	
MAIN BRANCHES		
branchno		Primary Key, Foreign key branchno references BRANCHES branchno
regionID		Foreign key regionID references REGIONS regionID, NOT NULL
ORDER DETAILS	Weak entity class to orders information	
orderID		Foreign key orderID references ORDERS orderID

prodID		Foreign key prodID references PRODUCTS prodID
quantity		
unitPrice		Price in (US $)
prod_discount		
ORDERS	Entity Class to orders information	
orderNo		Primary Key/Identifying Attribute Format char(10)
order_date		Foreign Key in Order Details discount in decimal (ie .25)
order_discount		
shipping_method		Ground', 'Freight', 'Priority', 'Overnight'
shipping_date		
delivery_date		
order_status		in process', 'in transit', 'cancelled', 'fulfilled', 'not fulfilled', 'backordered'
ord_status_notes		
PARTICIPATES	Relationship that models the employees participating in Training	
PERSONNEL MANAGERS	Subclass to Employee Information	
empID		Foreign key empdID references EMPLOYEES empID
yrs_exp		
PLACE	Relationship that models Clients who place orders	
PROBLEM INCIDENTS	Weak Entity Class to clients information	
incidentNo	Partial Identifying	Primary Key/Identifying Attribute Format

	Client Service Rep	char(10)
incidentDate		
reviewDate		Format mm/dd/yyyy
receivedDate		Format mm/dd/yyyy
incident_description		Format mm/dd/yyyy
incident_status		Open', 'Work in Progress', 'Awaiting Client', 'Resolved', 'Closed'
PRODUCT GROUPS	Composite entity to product information	
groupID		Primary Key/Identifying Attribute Format char(5)
groupName		Unique Name
numOfCSRs		
PRODUCT LINE	Composite entity to product information	Number
lineNo		Primary Key/Identifying Attribute Format char(6)
lineName		LineName is unique
line_beginDate		Format mm/dd/yyyy
line_notes		note: line_notes stores information about the nature of the line
line_status		
line_retireDate		
groupID		Foreign key groupID references PRODUCT_GROUPS groupID
PRODUCTS	Entity Class to products information	
prodID		Primary Key/Identifying Attribute Format char(10)
prodname		Unique Name
brand		
avg_costperunit		avg_costperunit > 0
launch_date		
prod_status		
retire_date		

lineNo		Foreign key references PRODUCT_LINES lineNo
PROMOTIONS	Entity to Promotions information	
promoID		Primary Key/Identifying Attribute Format char(10)
promoname		
promodescription		Format mm/dd/yyyy
prodID		Foreign key prodID references PRODUCTS prodID
mgrID		Foreign key mgrID references PRODLINE_MANAGERS empID
REGIONS	Composite entity to personnel Managers	
regionID		Primary Key/Identifying Attribute Format char(5)
regionName		
mgr_id		Foreign key mgr_id references PERSONNEL_MANAGERS empID, assuming that each region has only 1 manager
REPORT	Relationship that models clients report problem incidents	
REPRESENTS	Relationship that models client service rep represent clients	
RUN	Relationship that models which Brand Manager runs promotions	
RUN FOR	Relationship that models products run for promotion	

SALARIED EMPLOYEES	Subclass to Employee Information	
empID		Foreign key empID references EMPLOYEES empID
salary		Rate in (US $)
SPECIALIZE IN	Relationship that models product group that clients specialize in	
SUPERVISE	Relationship that models which Product Line is supervise by Brand Manager	
TEAMS	Aggregate class to client sites Information	
teamid		Primary Key/Identifying Attribute Format char(7)
teamName		
local_workarea		
team_leadID		Foreign key team_LeadID references EMPLOYEES empID
TRAINING SESSION	Entity Class to training session information	
sessionID		Primary Key/Identifying Attribute Format char(10)
location		
startDate		
endDate		
instructor_id		Foreign key instructor_id references EMPLOYEES empid, NOT NULL
courseNo		Foreign key courseNo references COURSES courseNo, NOT NULL
WORK IN	Relationship that models client service reps work	

	in teams

Schema

Field Name	Description	Key
AREAS		
area_id	Area ID	Primary key
area_name	Area Name	
area_description	Area Description	
ASSIGN_TEAMS_TO_CLIENTS		
siteID	Site ID	Foreign key siteID references CLIENT_SITES siteID
teamid	Team ID	Foreign key teamID references TEAMS teamID
begin_date	Begin Date	
end_date	End Date	
status	Status	
BRANCHES		
branchno	Branch Number	Primary Key
branchName	Branch Name	
type	Type	
street	Street	
city	City	
state	State	
zip	Zip Code	
main_branchno	Main Branch	Foreign key main_branchno references MAIN_BRANCHES branchno
CLIENT_SITES		
siteID	Site ID	Primary Key
site_name	Site Name	
street	Street	
city	City	
state	State	
zip	Zip Code	
clientID	Client ID	Foreign Key clientID references CLIENTS
area_ID	Area ID	Foreign key area_ID references AREAS
CLIENTS		
clientID	Client ID	Primary Key

orgname	Organization Name	
contact_name	Contact Name	
contact_email	Contact Email	
contact_phone	Contact Phone	
contact_title	Contact Title	
city	City	
street	Street	
state	State	
zip	Zip Code	
corp_pricing_discount	Corprate Pricing Discount	
COURSES		
courseNo	courseNo	Primary Key
courseName	courseName	
courseDesc	courseDesc	
internal_cost	internal_cost	
course_type	course_type	
CSR_EMP		
empid	employee ID	Foreign key empID references EMPLOYEES empID
csr_status	Client Service Representative Status	
comm_rate	Commision Rate	
seniority_rank	Seniority Rank	
actual_sales	Actual Sales	
numOfClients	Number of Clients	
numofAreas	Number of Areas	
EMPLOYEES		
empID	employee ID	Primary Key
lname	Last Name	
fname	First Name	
DOB	Date of Birth	
phone	Phone	
SSN	Social Security Number	
email	Emial	
gender	Gender	
street	Street	
city	City	
state	State	
zip	Zip Code	

position	Position	
branchNo	Branch Number	Foreign key branchNo references BRANCHES branchNo
FEEDBACK		
siteid	Site ID	foreign key siteID references CLIENT_SITES siteID
prodID	Product ID	foreign key prodID references PRODUCTS prodID
commentNo	Comment No	Primary key
comments	Comments	
feedback_date	Feedback Date	
FORM_TEAMS		
teamid	Team ID	Foreign Key teamid references TEAMS teamid
CSR_empID	Client Service Representative Employee ID	Foreign Key CSR_empID references CSR_EMP empID
FUNCTIONAL_MANAGERS		
empID	Employee ID	Foreign key empdID references EMPLOYEES empID)
mgr_type	Manager Type	
HOURLY_EMPLOYEES		
empID	Employee ID	Foreign key empID references EMPLOYEES empID
wage_rate	Wage Rate	
MAIN_BRANCHES		
branchno	Branch Number	Primary Key, Foreign key branchno references BRANCHES branchno
regionID	Region ID	Foreign key regionID references REGIONS regionID
ORDER_DETAILS		
orderID	Order ID	Foreign key orderID references ORDERS orderID
prodID	Product ID	Foreign key prodID references PRODUCTS prodID
quantity	Quantity	
unitPrice	Unit Price	
prod_discount	Product Discount	
ORDERS		
orderNo	Order Number	Primary Key
order_date	Order Date	
order_discount	Order Discount	

shipping_method	Shipping Method	
shipping_date	Shipping Date	
delivery_date	Delivery Date	
order_status	Order Status	
ord_status_notes	Order status Notes	
PERSONNEL_MANAGERS		
empID	Employee ID	Foreign key empdID references EMPLOYEES empID
yrs_exp	Years of Experience	
PLACE_ORDERS		
orderNo	Order Number	Foreign key orderID references ORDERS orderID
siteID	Site ID	Foreign key clientID referenced CLIENT_SITES clientID
CSR_empID	Client Service Representative Employee ID	Foreign key CSR_empdID references CSR_EMP empID
PROBLEM_ACTION_ITEMS		
incidentNo	Incident No	Foreign key incidentNo references PROBLEM_INCIDENTS incidentNo
action_item	Action Item	
due_date	Due Date	
action_status	Action Status	
PROBLEM_COMMENTS		
incidentNo	Incident Number	Foreign key incidentNo references PROBLEM_INCIDENTS incidentNo
comment	Comment	
PROBLEM_INCIDENTS		
incidentNo	Incident Number	Primary Key
incidentDate	Incident Date	
reviewDate	Review Date	
receivedDate	Received Date	
incident_description	Incident Description	
incident_status	Incident Status	
PRODLINE_MANAGERS		
empID	Employee ID	Foreign key empdID references EMPLOYEES empID
prod_lineNo	Product Line Numnber	Foreign key prod_lineNo references PRODUCT_LINES lineNo
PRODUCT_GROUPS		
groupID	Gorup ID	Primary Key

groupName	Group Name	
numOfCSRs	Number of Client Service Representative Employees	
PRODUCT_LINES		
lineNo	Line Number	Primary Key
lineName	Line Name	
line_beginDate	Line Begin Date	
line_notes	Line Notes	
line_status	Line Status	
line_retireDate	Line Retire Date	
groupID	Gorup ID	Foreign key groupID references PRODUCT_GROUPS groupID
PRODUCT_PRICING		
prodID	Product ID	Primary key, Foreign Key prodID references PRODUCTS prodID
state	State	Primary key
listPrice	List Price	
PRODUCTS		
prodID	Product ID	Primary Key
prodname	Product Name	
brand	Brand	
avg_costperunit	Average cost per Unit	
launch_date	Launch Date	
prod_status	Product Status	
retire_date	Retire Date	
lineNo	Line Number	Foreign key references PRODUCT_LINES lineNo
PROMOSTATE		
promoID	Promotion ID	Primary key, Foreign key promoID references PROMOTIONS promoID
state	State	Primary key
budget	Budget	
promo_start_date	Promotion Start Date	
promo_end_date	Promotion End Date	
PROMOTIONS		
promoID	Promotion ID	Primary Key
promoname	Promotion Name	
promodescription	promotion Description	
prodID	Product ID	Foreign key prodID references PRODUCTS prodID
mgrID	Manager ID	Foreign key mgrID references PRODLINE_MANAGERS empID

REGIONS		
regionID	Region ID	Primary Key
regionName	Region Name	
mgr_id	Manager ID	Foreign key mgr_id references PERSONNEL_MANAGERS empID
REPORT_PROBLEMS		
clientID	Client ID	Foreign key clientID references CLIENTS clientID
CSR_empID	Client Service Representative Employee ID	Foreign key CSR_empID references CSR_EMP empID
incidentNo	Incident Number	Foreign key incidentNo References PROBLEM_INCIDENTS incidentNo
SALARY_EMPLOYEES		
empID	Employee ID	Foreign key empID references EMPLOYEES empID
salary	Salary	
SPECIALIZE		
groupID	Gorup ID	Primary key, Foreign key groupID references PRODUCT_GROUPS groupID
CSR_empid	Client Service Representative Employee ID	Primary key, foreign key CSR_empID references CSR_EMP empid
sp_start_date	Specialized Start Date	
sp_end_date	Specialized End Date	
sp_status	Specialized Status	
SUPERVISE_LINES		
lineNo	Line Number	Primary key, Foreign key lineNo references PRODUCT_LINES lineNo
mgr_id	Manager ID	Primary key, Foreign key mgr_id references PRODLINE_MANAGERS empid
sup_start_date	Supervise Start Date	
sup_end_date	Supervise End Date	
supervise_status	Supervise Status	
TAKE_TRAINING		
empid	Employee ID	Foreign key empid references EMPLOYEES empId
sessionID	Session ID	Foreign key prodID references PRODUCTS prodID
TEAMS		
teamid	Team ID	Primary key
teamName	Team Name	
local_workarea	Local Workarea	

team_leadID	Team Lead ID	Foreign key team_LeadID references EMPLOYEES empID
TRAINING_SESSIONS		
sessionID	Session ID	Primary Key
location	Location	
startDate	Start Date	
endDate	End Date	
instructor_id	Instructor ID	Foreign key instructor_id references EMPLOYEES empid
courseNo	Course Number	Foreign key courseNo references COURSES courseNo
VISITLOG		
visit_date	Visit Date	Primary Key
CSR_empID	Client Service Representative Employee ID	Foreign key CSR_empID references CSR_EMP empID, Primary key
siteID	Site ID	Foreign key siteID references CLIENT_SITES siteID, Primary key
hours_worked	Hours Worked	

Chapter 3: Relational Schema

Normalized Relations

AREAS(area_id, area_name, area_description)

area_name *is UNIQUE and NOT NULL*

ASSIGN_TEAMS_TO_CLIENTS(siteID, teamID, begin_date, end_date, status)

Foreign key siteID references CLIENT_SITES siteID

Foreign key teamID references TEAMS teamID

Note: status can be 'active' or 'inactive'

BRANCHES(branchno, branchName, type, street, city, state, zip, main_branchno)

Foreign key main_branchno references MAIN_BRANCHES branchno

note :type is for local/main

note: branchName UNIQUE and NOT NULL

subclass:

MAIN_BRANCHES(branchno, regionID)

Foreign key branchno references BRANCHES branchno

Foreign key regionID references REGIONS regionID

<constrain main branches to 25 local branches>

note: regionID NOT NULL

CLIENTS(clientID, orgname, contact_name, contact_email, contact_phone, contact_title, city, street, state, zip, corp_pricing_discount)

note:orgName should be NOT NULL

CLIENT_SITES(site_ID, site_name, street, city, state, zip, clientID, area_ID)

Foreign Key clientID references CLIENTS clientID

Foreign key area_ID references AREAS area_ID

note: site_name NOT NULL

note: clientID and area_ID NOT NULL

COURSES(<u>courseNo</u>, courseName, courseDesc, internal_cost, course_type)

note: courseName UNIQUE and NOT NULL

EMPLOYEES(<u>empID</u>, lname, fname, DOB, phone, SSN, email, gender, street, city, state, zip, position, branchNo)

Foreign key branchNo references BRANCHES branchNo

note SSN is UNIQUE

subclasses:

CSR_EMP(<u>empID</u>, csr_status, seniority_rank, actual_sales, numOfClients, numofAreas, comm_rate)

Foreign key empdID references EMPLOYEES empID

actual_sales is derived

Comm_rate based on seniority

numOfClients is a derived attribute based on relationship with ASSIGN_TEAMS_TO_CLIENTS and FORM_TEAMS

numOfAreas is a derived attribute based on relationship with ASSIGN_TEAMS_TO_CLIENTS, FORM_TEAMS and CLIENT_SITES

FUNCTIONAL_MANAGERS(<u>empID</u>, mgr_type)

Foreign key empdID references EMPLOYEES empID)

PERSONNEL_MANAGERS(<u>empID</u>, yrs_exp)

Foreign key empdID references EMPLOYEES empID

PRODLINE_MANAGERS(<u>empID</u>, prod_lineNo)

Foreign key empdID references EMPLOYEES empID

Foreign key prod_linNo references PRODUCT_LINES lineNoy

HOURLY_EMPLOYEES(<u>empID</u>, wage_rate)

Foreign key empID references EMPLOYEES empID

SALARY_EMPLOYEES(<u>empID</u>, salary)

Foreign key empID references EMPLOYEES empID

FEEDBACK(commentNo. siteID, prodID, comment, feedback_date)

Foreign key siteID references CLIENT_SITES siteID

Foreign key prodID references PRODUCTS prodID

FORM_TEAMS(teamID, CSR_empID)

Foreign Key teamID references TEAMS teamID

Foreign Key CSR_empID references CSR_EMP empID)

check constraint Team has 1 to 10 employees

ORDERS(orderNo, order_date, order_discount, shipping_method, shipping_date, delivery_date, order_status, ord_status_notes)

Assumption: all products in an order use the same shipping method and are shipped as one package

note: order_status constraint check values (in process, in transit, cancelled, fulfilled, not fulfilled, backordered)

Check constraint shipping_method ground, freight, priority, overnight

ORDER_DETAILS(orderno prodID, quantity, unitPrice, prod_discount)

Foreign key orderno references ORDERS orderno

Foreign key prodID references PRODUCTS prodID

PLACE_ORDERS(orderNo, siteID, CSR_empID)

Foreign key CSR_empdID references CSR_EMP empID

Foreign key orderNo references ORDERS orderNo

Foreign key siteID referenced CLIENT_SITES siteID

PROBLEM_INCIDENTS(incidentNo, incidentDate, reviewDate, receivedDate, incident_description, incident_status)

PROBLEM_COMMENTS(incidentNo, comment)

Foreign key incidentNo references PROBLEM_INCIDENTS incidentNo

Assumption, every comment, incidentNo combination is unique

PROBLEM_ACTION_ITEMS(incidentNo, action_item, due_date, action_status)

Foreign key incidentNo references PROBLEM_INCIDENTS incidentNo

PRODUCTS(prodID, prodname, brand, avg_costperunit, launch_date, prod_status, retire_date, lineNo)

Foreign key lineNo references PRODUCT_LINES lineNo

note: prodname is UNIQUE and NOT NULL

PRODUCT_GROUPS(groupID, groupName, numOfCSRs)

note: groupName is UNIQUE and NOT NULL

note: numOfCSRs is a derived attribute based on relationship with SPECIALIZE

PRODUCT_LINES(lineNo, lineName, line_beginDate, line_notes, line_status, line_retireDate, groupID)

LineName UNIQUE and NOT NULL

Foreign key groupID references PRODUCT_GROUPS

note: line_notes stores information about the nature of the line

note: line_profitability is a derived attribute

note: line_volume is a derived attribute

PRODUCT_PRICING(prodID, state, listPrice)

Foreign Key prodID references PRODUCTS prodID

PROMOTIONS(promoID, promoname, promodescription, prodID, mgrID)

Foreign key prodID references PRODUCTS prodID

Foreign key mgrID references PRODLINE_MANAGERS empID

PROMOSTATE(promoID, state, budget, promo_start_date, promo_end_date)

Foreign key promoID references PROMOTIONS promoID

REGIONS(regionID, regionName, mgr_id)

Foreign key mgr_id references PERSONNEL_MANAGERS empID

regionName is UNIQUE and NOT NULL

note: assuming that each region has only 1 manager

REPORT_PROBLEMS(clientID, CSR_empID, incidentNo)

Foreign key clientID references CLIENTS clientID

Foreign key CSR_empID references CSR_EMP empID

Foreign key incidentNo References PROBLEM_INCIDENTS incidentNo

SPECIALIZE(groupID, CSR_empID, sp_start_date, sp_end_date, sp_status)

Foreign key groupID references PRODUCT_GROUPS groupID

Foreign key empID references CSR_EMP empID

SUPERVISE_LINES(lineNo, mgr_id, sup_start_date, sup_end_date, supervise_status)

Foreign key lineNo references PRODUCT_LINES lineNo

Foreign key mgr_id references PRODLINE_MANAGERS empid

TAKE_TRAINING(empid, sessionID)

Foreign key empid references EMPLOYEES empId

Foreign key sessionID references TRAINING_SESSIONS sessionID

TEAMS(teamID, local_workarea, team_leadID)

Foreign key team_leadID references EMPLOYEES empID

TRAINING_SESSIONS(sessionID, location, startDate, endDate, instructor_id, courseNo)

Foreign key courseNo references COURSES courseNo

Foreign key instructor_id references EMPLOYEES empid

note: courseNo and instructor_id is NOT NULL

VISITLOG(visit_date, CSR_empID, siteID, hours_worked)

Foreign key CSR_empID references CSR_EMP empID

Foreign key siteID references CLIENT_SITES siteID

Explanations for Relational Schema Subclass/Superclass Choices

For Employee subclasses, we chose to implement Option “A” where we have the superclass EMPLOYEE table and we have a separate table for each subclass CSR_EMP, FUNCTIONAL_MANAGERS, PERSONNEL_MANAGERS, PRODLINE_MANAGERS, HOURLY_EMPLOYEES and SALARY_EMPLOYEES. We chose this approach because we have a “O” subclass for employee position and a “P” subclass for employee pay type (hourly or salary). By using this approach, we avoid having redundant data and unused (null) attributes. We only create subclass tables for employee positions that have unique attributes and relationships.

For Branches, we created a subclass for the main branch because it has a relationship to regions that the local branches do not have. We chose option “A” for this subclass. It has a “P” type relationship; the region is either local or main, so this makes for a clean and organized solution.

Data Dictionary (Relational)

Schema Construct	Data Type	Constraint
AREAS	Relation representing the entity class AREAS	
area_id	char (5)	Primary key
area_name	varchar2 (25)	UNIQUE and NOT NULL
area_description	varchar2 (25)	
FD: area_id --> area_id, area_name, area_description		
FD: area_name --> area_id, area_name, area_description		
ASSIGN_TEAMS_TO_CLIENTS	Relation representing the entity class ASSIGN_TEAMS_TO_CLIENTS	
siteID	char (8)	Foreign key siteID references CLIENT_SITES siteID
teamid	char (7)	Foreign key teamID references TEAMS teamID
begin_date	date	
end_date	date	
status	varchar2 (25)	CHECK (status IN ('active', 'inactive'))
FD: siteID, teamID --> siteID, teamID, begin_date, end_date, status		
BRANCHES	Relation representing the entity class BRANCHES	
branchno	char (5)	Primary Key

branchName	varchar2 (25)	UNIQUE and NOT NULL
type	varchar2 (5)	CHECK (type IN('local', ‘main’)
street	varchar2 (25)	
city	varchar2 (25)	
state	char (2)	
zip	char (5)	
main_branchno	char (5)	Foreign key main_branchno references MAIN_BRANCHES branchno
FD: branchNo --> branchNo, branchName, type, street, city, state, zip, main_branchno		
FD: branchName --> branchNo, branchName, type, street, city, state, zip, main_branchno		
CLIENT_SITES	Relation representing the entity class CLIENT_SITES	
siteID	char (8)	Primary Key
site_name	varchar2 (25)	NOT NULL
street	varchar2 (25)	
city	varchar2 (25)	
state	char (2)	
zip	char (5)	
clientID	char (10)	Foreign Key clientID references CLIENTS clientID,NOT NULL
area_ID	char (5)	Foreign key area_ID references AREAS area_ID,NOT NULL
FD:		
CLIENTS	Relation representing the entity class CLIENTS	
clientID	char (10)	Primary Key
orgname	varchar2 (25)	NOT NULL
contact_name	varchar2 (25)	
contact_email	varchar2 (25)	
contact_phone	char (10)	
contact_title	varchar2 (25)	
city	varchar2 (25)	
street	varchar2 (25)	
state	char (2)	
zip	char (5)	
corp_pricing_discount	number (3,2)	CHECK (corp_pricing_discount BETWEEN 0 AND 1)
FD: clientID --> clientID, orgname, contact_name, contact_email, contact_phone, contact_title, city, street,		

state, zip, corp_pricing_discount		
COURSES	Relation representing the entity class COURSES	
courseNo	char (6)	Primary Key
courseName	varchar2 (25)	UNIQUE and NOT NULL
courseDesc	varchar2 (40)	
internal_cost	number (7,2)	
course_type	varchar2 (25)	
FD: courseNo --> courseNo, courseName, courseDesc, internal_cost, course_type		
FD: courseName --> courseNo, courseName, courseDesc, internal_cost, course_type		
CSR_EMP	Relation representing the entity class CSR_EMP	
empid	char(10)	Foreign key empID references EMPLOYEES empID
csr_status	varchar2 (25)	
comm_rate	number (3,2)	
seniority_rank	varchar2 (10)	CHECK(seniority_rank('Senior','Junior','Associate'))
actual_sales	number(10,2)	
numOfClients	number (10)	
numofAreas	number (10)	
FD: empid --> empid, csr_status, comm_rate, seniority_rank, actual_sales, numOfClients, numofAreas		
EMPLOYEES	Relation representing the entity class EMPLOYEES	
empID	char (10)	Primary Key
lname	varchar2 (25)	NOT NULL
fname	varchar2 (25)	NOT NULL
DOB	date	NOT NULL
phone	char (10)	
SSN	char (10)	UNIQUE
email	varchar2 (30)	
gender	varchar2 (20)	
street	varchar2 (25)	
city	varchar2 (20)	
state	char (2)	
zip	char (5)	
position	varchar2 (25)	
branchNo	char (5)	Foreign key branchNo references BRANCHES branchNo

FD: empID --> empID, lname, fname, DOB, phone, SSN, email, gender, street, city, state, zip, position, branchNo		
FEEDBACK	Relation representing the entity class FEEDBACK	
siteid	char (8)	foreign key siteID references CLIENT_SITES siteID
prodID	char (10)	foreign key prodID references PRODUCTS prodID
commentNo	char (6)	Primary key
comments	varchar2 (50)	
feedback_date	date	
FD: commentNo --> siteid, prodID, commentNo, comments, feedback_date		
FORM_TEAMS	Relation representing the entity class FORM_TEAMS	
teamid	char (7)	Foreign Key teamid references TEAMS teamid
CSR_empID	char (10)	Foreign Key CSR_empID references CSR_EMP empID
FD: teamid, CSR_empID --> teamid, CSR_empID		
FUNCTIONAL_MANAGERS	Relation representing the entity class FUNCTIONAL_MANAGERS	
empID	char (10)	Foreign key empdID references EMPLOYEES empID)
mgr_type	varchar2 (30)	
FD: empID --> empID, mgr_type		
HOURLY_EMPLOYEES	Relation representing the entity class HOURLY_EMPLOYEES	
empID	char (10)	Foreign key empID references EMPLOYEES empID
wage_rate	number (5,2)	
FD: empID --> empID, wage_rate		
MAIN_BRANCHES	Relation representing the subclass MAIN_BRANCHES	
branchno	char (5)	Primary Key, Foreign key branchno references BRANCHES branchno
regionID	char (5)	Foreign key regionID references REGIONS regionID, NOT NULL

FD: branchNo --> branchNo, regionID		
ORDER_DETAILS	Relation representing the entity class ORDER_DETAILS	
orderID	char (10)	Foreign key orderID references ORDERS orderID
prodID	char (10)	Foreign key prodID references PRODUCTS prodID
quantity	number (5)	
unitPrice	number(12,2)	
prod_discount	number (7,2)	
FD: orderID, prodID --> orderID, prodID, quantity, unitPrice, prod_discount		
ORDERS	Relation representing the entity class ORDERS	
orderNo	char (10)	Primary Key
order_date	date	
order_discount	number (7,2)	
shipping_method	varchar2 (15)	CHECK(shipping_method IN ('Ground', 'Freight', 'Priority', 'Overnight'))
shipping_date	date	
delivery_date	date	
order_status	varchar2 (25)	CHECK (order_status IN('in process', 'in transit', 'cancelled', 'fulfilled', 'not fulfilled', 'backordered'))
ord_status_notes	varchar2 (40)	
FD: orderno --> orderno, order_date, order_discount, shipping_method, shipping_date, delivery_date, orders_status, ord_status_notes		
PERSONNEL_MANAGERS	Relation representing the entity class PERSONNEL_MANAGERS	
empID	char (10)	Foreign key empdID references EMPLOYEES empID
yrs_exp	number (2)	
FD: empID --> empID, yrs_exp		
PLACE_ORDERS	Relation representing the entity class PLACE_ORDERS	
orderNo	char (10)	Foreign key orderID references ORDERS orderID
siteID	char (8)	Foreign key clientID referenced

		CLIENT_SITES clientID
CSR_empID	char (10)	Foreign key CSR_empdID references CSR_EMP empID
FD: orderNo, siteID, CSR_empID --> orderNo, siteID, CSR_empID		
PROBLEM_ACTION_ITEMS	Relation representing the entity class PROBLEM_ACTION_ITEMS	
incidentNo	char (10)	Foreign key incidentNo references PROBLEM_INCIDENTS incidentNo
action_item	varchar2 (40)	
due_date	date	
action_status	varchar2 (25)	
FD: incidentNo, action_item --> incidentNo, action_item, due_date, action_status		
PROBLEM_COMMENTS	Relation representing the entity class PROBLEM_COMMENTS	
incidentNo	char (10)	Foreign key incidentNo references PROBLEM_INCIDENTS incidentNo
comment	varchar2 (100)	
FD: incidentNo, comment --> incidentNo, comment		
PROBLEM_INCIDENTS	Relation representing the entity class PROBLEM_INCIDENTS	
incidentNo	char (10)	Primary Key
incidentDate	date	
reviewDate	date	
receivedDate	date	
incident_description	varchar2 (25)	
incident_status	varchar2 (20)	CHECK (incident_status IN('Open', 'Work in Progress', 'Awaiting Client', 'Resolved', 'Closed'))
FD: incidentNo --> incidentNo, incidentDate, ReviewDate, ReceivedDate, incident_description, incident_status		
PRODLINE_MANAGERS	Relation representing the entity class PRODLINE_MANAGERS	
empID	char (10)	Foreign key empdID references EMPLOYEES empID

prod_lineNo	char(6)	Foreign key prod_lineNo references PRODUCT_LINES lineNo
FD: empID --> empID, prod_lineNo		
PRODUCT_GROUPS	Relation representing the entity class PRODUCT_GROUPS	
groupID	char (5)	Primary Key
groupName	varchar2 (25)	UNIQUE, NOT NULL
numOfCSRs	varchar2 (25)	
FD: groupID --> groupID, groupName, numOfCSRs		
FD: groupName--> groupID, groupName, numOfCSRs		
PRODUCT_LINES	Relation representing the entity class PRODUCT_LINES	
lineNo	char (6)	Primary Key
lineName	varchar2 (25)	LineName UNIQUE and NOT NULL
line_beginDate	date	
line_notes	varchar2 (25)	
line_status	varchar2 (25)	
line_retireDate	date	
groupID	char (5)	Foreign key groupID references PRODUCT_GROUPS groupID
FD: lineNo --> lineNo, lineName, line_beginDate, line_notes, line_status, line_retireDate, groupID		
FD: lineName --> lineNo, lineName, line_beginDate, line_notes, line_status, line_retireDate, groupID		
PRODUCT_PRICING	Relation representing the entity class PRODUCT_PRICING	
prodID	char (10)	Primary key, Foreign Key prodID references PRODUCTS prodID
state	char (2)	Primary key
listPrice	number (7,2)	
FD: prodID, state --> prodID, state, listPrice		
PRODUCTS	Relation representing the entity class PRODUCTS	
prodID	char (10)	Primary Key
prodname	varchar2 (25)	Unique, NOT NULL
brand	varchar2 (25)	
avg_costperunit	number (7,2)	CHECK (avg_costperunit > 0)

launch_date	date	
prod_status	varchar2 (20)	
retire_date	date	
lineNo	char (6)	Foreign key references PRODUCT_LINES lineNo
FD: prodID --> prodID, prodname, brand, avg_costperunit, launch_date, prod_status, retire_date, lineNo		
FD: prodname --> prodID, prodname, brand, avg_costperunit, launch_date, prod_status, retire_date, lineNo		
PROMOSTATE	Relation representing the entity class PROMOSTATE	
promoID	char (10)	Primary key, Foreign key promoID references PROMOTIONS promoID
state	char (2)	Primary key
budget	number (12,2)	
promo_start_date	date	
promo_end_date	date	
FD: promoID, state --> promoID, state, budget, promo_start_date, promo_end_date		
PROMOTIONS	Relation representing the entity class PROMOTIONS	
promoID	char (10)	Primary Key
promoname	varchar2 (25)	
promodescription	varchar2 (25)	
prodID	char (10)	Foreign key prodID references PRODUCTS prodID
mgrID	char (10)	Foreign key mgrID references PRODLINE_MANAGERS empID
FD: promoID --> promoID, promoname, promodescription, prodID, mgrID		
REGIONS	Relation representing the entity class REGIONS	
regionID	char (5)	Primary Key
regionName	varchar2 (25)	
mgr_id	char (10)	Foreign key mgr_id references PERSONNEL_MANAGERS empID, assuming that each region has only 1 manager
FD: regionID --> regionID, regionName, mgr_id		
REPORT_PROBLEMS	Relation representing the entity class	

	REPORT_PROBLEMS	
clientID	char (10)	Foreign key clientID references CLIENTS clientID
CSR_empID	char (10)	Foreign key CSR_empID references CSR_EMP empID
incidentNo	char (10)	Foreign key incidentNo References PROBLEM_INCIDENTS incidentNo
FD: clientID, CSR_empID, incidentNo --> clidentID, CSR_empID, incidentNo		
SALARY_EMPLOYEES	Relation representing the entity class SALARY_EMPLOYEES	
empID	char (10)	Foreign key empID references EMPLOYEES empID
salary	number(10,2)	
FD: empID --> empID, salary		
FD:		
SPECIALIZE	Relation representing the entity class SPECIALIZE	
groupID	char (5)	Primary key, Foreign key groupID references PRODUCT_GROUPS groupID
CSR_empid	char (10)	Primary key, foreign key CSR_empID references CSR_EMP empid
sp_start_date	date	
sp_end_date	date	
sp_status	varchar2 (7)	CHECK (sp_status IN('active', 'inactive'))
FD: groupID, CSR_empid		
SUPERVISE_LINES	Relation representing the entity class SUPERVISE_LINES	
lineNo	char (6)	Primary key, Foreign key lineNo references PRODUCT_LINES lineNo
mgr_id	char (10)	Primary key, Foreign key mgr_id references PRODLINE_MANAGERS empid
sup_start_date	date	
sup_end_date	date	
supervise_status	varchar2 (7)	CHECK (supervise_status IN('active', 'inactive'))

FD: lineNo, mgr_id --> lineNo, mgr_id, sup_start_date, sup_end_date, supervice_status		
TAKE_TRAINING	Relation representing the entity class TAKE_TRAINING	
empid	char (10)	Foreign key empid references EMPLOYEES empId
sessionID	char (10)	Foreign key prodID references PRODUCTS prodID
FD: empID, sessionID --> empID, sessionID		
TEAMS	Relation representing the entity class TEAMS	
teamid	char (7)	Primary key
teamName	varchar2 (25)	NOT NULL
local_workarea	varchar2 (25)	
team_leadID	char (10)	Foreign key team_LeadID references EMPLOYEES empID
FD: teamid --> teamid, teamName, local_workarea, team_leadID		
TRAINING_SESSIONS	Relation representing the entity class TRAINING_SESSIONS	
sessionID	char (10)	Primary Key
location	varchar2 (25)	
startDate	date	
endDate	date	
instructor_id	char (10)	Foreign key instructor_id references EMPLOYEES empid, NOT NULL
courseNo	char (6)	Foreign key courseNo references COURSES courseNo, NOT NULL
FD: sessionID --> sessionID, location, startDate, endDate, instructor_id, courseNo		
VISITLOG	Relation representing the entity class VISITLOG	
visit_date	date	Primary Key
CSR_empID	char (10)	Foreign key CSR_empID references CSR_EMP empID, Primary key
siteID	char (8)	Foreign key siteID references CLIENT_SITES siteID, Primary key
hours_worked	number(4,2)	
FD: visit_date, CSR_empID, siteid --> visit_date, CSR_empID, siteid, hours_worked		

Appendix to Chapter 3

SQL Table Creation and Sequences

```
                    /* DROP TABLE SEQUENCE SCRIPT */


drop sequence AREA_SEQ;

drop sequence BRANCH_SEQ;

drop sequence CLIENTS_SEQ;

drop sequence CLIENTSITE_SEQ;

drop sequence COMMENT_SEQ;

drop sequence COURSE_SEQ;

drop sequence EMPLOYEES_SEQ;

drop sequence INCIDENT_SEQ;

drop sequence ORDERS_SEQ;

drop sequence PGROUP_SEQ;

drop sequence PLINE_SEQ;

drop sequence PRODUCT_SEQ;

drop sequence PROMO_SEQ;

drop sequence REGION_SEQ;

drop sequence SESSION_SEQ;

drop sequence TEAMS_SEQ;
```

```
DROP TABLE FEEDBACK;

DROP TABLE PROMOSTATE;

DROP TABLE PROMOTIONS;

DROP TABLE PRODUCT_PRICING;

DROP TABLE SUPERVISE_LINES;

DROP TABLE SPECIALIZE;

DROP TABLE VISITLOG;

DROP TABLE ASSIGN_TEAMS_TO_CLIENTS;

DROP TABLE FORM_TEAMS;

DROP TABLE TEAMS;

DROP TABLE REPORT_PROBLEMS;

DROP TABLE PROBLEM_ACTION_ITEMS;

DROP TABLE PROBLEM_COMMENTS;

DROP TABLE PLACE_ORDERS;

DROP TABLE ORDER_DETAILS;

DROP TABLE PRODUCTS;

DROP TABLE TAKE_TRAINING;

DROP TABLE TRAINING_SESSIONS;

DROP TABLE CLIENT_SITES;

ALTER TABLE BRANCHES DROP CONSTRAINT mainbranch_fk;

DROP TABLE MAIN_BRANCHES;

DROP TABLE REGIONS;

DROP TABLE PERSONNEL_MANAGERS;

DROP TABLE SALARY_EMPLOYEES;

DROP TABLE HOURLY_EMPLOYEES;

DROP TABLE PRODLINE_MANAGERS;

DROP TABLE FUNCTIONAL_MANAGERS;
DROP TABLE CSR_EMP;

DROP TABLE PRODUCT_LINES;

DROP TABLE PRODUCT_GROUPS;

DROP TABLE AREAS;

DROP TABLE PROBLEM_INCIDENTS;

DROP TABLE COURSES;

DROP TABLE ORDERS;

DROP TABLE CLIENTS;

DROP TABLE EMPLOYEES;

DROP TABLE BRANCHES;

                   /* BEGIN TABLE CREATION SCRIPT */


CREATE TABLE AREAS

(

area_id char(5),

area_name varchar2(25) NOT NULL,

area_description varchar2(25),

CONSTRAINT area_pk PRIMARY KEY (area_id),

CONSTRAINT areaName_unique UNIQUE (area_name)

);


CREATE TABLE CLIENTS

(

clientID char (10),
orgname varchar2 (25) NOT NULL,

contact_name varchar2 (25),

contact_email varchar2 (25),

contact_phone char (10),

contact_title varchar2 (25),

city varchar2 (25),

street varchar2 (25),

state char (2),

zip char (5),

corp_pricing_discount number (3,2),

CONSTRAINT clients_pk PRIMARY KEY (clientID),

CONSTRAINT clients_percdiscount_check CHECK (corp_pricing_discount
BETWEEN 0 AND 1) --Percentage discount

);



CREATE TABLE CLIENT_SITES

(

siteid char (8),

site_name varchar2 (25) NOT NULL,

street varchar2 (25),

city varchar2 (25),

state char (2),

zip char (5),

clientID char (10) NOT NULL,

area_ID char (5) NOT NULL,

CONSTRAINT client_sites_pk PRIMARY KEY (siteid),
FOREIGN KEY (clientID) REFERENCES CLIENTS (clientID),

FOREIGN KEY (area_ID) REFERENCES AREAS (area_ID)

);



CREATE TABLE ORDERS

(

orderNo char(10),

order_date date,

order_discount number(7,2),

shipping_method varchar2(15) CHECK(shipping_method IN ('Ground',
'Freight', 'Priority', 'Overnight')),

shipping_date date ,

delivery_date date,

order_status varchar2(25) CHECK (order_status IN('in process', 'in
transit', 'cancelled', 'fulfilled', 'not fulfilled', 'backordered')),

ord_status_notes varchar2(40),

CONSTRAINT orderNo_pk PRIMARY KEY (orderNo)

);



CREATE TABLE COURSES

(

courseNo char(6),

courseName varchar2(25) NOT NULL,

courseDesc varchar2(40),

internal_cost number(7,2),
course_type varchar2(25),

CONSTRAINT courses_pk PRIMARY KEY (courseNo),

CONSTRAINT courseName_unique UNIQUE (courseName)

);



CREATE TABLE PROBLEM_INCIDENTS

(

incidentNo char(10),

incidentDate date,

reviewDate date,

receivedDate date,

incident_description varchar2(40),

incident_status varchar(20) CHECK (incident_status IN('Open', 'Work in
Progress', 'Awaiting Client', 'Resolved', 'Closed')),

CONSTRAINT incidentNo_pk PRIMARY KEY (incidentNo)

);



CREATE TABLE PRODUCT_GROUPS

(

groupID char(5),

groupName varchar2(25)NOT NULL,

numOfCSRs varchar2(25),

CONSTRAINT groupID_pk PRIMARY KEY (groupID),

CONSTRAINT groupName_unique UNIQUE (groupName)

);
```

```
CREATE TABLE PRODUCT_LINES

(

lineNo char(6),

lineName varchar2(25) NOT NULL,

line_beginDate date,

line_notes varchar2(25),

line_status varchar2(25),

line_retireDate date,

groupid char (5),

CONSTRAINT lineNo_pk PRIMARY KEY (lineNo),

CONSTRAINT lineName_unique UNIQUE (lineName)

);



CREATE TABLE BRANCHES

(

BranchNo char (5),

BranchName varchar2 (25) NOT NULL,

type varchar2 (5) CHECK (type IN ('local', 'main')),

street varchar2 (25),

city varchar2 (25),

state char (2),

zip char(5),

Main_branchno char(5),

CONSTRAINT branches_pk PRIMARY KEY (BranchNo),
CONSTRAINT BranchName_unique UNIQUE (BranchName)
);


CREATE TABLE EMPLOYEES
(
empID char(10),
lname varchar2(25)NOT NULL,
fname varchar2(25)NOT NULL,
DOB date NOT NULL,
phone char(10),
SSN char (9),
email varchar2(50),
gender varchar2(20),
street varchar2(25),
city varchar2(25),
state char(2),
zip char(5),
position varchar2(25),
branchNo char(5),
CONSTRAINT SSN_unique UNIQUE (SSN),
CONSTRAINT empID_pk PRIMARY KEY (empID),
FOREIGN KEY (branchNo) REFERENCES BRANCHES (branchNo)
);


CREATE TABLE CSR_EMP
```

```
(

empID char(10),

csr_status varchar2(25),

seniority_rank varchar2(10) CHECK (seniority_rank IN
('Senior','Junior','Associate')),

actual_sales number(10,2),

numOfClients number(10),

numofAreas number(10),

CONSTRAINT csrID_pk PRIMARY KEY (empID),

FOREIGN KEY (empID) REFERENCES EMPLOYEES (empID)

);



CREATE TABLE FUNCTIONAL_MANAGERS

(

empID char(10),

mgr_type char(5),

CONSTRAINT functID_pk PRIMARY KEY (empID),

FOREIGN KEY (empID) REFERENCES EMPLOYEES (empID)

);



CREATE TABLE PERSONNEL_MANAGERS

(

empID char(10),

yrs_exp number(2),

CONSTRAINT pers_pk PRIMARY KEY (empID),
FOREIGN KEY (empID) REFERENCES EMPLOYEES (empID)
);


CREATE TABLE PRODLINE_MANAGERS
(
empID char(10),
prod_lineNo char(6),
CONSTRAINT line_pk PRIMARY KEY (empID),
FOREIGN KEY (empID) REFERENCES EMPLOYEES (empID),
FOREIGN KEY (prod_lineNo) REFERENCES PRODUCT_LINES (lineNo)
);


CREATE TABLE HOURLY_EMPLOYEES
(
empID char(10),
wage_rate number(5,2),
CONSTRAINT hour_pk PRIMARY KEY (empID),
FOREIGN KEY (empID) references EMPLOYEES (empID)
);

CREATE TABLE SALARY_EMPLOYEES
(
empID char(10),
salary number(10,2),
CONSTRAINT sal_pk PRIMARY KEY (empID),
FOREIGN KEY (empID) references EMPLOYEES (empID)
);


CREATE TABLE REGIONS
(
regionID char(5),
regionName varchar2 (25),
mgr_id char(10),
CONSTRAINT regions_pk PRIMARY KEY (regionID),
FOREIGN KEY (mgr_id) REFERENCES PERSONNEL_MANAGERS (empID)
);


CREATE TABLE MAIN_BRANCHES
(
BranchNo char (5),
regionID char (5) NOT NULL,
CONSTRAINT main_branches_pk PRIMARY KEY (BranchNo),
FOREIGN KEY (branchNo) REFERENCES BRANCHES(branchNo),
FOREIGN KEY (regionID) REFERENCES REGIONS(regionID)
);

ALTER TABLE BRANCHES
ADD CONSTRAINT mainbranch_fk
FOREIGN KEY (Main_branchno) REFERENCES MAIN_BRANCHES (BranchNo);
```

```
CREATE TABLE TRAINING_SESSIONS

(

sessionID char(10),

location varchar2(25),

startDate date,

endDate date,

instructor_id char(10)NOT NULL,

courseNo char(6)NOT NULL,

CONSTRAINT sessionID_pk PRIMARY KEY (sessionID),

FOREIGN KEY (instructor_id) REFERENCES EMPLOYEES (empID),

FOREIGN KEY (courseNo) REFERENCES COURSES (courseNo)

);



CREATE TABLE TAKE_TRAINING

(

empID char(10),

sessionID char(10),

CONSTRAINT take_training_pk PRIMARY KEY (empID, sessionID),

FOREIGN KEY (empID) REFERENCES EMPLOYEES (empID),

FOREIGN KEY (sessionID) REFERENCES TRAINING_SESSIONS (sessionID)

);



CREATE TABLE PRODUCTS

(
prodID char(10),

prodname varchar2(25) NOT NULL,

brand varchar2(25),

avg_costperunit number(7,2) CHECK (avg_costperunit > 0),

launch_date date,

prod_status varchar2(20),

retire_date date,

lineNo char(6),

CONSTRAINT prodID_pk PRIMARY KEY (prodID),

CONSTRAINT prodname_unique UNIQUE (prodname),

FOREIGN KEY (lineNo) REFERENCES PRODUCT_LINES (lineNo)

);



CREATE TABLE ORDER_DETAILS

(

orderNo char(10),

prodID char(10),

quantity number(5),

unitPrice number(12,2),

prod_discount number(7,2),

CONSTRAINT order_deatils_pk PRIMARY KEY (orderNo, prodID),

FOREIGN KEY (orderNo) REFERENCES ORDERS (orderNo),

FOREIGN KEY (prodID) REFERENCES PRODUCTS (prodID)

);
```

```
CREATE TABLE PLACE_ORDERS

(

orderNo char(10),

siteID char(8),

CSR_empID char(10),

CONSTRAINT place_orders_pk PRIMARY KEY (orderNo, siteID, CSR_empID),

FOREIGN KEY (orderNo) REFERENCES ORDERS (orderNo),

FOREIGN KEY (siteID) REFERENCES CLIENT_SITES (siteID),

FOREIGN KEY (CSR_empID) REFERENCES CSR_EMP (empID)

);



CREATE TABLE PROBLEM_COMMENTS

(

incidentNo char(10),

comments varchar2(100),

CONSTRAINT problem_comments_pk PRIMARY KEY (incidentNo, comments),

FOREIGN KEY (incidentNo) REFERENCES PROBLEM_INCIDENTS(incidentNo)

);



CREATE TABLE PROBLEM_ACTION_ITEMS

(

incidentNo char(10),

action_item varchar2(40),

due_date date,

action_status varchar2(25),
CONSTRAINT problem_action_items PRIMARY KEY (incidentNo, action_item),

FOREIGN KEY (incidentNo) REFERENCES PROBLEM_INCIDENTS(incidentNo)

);



CREATE TABLE REPORT_PROBLEMS

(

clientID char(10),

CSR_empID char(10),

incidentNo char(10),

CONSTRAINT report_problems PRIMARY KEY (clientID, CSR_empID,
incidentNo),

FOREIGN KEY (clientID) REFERENCES CLIENTS(clientID),

FOREIGN KEY (CSR_empID) REFERENCES CSR_EMP(empID),

FOREIGN KEY (incidentNo) REFERENCES PROBLEM_INCIDENTS(incidentNo)

);



CREATE TABLE TEAMS

(

teamid char(7),

teamName varchar2(25) NOT NULL,

local_workarea varchar2(25),

team_leadID char(10),

CONSTRAINT teamid_pk PRIMARY KEY (teamid),

FOREIGN KEY (team_leadID) REFERENCES EMPLOYEES (empID)

);
```

```
CREATE TABLE FORM_TEAMS

(

teamid char(7),

CSR_empID char(10),

CONSTRAINT fomr_teams_pk PRIMARY KEY (teamid, CSR_empID),

FOREIGN KEY (teamid) REFERENCES TEAMS(teamid),

FOREIGN KEY (CSR_empID) REFERENCES CSR_EMP(empID)

);



CREATE TABLE ASSIGN_TEAMS_TO_CLIENTS

(

siteID char(8),

teamID char(7),

begin_date date,

end_date date,

status varchar2(10) CHECK (status IN('active', 'inactive')),

CONSTRAINT assignteam_pk PRIMARY KEY (siteID, teamID),

FOREIGN KEY (siteID) REFERENCES CLIENT_SITES (siteID),

FOREIGN KEY (teamID) REFERENCES TEAMS (teamID)

);



CREATE TABLE VISITLOG

(
visit_date date,

CSR_empID char(10),

siteID char(8),

hours_worked number(4,2),

CONSTRAINT visit_date_pk PRIMARY KEY (visit_date, CSR_empID, siteID),

FOREIGN KEY (CSR_empID) REFERENCES CSR_EMP (empID),

FOREIGN KEY (siteID) REFERENCES CLIENT_SITES (siteID)

);



CREATE TABLE SPECIALIZE

(

groupID char(5),

CSR_empID char(10),

sp_start_date date,

sp_end_date date,

sp_status varchar2(7) CHECK (sp_status IN('active', 'inactive')),

CONSTRAINT special_pk PRIMARY KEY (groupID, CSR_empID),

FOREIGN KEY (CSR_empID) references CSR_EMP (empID),

FOREIGN KEY (groupID) references PRODUCT_GROUPS (groupID)

);



CREATE TABLE SUPERVISE_LINES

(

lineNo char(6),

mgr_id char(10),
sup_start_date date,

sup_end_date date,

supervise_status varchar2(7) CHECK (supervise_status IN('active',
'inactive')) ,

CONSTRAINT super_pk PRIMARY KEY (lineNo, mgr_ID),

FOREIGN KEY (lineNo) REFERENCES PRODUCT_LINES (lineNo),

FOREIGN KEY (mgr_id) REFERENCES PRODLINE_MANAGERS (empID)

);



CREATE TABLE PRODUCT_PRICING

(

prodID char(10),

state char (2),

listPrice number(7,2),

CONSTRAINT price_pk PRIMARY KEY (prodID, state),

FOREIGN KEY (prodID) REFERENCES PRODUCTS (prodID)

);


CREATE TABLE PROMOTIONS

(

promoID char(10),

promoname varchar2(25),

promodescription varchar2(25),

prodID char(10),

mgrID char(10),

CONSTRAINT promoID_pk PRIMARY KEY (promoID),
FOREIGN KEY (prodID) REFERENCES PRODUCTS (prodID),

FOREIGN KEY (mgrID) REFERENCES PRODLINE_MANAGERS (empID)

);


CREATE TABLE PROMOSTATE

(

promoID char(10),

state char(2),

budget number (12,2),

promo_start_date date,

promo_end_date date,

CONSTRAINT promostate_pk PRIMARY KEY (promoID, state),

FOREIGN KEY (promoID) REFERENCES PROMOTIONS (promoID)

);


CREATE TABLE FEEDBACK

(

siteid char (8),

prodID char(10),

commentNo char(6),

comments varchar2(50),

feedback_date date,

CONSTRAINT feedback_pk PRIMARY KEY (commentNo),

FOREIGN KEY (siteID) REFERENCES CLIENT_SITES (siteID),

FOREIGN KEY (prodID) REFERENCES PRODUCTS (prodID)

);
```

```
commit;


                 /* BEGIN SEQUENCE CREATION SCRIPT */

CREATE SEQUENCE AREA_SEQ
     INCREMENT BY 1
START WITH 1001
MAXVALUE 9999;

CREATE SEQUENCE REGION_SEQ
     INCREMENT BY 1
START WITH 1001
MAXVALUE 9999;

CREATE SEQUENCE BRANCH_SEQ
     INCREMENT BY 1
START WITH 1001
MAXVALUE 9999;

CREATE SEQUENCE PROMO_SEQ
     INCREMENT BY 1
START WITH 100000001
MAXVALUE 999999999;

CREATE SEQUENCE PLINE_SEQ
     INCREMENT BY 1
```

```
START WITH 10001
MAXVALUE 99999;

CREATE SEQUENCE PGROUP_SEQ
      INCREMENT BY 1
START WITH 1001
MAXVALUE 9999;

CREATE SEQUENCE PRODUCT_SEQ
      INCREMENT BY 1
START WITH 1000001
MAXVALUE 9999999;

CREATE SEQUENCE CLIENTS_SEQ
      INCREMENT BY 1
START WITH 100000001
MAXVALUE 999999999;

CREATE SEQUENCE CLIENTSITE_SEQ
      INCREMENT BY 1
START WITH 100001
MAXVALUE 999999;

CREATE SEQUENCE INCIDENT_SEQ
      INCREMENT BY 1
START WITH 100000001
MAXVALUE 999999999;
```

```
CREATE SEQUENCE SESSION_SEQ
     INCREMENT BY 1
START WITH 10000001
MAXVALUE 99999999;

CREATE SEQUENCE COURSE_SEQ
     INCREMENT BY 1
START WITH 1001
MAXVALUE 9999;

CREATE SEQUENCE ORDERS_SEQ
     INCREMENT BY 1
START WITH 100000001
MAXVALUE 999999999;

CREATE SEQUENCE EMPLOYEES_SEQ
     INCREMENT BY 1
START WITH 10000001
MAXVALUE 99999999;

CREATE SEQUENCE COMMENT_SEQ
     INCREMENT BY 1
START WITH 10001
MAXVALUE 99999;

CREATE SEQUENCE TEAMS_SEQ
```

```
     INCREMENT BY 1
START WITH 100001
MAXVALUE 999999;

commit;



                  /* BEGIN TRIGGER CREATION SCRIPT */

CREATE OR REPLACE TRIGGER AREA_ID_generator
BEFORE INSERT
ON AREAS
FOR EACH ROW

DECLARE
    temp_area_id AREAS.area_id%type;

BEGIN
     SELECT 'A'||AREA_SEQ.nextval INTO temp_area_id FROM dual;
     :new.area_id := temp_area_id;
END;
/



CREATE OR REPLACE TRIGGER EMP_ID_generator
BEFORE INSERT
ON EMPLOYEES
```

```
FOR EACH ROW

DECLARE
    temp_emp_id employees.empid%type;

BEGIN
      SELECT 'EM'||EMPLOYEES_SEQ.nextval INTO temp_emp_id FROM dual;
      :new.empid := temp_emp_id;
END;
/

CREATE OR REPLACE TRIGGER CLIENT_ID_generator
BEFORE INSERT
ON CLIENTS
FOR EACH ROW

DECLARE
    temp_client_id CLIENTS.clientid%type;

BEGIN
      SELECT 'C'||CLIENTS_SEQ.nextval INTO temp_client_id FROM dual;
      :new.clientid := temp_client_id;
END;
/


CREATE OR REPLACE TRIGGER BRANCH_No_generator
```

```
BEFORE INSERT

ON BRANCHES

FOR EACH ROW


DECLARE

    temp_branchno BRANCHES.branchno%type;


BEGIN

     SELECT 'B'||BRANCH_SEQ.nextval INTO temp_branchno FROM dual;

     :new.branchno := temp_branchno;

END;

/




CREATE OR REPLACE TRIGGER CLIENT_SITE_ID_generator

BEFORE INSERT

ON CLIENT_SITES

FOR EACH ROW


DECLARE

    temp_client_site_id CLIENT_SITES.siteid%type;


BEGIN

     SELECT 'CS'||CLIENTSITE_SEQ.nextval INTO temp_client_site_id FROM
dual;

     :new.siteid := temp_client_site_id;

END;
```

```
/

CREATE OR REPLACE TRIGGER REGION_ID_generator
BEFORE INSERT
ON REGIONS
FOR EACH ROW

DECLARE
    temp_region_id REGIONS.regionid%type;

BEGIN
     SELECT 'R'||REGION_SEQ.nextval INTO temp_region_id FROM dual;
     :new.regionid := temp_region_id;
END;
/

CREATE OR REPLACE TRIGGER PROMO_ID_generator
BEFORE INSERT
ON PROMOTIONS
FOR EACH ROW

DECLARE
    temp_promo_id PROMOTIONS.promoid%type;

BEGIN
     SELECT 'P'||PROMO_SEQ.nextval INTO temp_promo_id FROM dual;
     :new.promoid := temp_promo_id;
```

```
END;
/

CREATE OR REPLACE TRIGGER PLINE_ID_generator
BEFORE INSERT
ON PRODUCT_LINES
FOR EACH ROW

DECLARE
    temp_pline_id PRODUCT_LINES.lineNo%type;

BEGIN
     SELECT 'L'||PLINE_SEQ.nextval INTO temp_pline_id FROM dual;
     :new.lineNo := temp_pline_id;
END;
/

CREATE OR REPLACE TRIGGER PGROUP_ID_generator
BEFORE INSERT
ON PRODUCT_GROUPS
FOR EACH ROW

DECLARE
    temp_pgroup_id PRODUCT_GROUPS.groupID%type;

BEGIN
     SELECT 'G'||PGROUP_SEQ.nextval INTO temp_pgroup_id FROM dual;
```

```
      :new.groupID := temp_pgroup_id;
END;
/

CREATE OR REPLACE TRIGGER PRODUCT_ID_generator
BEFORE INSERT
ON PRODUCTS
FOR EACH ROW

DECLARE
    temp_prod_id PRODUCTS.prodID%type;

BEGIN
      SELECT 'PRD'||PRODUCT_SEQ.nextval INTO temp_prod_id FROM dual;
      :new.prodID := temp_prod_id;
END;
/

CREATE OR REPLACE TRIGGER INCIDENT_No_generator
BEFORE INSERT
ON PROBLEM_INCIDENTS
FOR EACH ROW

DECLARE
    temp_inc_id PROBLEM_INCIDENTS.incidentNo%type;

BEGIN
```

```
      SELECT 'I'||INCIDENT_SEQ.nextval INTO temp_inc_id FROM dual;

      :new.incidentNo := temp_inc_id;

END;

/


CREATE OR REPLACE TRIGGER SESSION_ID_generator

BEFORE INSERT

ON TRAINING_SESSIONS

FOR EACH ROW


DECLARE

    temp_sess_id TRAINING_SESSIONS.sessionID%type;


BEGIN

      SELECT 'TS'||SESSION_SEQ.nextval INTO temp_sess_id FROM dual;

      :new.sessionID := temp_sess_id;

END;

/


CREATE OR REPLACE TRIGGER COURSE_No_generator

BEFORE INSERT

ON COURSES

FOR EACH ROW


DECLARE

    temp_co_id COURSES.courseNo%type;
```

```
BEGIN
      SELECT 'CO'||COURSE_SEQ.nextval INTO temp_co_id FROM dual;
      :new.courseNo := temp_co_id;
END;
/

CREATE OR REPLACE TRIGGER ORDER_No_generator
BEFORE INSERT
ON ORDERS
FOR EACH ROW

DECLARE
    temp_o_no ORDERS.orderNo%type;

BEGIN
      SELECT 'O'||ORDERS_SEQ.nextval INTO temp_o_no FROM dual;
      :new.orderNo := temp_o_no;
END;
/

CREATE OR REPLACE TRIGGER COMMENT_ID_generator
BEFORE INSERT
ON FEEDBACK
FOR EACH ROW

DECLARE
    temp_f_id FEEDBACK.commentNo%type;
```

```
BEGIN
      SELECT 'F'||COMMENT_SEQ.nextval INTO temp_f_id FROM dual;
      :new.commentNo := temp_f_id;
END;
/

CREATE OR REPLACE TRIGGER TEAM_ID_generator
BEFORE INSERT
ON TEAMS
FOR EACH ROW

DECLARE
    temp_team_id TEAMS.teamid%type;

BEGIN
      SELECT 'T'||TEAMS_SEQ.nextval INTO temp_team_id FROM dual;
      :new.teamid := temp_team_id;
END;
/

commit;
```

Chapter 4: SQL Complex Queries and Explanations

Table Insertion Queries

Query 1

```
SELECT P.INCIDENTNO AS "Incident Number", P.INCIDENT_STATUS AS
"Incident Status", P.INCIDENTDATE AS "Incident Date", C.COMMENTS AS
"Problem Comments"

FROM PROBLEM_INCIDENTS P, PROBLEM_COMMENTS C

WHERE P.INCIDENTNO = C.INCIDENTNO AND P.INCIDENTDATE BETWEEN '01-JUL-
19' AND '30-JUL-19'

AND P.INCIDENT_STATUS = 'Open'

ORDER BY P.INCIDENTDATE;
```

This query finds all Problem Incidents with an "Open" status between a given time frame and displays the results ordered by the Incident Date. The query uses joins to match the data from the PROBLEM_INCIDENTS and PROBLEM_COMMENTS tables. This query would be useful to employees who may need to find a problem incident from the past that was not closed, and the Incident Number was forgotten.

Query 2

```
SELECT SHIPPING_METHOD AS "Shipping Method", to_char(avg(DELIVERY_DATE
- SHIPPING_DATE), '999.99') as "AVG Shipping Time",

(CASE WHEN avg(DELIVERY_DATE - SHIPPING_DATE) >= 4 THEN 'Slow'

WHEN avg(DELIVERY_DATE - SHIPPING_DATE) BETWEEN 2 AND 4 THEN 'Average'

WHEN avg(DELIVERY_DATE - SHIPPING_DATE) <= 2 THEN 'Fast'

END) AS "Delivery Speed"

FROM ORDERS

GROUP BY SHIPPING_METHOD;
```

This query calculates the average shipping times of each of the four shipping types (Ground, Freight, Priority, and Overnight) and displays their speed as "Average", "Fast" or "Slow" depending on the amount of days between the shipping date and delivery date. This query would help employees verify if the various shipping methods are being delivered within the stated time frames and to make sure the use of various shipping methods are being optimized.

Query 3

```
SELECT PL.LINENAME AS "Line Name", count(P.PRODID) AS "Products Sold
in Line", to_char(sum(OD.QUANTITY*OD.UNITPRICE), '$9,999,999.99') AS
"Total Revenue From Product",

RANK() OVER (PARTITION BY 'Line Name' ORDER BY
sum(OD.QUANTITY*OD.UNITPRICE) desc) AS "Sales Rank"

FROM ORDER_DETAILS OD, Products P, PRODUCT_LINES PL WHERE OD.PRODID =
P.PRODID AND P.LINENO = PL.LINENO

GROUP BY PL.LINENAME;
```

This query finds the number of products sold in each line along with the total revenue, and ranks the lines based on total revenue of products sold in the line. To calculate these columns, the ORDER_DETAILS, PRODUCTS, and PRODUCT_LINES tables are joined on PROD ID and LINENO. The function to_char is also used to put the Total Revenue column into a dollar format.

Query 4

```
SELECT C.CLIENTID AS "ClientID", C.ORGNAME "Organization",
SUM(QUANTITY * UNITPRICE) AS "Total Revenue"

FROM CLIENTS C

    JOIN CLIENT_SITES CS ON C.CLIENTID = CS.CLIENTID

    JOIN PLACE_ORDERS PO ON CS.SITEID = PO.SITEID

    JOIN ORDERS O ON PO.ORDERNO = O.ORDERNO

    JOIN ORDER_DETAILS OD ON PO.ORDERNO = OD.ORDERNO

WHERE O.ORDER_DATE >= '01-JAN-2019'

GROUP BY C.CLIENTID, C.ORGNAME;
```

This query displays clientID, organization name, and total revenue since January 1, 2019, for all clients that placed orders during that time period and ranks the clients by total revenue. Since orders are placed by client sites, the CLIENTS table was joined to CLIENT_SITES, which was then joined to ORDERS and ORDER_DETAILS, in order to determine quantity sold and unit price. The GROUP BY function was used to group by clientID and the aggregate function SUM() was used to determine total revenue for each clientID.

Query 5

```
SELECT CSR.EMPID AS "CSR ID", E.FNAME || ' ' || E.LNAME AS "CSR Name",
CSR.SENIORITY_RANK AS "Rank", coalesce(COUNT(PO.ORDERNO),0) AS "Num
Sales"

FROM CSR_EMP CSR

     JOIN EMPLOYEES E ON CSR.EMPID = E.EMPID

     LEFT JOIN PLACE_ORDERS PO ON CSR.EMPID = PO.CSR_EMPID

WHERE E.CITY = 'Tucson' AND E.STATE = 'AZ'

GROUP BY CSR.EMPID, E.LNAME, E.FNAME, CSR.SENIORITY_RANK

ORDER BY COUNT(PO.ORDERNO) desc;
```

This query displays empID, first and last names, seniority rank, and count of orders for all client-service representatives in Tucson, AZ. If a client-service representative does not have any orders yet, the coalesce function is used to display zero for that representative. The CSR_EMP and EMPLOYEES tables are joined to display descriptive information about the representative. LEFT JOIN is used to include representatives in the output query that do not have any orders. The output is sorted by count of orders in descending order.

Query 6

```
SELECT EXTRACT(MONTH FROM TS.STARTDATE) AS "Month", COUNT(TT.EMPID) AS
"Num Attendees"

FROM TRAINING_SESSIONS TS

     JOIN TAKE_TRAINING TT ON TS.SESSIONID = TT.SESSIONID

WHERE EXTRACT(YEAR FROM TS.STARTDATE) = '2019'

GROUP BY EXTRACT(MONTH FROM TS.STARTDATE)

HAVING COUNT(TT.EMPID) > (

     SELECT AVG(COUNT(TT.EMPID))

     FROM TRAINING_SESSIONS TS

          JOIN TAKE_TRAINING TT ON TS.SESSIONID = TT.SESSIONID

     WHERE EXTRACT(YEAR FROM TS.STARTDATE) < '2019'

     GROUP BY EXTRACT(MONTH FROM TS.STARTDATE));
```

This query finds one or more months in 2019 in which the count of training session attendees exceeds the average count of attendees in all months prior to 2019. The tables

TRAINING_SESSIONS and TAKE_TRAINING are joined to determine training session start date and attending employees. The EXTRACT() function and GROUP BY clause is used to extract and group the months in which the training sessions are held. The condition that the count for a given month be greater than the average of all prior months is held using the HAVING clause, which checks the values against an in-line select query that returns the average of all prior months.

Query 7

```
SELECT PL.LINENAME AS "Line Name", P.PRODNAME AS "Product", P.BRAND AS
"BRAND", to_char(P.AVG_COSTPERUNIT, '$999.99') AS "Average Unit Cost"

FROM PRODUCT_LINES PL

    JOIN PRODUCTS P ON PL.LINENO = P.LINENO

WHERE PL.LINENO = 'L10002'

AND P.AVG_COSTPERUNIT >= 4

AND P.LAUNCH_DATE >= PL.LINE_BEGINDATE

ORDER BY PL.LINENO, P.PRODNAME;
```

This query displays all products in a given product line that have been launched since the line begin date. The results are also restricted to products where the average unit cost is greater than a given amount. The PRODUCT_LINES and PRODUCTS tables were joined to determine products within a product line. The WHERE clause was used to specify the line number and average cost per unit, and to restrict output to products with a launch date greater than line begin date. Finally, the results were filtered by line number, product name.

Query 8

```
SELECT *

FROM

(

      SELECT PS.STATE, EXTRACT(YEAR FROM PS.PROMO_START_DATE) AS Year,
PS.Budget

      FROM PROMOSTATE PS

)

PIVOT

(
      AVG(BUDGET)

      FOR Year IN ('2017', '2018', '2019')

);
```

This query returns the average budget for promotions in every state per year. The PIVOT function is used to display the average budget amounts in a pivot table with rows as states and columns as years. The promotion information is queried from the PROMOSTATES table. The aggregate function AVG is used to calculate the average for each state for each year.

Query 9

```
SELECT C.CLIENTID AS "Client ID", CS.SITE_NAME AS "Client Site Name",
COALESCE(COUNT(FT.CSR_EMPID), 0) AS "Num CSRs"

FROM CLIENTS C

      LEFT JOIN CLIENT_SITES CS ON CS.CLIENTID = C.CLIENTID

      JOIN ASSIGN_TEAMS_TO_CLIENTS ATC ON ATC.SITEID = CS.SITEID

      JOIN FORM_TEAMS FT ON FT.TEAMID = ATC.TEAMID

GROUP BY ROLLUP(C.CLIENTID, CS.SITE_NAME);
```

This query displays clientID, client site name, and the count of client service reps for each client site. The ROLLUP function is used to display the total count for each clientID and the grand total for the whole table. The COALESCE function is used to display zero if a client site does not have any client service reps assigned to it. The tables CLIENTS, CLIENT_SITES, ASSIGN_TEAMS_TO_CLIENTS, and FORM_TEAMS were joined in order to determine the count of client service rep employees assigned to each client through teams.

Query 10

```
SELECT V.SITEID AS "Site ID", C.ORGNAME AS "Client Name", CS.SITE_NAME
AS "Site Name", sum(V.HOURS_WORKED) AS "Hours Worked"

FROM VISITLOG V, CLIENT_SITES CS, CLIENTS C

WHERE V.SITEID = CS.SITEID AND CS.CLIENTID = C.CLIENTID

AND V.VISIT_DATE BETWEEN '01-MAR-19' AND '31-MAR-19'

GROUP BY V.SITEID, CS.SITE_NAME, C.ORGNAME

ORDER BY sum(V.HOURS_WORKED) desc;
```

This query shows the total hours worked by all client service employees at a client site for a given time frame. It is important for TriannaCorp to make sure that each client site is receiving an adequate amount of customer service and this query can be used as an important metric in determining if an appropriate level of customer service is being provided. The query joins three tables on two fields in order to gather the needed information. the sum and group by functions are used to calculate the hours worked per site ID.

Chapter 5: Triggers and Procedures

PL/SQL Triggers / Procedures and Explanations

Trigger 1a

```
CREATE OR REPLACE TRIGGER csr_area_limits_assign_team

    BEFORE INSERT OR UPDATE OF teamID

    ON assign_teams_to_clients

    FOR EACH ROW

DECLARE

    area_count integer;

    CURSOR CSRS IS SELECT csr_empid FROM form_teams

        WHERE teamID = :new.teamID;

BEGIN

    FOR C IN CSRS LOOP

        area_count := 0;

        SELECT count(distinct area_ID) INTO area_count FROM
assign_teams_to_clients

        NATURAL JOIN client_sites

        WHERE status = 'active' AND teamID IN (

            SELECT teamID FROM form_teams where csr_empid =
C.csr_empid);

        IF area_count >= 5 THEN

            raise_application_error(-20010,

            'Error: Change not made. Customer service representative
'||C.csr_empid||' would have more than 5 areas.');

        END IF;

    END LOOP;
```

```
END;

/
```

Trigger 1b

```
CREATE OR REPLACE TRIGGER csr_area_limits_form_team

    BEFORE INSERT OR UPDATE

    ON form_teams

    FOR EACH ROW

DECLARE

    area_count integer := 0;

BEGIN

        SELECT count(distinct area_ID) INTO area_count FROM
assign_teams_to_clients

        NATURAL JOIN client_sites

        WHERE status = 'active' AND teamID IN (

            SELECT teamID FROM form_teams where csr_empid =
:new.csr_empid);

        IF area_count >= 5 THEN

            raise_application_error(-20011,

            'Error: Change not made. Customer service representative
'||:new.csr_empid||' would have more than 5 areas.');

        END IF;


END;

/
```

Explanation for Triggers csr_area_limits_assign_team and csr_area_limits_form_team:

TriannaCorp wants their customer service employees to be assigned to a maximum of 5 “areas”. In order to achieve this business rule, we are implementing a trigger on the Assign_Teams_To_Clients table and the Form_Teams table. If there is a change in either of these tables, then we need to check if this change will cause any customer service representatives to exceed the 5 area limit. For the trigger on the Assign_Teams_To_Clients table, we need to first determine which team is being affected. Then, we create a cursor for all the CSRs assigned to this particular team. For each CSR, we must count how many distinct areas they are assigned to by checking all the teams they are working on and which clients (and associated areas) those teams are representing. For the trigger on the Form_Teams table, we do a similar algorithm except only for the current customer service representative being inserted/updated into the Form_Teams table..

Trigger 2

```
CREATE OR REPLACE TRIGGER alert_excess_incidents

    BEFORE INSERT OR UPDATE OF clientID

    ON report_problems

    FOR EACH ROW

DECLARE

    month_count integer := 0;

    client_count integer := 0;

    cur_month number(2) := 0;

    cur_year number(4) := 0;


BEGIN



        SELECT EXTRACT(month FROM incidentDate), EXTRACT(year FROM
incidentDate) INTO cur_month, cur_year

            FROM problem_incidents P LEFT OUTER JOIN report_problems R

                ON P.incidentNo = R.incidentNo

                WHERE P.incidentNo = :new.incidentNo;

        SELECT count(incidentNo) INTO month_count
```

```
            FROM problem_incidents
                WHERE EXTRACT(month FROM incidentDate) = cur_month
                AND EXTRACT(year FROM incidentDate) = cur_year;
        IF month_count >= 5 THEN
            dbms_output.put_line ('Warning: There have been more than
5 incidents during the month of '||cur_month||', '||cur_year||'!');
        END IF;
        SELECT count(distinct incidentNo) INTO client_count FROM
problem_incidents
            NATURAL JOIN report_problems
            WHERE EXTRACT(month FROM incidentDate) = cur_month
                AND EXTRACT(year FROM incidentDate) = cur_year
                AND clientID = :new.clientID;
        IF client_count >= 1 THEN
            dbms_output.put_line ('Urgent Warning!!! There has been
more than 1 incident during the month of '||cur_month||',
'||cur_year||' for this client!!');
        END IF;


END;
/
```

Explanation for Trigger alert_excess_incidents:

TriannaCorp prides itself on good customer service and wants to stay on top of any incident reports. They have a business rule that if there are more than 5 incidents reported in any given month, they want management to be alerted so they can address the situation. If any client has 2 or more incidents in a month, this is especially concerning. In order to achieve this, we created a trigger on the Report_Problems table. Every time a new incident is reported or there is a change, we check the incident date and count how many incidents have occurred within that same month (and year). If the count exceeds 5, we give a warning message to the user. We also count how many incidents for this client in that same month. If this client has 2 or more incidents that month, we give an urgent warning message to the user.

Trigger 3a

```
CREATE OR REPLACE TRIGGER numOfClientSitesandAreas_trig

AFTER INSERT OR DELETE OR UPDATE OF teamID, siteID, status

ON assign_teams_to_clients


DECLARE

    site_count integer :=0;

    area_count integer :=0;

    CURSOR CSR_cur IS select empID FROM CSR_EMP join FORM_TEAMS

    ON empID = CSR_empID;


BEGIN


    FOR C IN CSR_cur LOOP

        site_count := 0;

        area_count := 0;

        SELECT COUNT(DISTINCT siteid) INTO site_count

            FROM ASSIGN_TEAMS_TO_CLIENTS NATURAL JOIN FORM_TEAMS

            where CSR_empID = C.empid

            AND LOWER(status) = 'active';

        SELECT count(distinct area_ID) INTO area_count FROM
assign_teams_to_clients

         NATURAL JOIN client_sites

            WHERE LOWER(status) = 'active' AND teamID IN (

                SELECT teamID FROM form_teams where csr_empid =
C.empid);

        UPDATE CSR_EMP SET numOfClients = site_count, numOfAreas =
area_count
```

```
        WHERE empID = C.empID;

    END LOOP;


END;

/
```

Trigger 3b

```
CREATE OR REPLACE TRIGGER formTeam_numOfSitesAreas_trig

AFTER INSERT OR DELETE OR UPDATE

ON form_teams


DECLARE

    site_count integer :=0;

    area_count integer :=0;

    CURSOR CSR_cur IS select empID FROM CSR_EMP join FORM_TEAMS

    ON empID = CSR_empID;


BEGIN


    FOR C IN CSR_cur LOOP

        site_count := 0;

        area_count := 0;

        SELECT COUNT(DISTINCT siteid) INTO site_count

            FROM ASSIGN_TEAMS_TO_CLIENTS NATURAL JOIN FORM_TEAMS

            where CSR_empID = C.empid

            AND LOWER(status) = 'active';

        SELECT count(distinct area_ID) INTO area_count FROM
assign_teams_to_clients
```

```
         NATURAL JOIN client_sites

            WHERE LOWER(status) = 'active' AND teamID IN (

                SELECT teamID FROM form_teams where csr_empid =
C.empid);

        UPDATE CSR_EMP SET numOfClients = site_count, numOfAreas =
area_count

        WHERE empID = C.empID;

    END LOOP;


END;

/
```

Explanation for Triggers numOfClientSitesandAreas_trig and formTeam_numOfSitesAreas_trig:

These triggers compute derived attributes for client service representatives. Anytime there is a change in team assignment or a formation of a team, the number of client sites and the number of areas for each CSR (who is currently on a team) is computed.

<u>Trigger 4</u>

```
CREATE OR REPLACE TRIGGER groupNumOfCSRs_trig

AFTER INSERT OR DELETE OR UPDATE OF groupID, CSR_empID, sp_status

ON specialize


DECLARE

    CSR_count integer :=0;


    CURSOR CUR IS select groupID FROM product_groups;


BEGIN


    FOR C IN CUR LOOP
```

```
        CSR_count := 0;

        SELECT COUNT(DISTINCT CSR_empID) INTO CSR_count

            FROM specialize

            where groupID = C.groupID

            AND LOWER(sp_status) = 'active';

        UPDATE product_groups SET numOfCSRs = CSR_count

        WHERE groupID = C.groupID;

    END LOOP;

END;

/
```

Explanation for Trigger groupNumOfCSRs_trig:

This trigger computes a derived attribute for product groups. Anytime there is a change in a CSR's group specialization or status of specialization, the number of active CSRs specializing in each each group is computed.

<u>Trigger 5</u>

```
CREATE OR REPLACE TRIGGER CSRsspecialization_trig

BEFORE INSERT OR UPDATE OF groupID, CSR_empID, sp_status

ON specialize

FOR EACH ROW


DECLARE

    group_count integer :=0;


BEGIN


    SELECT COUNT(DISTINCT groupID) INTO group_count

        FROM specialize
        where CSR_empID = :new.CSR_empID

        AND LOWER(sp_status) = 'active';

    IF group_count >= 3 THEN

        dbms_output.put_line ('Warning! This client service
representative is specializing in 3 or more groups!');

    END IF;


END;

/
```

Explanation for Trigger CSRSpecialization_trig:

This trigger computes how many groups the CSR is actively specializing in. If it is 3 or more, a warning is displayed to the user, as TriannaCorp tries to limit their CSRs to 3 active group specializations at one time.

Trigger 6

```
CREATE OR REPLACE TRIGGER MainBranches_trig

BEFORE INSERT OR UPDATE OF main_branchno

ON branches

FOR EACH ROW


DECLARE

    branch_count integer :=0;


BEGIN


    IF :new.main_branchno IS NOT NULL THEN

        SELECT COUNT(DISTINCT branchno) INTO branch_count

         FROM branches

            where main_branchno = :new.main_branchno;
        IF branch_count >= 25 THEN

            raise_application_error ('-20120', 'Error: This main
branch already has 25 or more local branches assigned to it.');

        END IF;

   END IF;


END;

/
```

Explanation for Trigger MainBranches_trig:

This trigger computes how many branches the main branch is currently managing. If this new branch assignment will cause the main branch to have more than 25 local branches assigned to, it raises an error and prevents this assignment.

Trigger 7

```
CREATE OR REPLACE TRIGGER TeamCount_trig

BEFORE INSERT OR UPDATE

ON form_teams

FOR EACH ROW


DECLARE

    CSR_count integer :=0;


BEGIN


    SELECT COUNT(DISTINCT CSR_empID) INTO CSR_count

        FROM form_teams

        where teamID = :new.teamID;

    IF CSR_count >= 10 THEN
```

```
        raise_application_error ('-20160', 'Error: This team already
has 10 CSRs assigned to it.');

    END IF;


END;

/
```

Explanation for Trigger TeamCount_trig:

This trigger computes how many members the current team has. If the team has 10 or members, it raises an error and prevents this assignment. TriannaCorp limits its teams to 10 members at most.

<u>Procedure</u>

```
CREATE OR REPLACE PROCEDURE AnnualActualSales_proc (year_p number) AS


sales csr_emp.actual_sales%type;

counter integer;

Cursor CSRs IS select empid from CSR_emp

FOR UPDATE OF actual_sales;

BEGIN


    IF to_char(year_p) > extract(year from sysdate) then

        raise_application_error(-20000, 'The year entered is in the
future.');

    END IF;


    For C in CSRs Loop

        sales :=0;

        SELECT count(orderno) INTO counter
        FROM csr_emp CS left join place_orders PO on CS.empid = 
PO.csr_empid

        WHERE empid = C.empid;

        IF counter > 0 THEN

            select coalesce(sum((quantity*unitPrice)*(1-
prod_discount)),0) into sales

                FROM order_details OD join orders O ON O.orderno = 
OD.orderno

                join place_orders P on P.orderno = O.orderno

                where csr_empid = C.empid and extract(year from 
order_date) = year_p

                group by csr_empid;

        END IF;

            UPDATE csr_emp set actual_sales = sales WHERE CURRENT OF 
CSRs;

    END LOOP;

    EXCEPTION WHEN NO_DATA_FOUND THEN

        sales := 0;

END;

/


exec AnnualActualSales_proc(2019);

show errors;
```

Explanation for Procedure AnnualActualSales_proc:

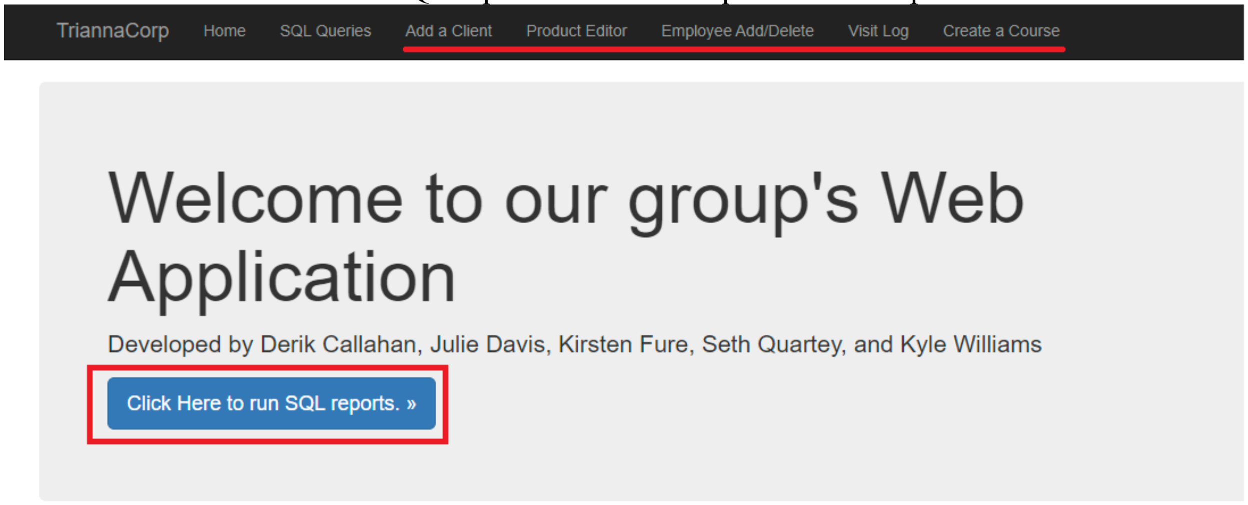
This trigger computes a derived attribute for the CSR_emp table. It calculates the annual sales for every CSR for the year passed as a parameter. TriannaCorp uses CSR's actual sales amounts to determine their annual bonuses and commissions. This procedure can be run on demand and calculates annual sales for the year passed.

Chapter 6: Interface and Reports

Web Interface and Reports User Walkthrough and Screen Captures

URL: https://mis531.com

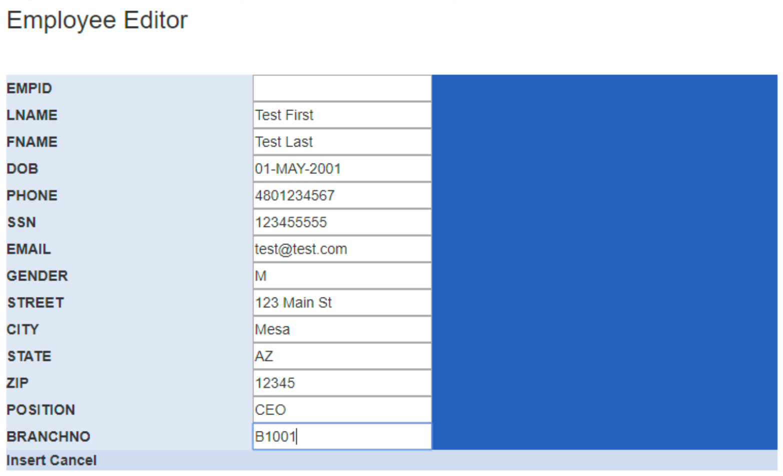
1. When website loads. Please notice the navigation bar on top and the "Click Here to run SQL reports".
2. Click on the button to run the SQL reports found in Chapter 4 of the report.



Welcome to our group's Web Application

Developed by Derik Callahan, Julie Davis, Kirsten Fure, Seth Quartey, and Kyle Williams

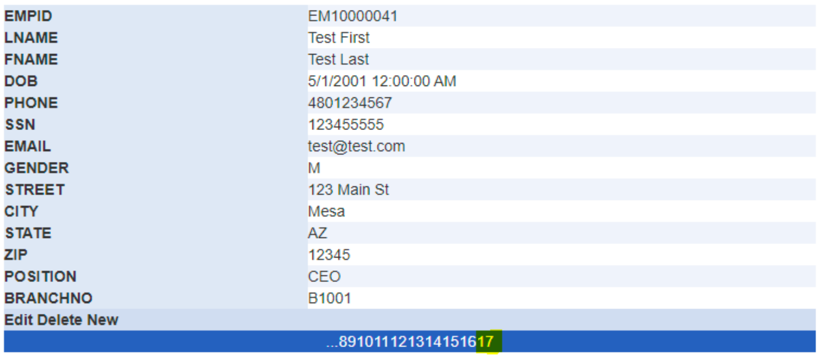
3. All reports load on the page. Please refer to Chapter 4 for a description of each query.
4. The Add a Client, Product Editor, Employee Add/Delete, Visit Log, and Create a Course tabs lead to pages where you can insert, edit, or delete tuples from their respective tables.
5. As an example, to add an employee:
   a. Click on the Employee Add/Delete tab.
   b. Click "New" on the bottom of the table.
   c. Fill the details in as shown below. You can leave the EMPID blank as the system will generate one for you automatically.



Employee Editor

EMPID	
LNAME	Test First
FNAME	Test Last
DOB	01-MAY-2001
PHONE	4801234567
SSN	123455555
EMAIL	test@test.com
GENDER	M
STREET	123 Main St
CITY	Mesa
STATE	AZ
ZIP	12345
POSITION	CEO
BRANCHNO	B1001

   d. Finally, to view your result, navigate to the latest entry in the Employee view, shown here.



EMPID	EM10000041
LNAME	Test First
FNAME	Test Last
DOB	5/1/2001 12:00:00 AM
PHONE	4801234567
SSN	123455555
EMAIL	test@test.com
GENDER	M
STREET	123 Main St
CITY	Mesa
STATE	AZ
ZIP	12345
POSITION	CEO
BRANCHNO	B1001
Edit Delete New	

...891011121314151617

Chapter 7: Implementation Plan

A company with the size and scope of TriannaCorp must spend significant time and investment not only on the logical and conceptual design of a database, but also on physical design and implementation. However, even though there is a rather large cost to implementing a company wide database system, a well-planned database will easily pay for itself over time. A well-designed database that is implemented properly will increase efficiency, reduce errors, and provide a competitive advantage over competitors with inferior systems. Studies have shown that 40-60% of database system failures occur from errors in analysis and design phases (Snodgrass, Jensen, Torp, Dyreson, & Currim, 2019).

In order to implement TriannaCorp's database in a robust and cost-efficient manner, it is recommended that the relational database management system (RDBMS) be Oracle Database deployed to the cloud with Amazon Relational Database Service (RDS). This RDBMS implementation aligns well with the company's current trend towards e-tailing and centralized IT systems. TriannaCorp could also take advantage of other products from AWS to continue the trend.

Oracle Database is a long-time leader in the enterprise RDBMS market and continuously receives high ratings in terms of performance and reliability (Adrian, Feinberg, & Cook, 2019). Using Oracle database will also allow TriannaCorp's database administrators the ability to use the exact code written in this report without the need for any modifications due to different syntaxes. Once the data is uploaded, the previous problems with heterogeneities in data and semantics will also be resolved.

Using AWS and Amazon RDS to deploy Oracle database will allow TriannaCorp to reduce operational expenses as they will not need to house the database on a physical server on premise. Deploying Oracle Database with Amazon RDS is also fairly easy with Amazon Quick Start. Additionally, using AWS allows for the database to be deployed in a highly secure environment with extremely high availability due to the use of multiple availability zones. If one zone goes down, the database will instantly be available on the other zone.

Timeline

The expected timeline for implementing the new RDBMS is expected to take between 1-3 months. This timeframe will mostly be broken down into three steps:

1. Procurement (1-4 weeks)
2. Training (2-4 weeks)
3. Deployment (1-3 weeks)

Procurement of software licenses along with setting up their accounts are the first step of the implementation. TriannaCorp will need to obtain an Oracle Enterprise Edition software license and set up their AWS accounts. This step will require management and technical personnel to meet with Oracle and AWS sales reps to make sure licensing and account details are set up and negotiated properly.

Training is the second major step in the implementation plan. Database administrators should receive training on Oracle Database and AWS cloud services if they are not already familiar with them. System Administrators should also receive AWS training if they are not already familiar with the systems. Oracle and AWS both provide classroom training options that should be negotiated into the licensing contracts if possible.

The third and final step of the plan is to deploy the Oracle Database software. Steps for this include setting up a virtual private cloud (VPC) on AWS, creating the appropriate subnets gateways, and security groups, followed by uploading the Oracle Database software into the VPC. These steps can be completed significantly faster than using a traditional on-premise server. An experienced System Administrator can deploy these AWS products in less than 10 minutes. Once the Oracle software is uploaded, the final step would be to run the database creation and population scripts that were already created.

Costs

As mentioned earlier, a properly designed and implemented database system for a company as large as TriannaCorp requires a significant investment. The major expenses of this project will come from software licensing and support, initial training, and ongoing cloud data storage and transfer costs. Figure C.1 below shows the breakdown of expenditures:

Figure C.1

TriannaCorp Database Modernization Costs		
Database Expenses	**Year 1 Cost**	**Recurring Costs**
Oracle DB License	$47,500.00	
Oracle DB Support	$10,450.00	$10,450.00
Amazon RDS(db.r5.large)	$2,204.00	$2,204.00

AWS Storage Costs(2tb)	$2,760.00	$2,760.00
AWS Transfer Costs(100GB/month)	$90.00	$90.00
Total Database Expenses	$61,624.00	$14,124.00
Training Expenses		
AWS Classroom Training	$1,000.00	
Oracle Training	$1,000.00	
Additional Employee hourly expenses $30/hr x 320 hours	$9,600.00	
Total Training Expenses	$11,600.00	
Total Expenses	$73,224.00	$14,124.00

The largest cost of this project occurs from the initial licensing of Oracle Database Enterprise Edition. However, this is only a one-time cost and subsequent renewal and support is only a fourth of the initial costs. The second largest expense is the initial training to get employees up to speed on Oracle DB and AWS. This expense assumes a $30/hour rate for 320 hours, or four employees training for two work weeks each. Once they are initially trained, additional ongoing training can be included in other already occurring training budgets.

The main ongoing expenses of this project are from the use of Amazon RDS and associated AWS expenses. This cost estimate assumes TriannaCorp will benefit the most from using the Amazon RDS db.r5.large product. This product gives TriannaCorp a reserved instance and plenty of computing power to run the database efficiently and securely, especially considering the memory needs of Oracle DB. Given the sensitive nature of TriannaCorp's data, having a reserved, rather than shared instance will also provide an additional layer of security. This cost can also be scaled up or down to another product if future database needs change. The AWS storage costs assume storage of 2tb of data and 100GB of monthly data transfer, which will mostly be from backups.

It is expected that once the database is operational, yearly savings from increases in efficiency and error reductions will exceed $20,000 per year. Given the reduction in old system costs and new savings, the new RDBMS will pay for itself in the next 3-5 years.

Conclusion

This database project case has provided the team with practical experience on how to go about designing and implementing a database for a nationwide company. This project has covered portions of each of the stages of the database lifecycle including requirements analysis, logical design, physical design, implementation, and monitoring & maintenance.

Many lessons were learned over the course of the project, with a few of them standing out as particularly important. One of the main lessons learned from this project was the reinforcement of how critical good database design is. It is extremely important to spend the extra time diving into the requirements analysis and forming a plan before attempting to create the database. A poorly designed database can end up being nearly useless to users. Another lesson learned from this project is the importance of maintaining consistency of data types across tables and columns. Small errors or differences in column data types across tables can cause large errors that prevent queries from executing properly. A third lesson learned was the importance of testing and quality assurance. It is important to run many test queries to make sure the database is operating as it should.

Overall, this project has provided the team with significant experience in database design and development and a lot has been learned on the subject.

References

Adrian, M., Feinberg, D., Cook, H. (November 2019). *Magic Quadrant for Operational Database Management Systems.* Retrieved from https://www.gartner.com/doc/reprints?id=1-1XT9MEFJ&ct=191125&st=sb

Oracle Database on AWS. December 2019. Retrieved from https://docs.aws.amazon.com/quickstart/latest/oracle-database/overview.html

Oracle Technology Global Price List. December 5, 2019. Retrieved from

https://www.oracle.com/assets/technology-price-list-070617.pdf

Snodgrass, R., Jensen, C., Torp, K., Dyerson, C. Currim, S., Currim, F. (2019). Conceptual Database Design: The Entity Relationship (ER) Model.