**MIS 531 GROUP PROJECT**

Shape

Description automatically generated

**EXPENSE MANAGEMENT TOOL**

**[TEAM S1G]: HEXAGON THEORY**

Mane, Tanvi Tanaji - (tanvimane)

Nanoti, Abhishek - (ananoti)

Ramaraj, Aishwarya Srevaikundam – (asramaraj)

Shantha Kumar, Nidhi - (nidhis)

Shishangia, Shivam Chandresh - (shivams)

Tripuraneni, Likitha - (ltripuraneni)

Contents

[**CHAPTER 1: TRIPACTIONS EXPENSE MANAGEMENT** 5](#_Toc121349872)

[**REQUIREMENT ANALYSIS** 5](#_Toc121349873)

[**Chapter 2: ER DIAGRAM AND DATA DICTIONARY** 7](#_Toc121349874)

[**DATA DICTIONARY (Conceptual / ER Modeling)** 7](#_Toc121349875)

[**CHAPTER 3: RELATIONAL SCHEMA AND DATA DICTIONARY** 18](#_Toc121349876)

[**ER TO RELATIONAL (TABLES BEFORE NORMALIZATION)** 18](#_Toc121349877)

[**NORMALIZATION AND FUNCTIONAL DEPENDENCIES:** 21](#_Toc121349878)

[**TABLES AFTER NORMALIZATION** 28](#_Toc121349879)

[**RELATIONAL DATA DICTIONARY:** 31](#_Toc121349880)

[**TABLE CREATION SCRIPTS FOR TRIPACTIONS DATABASE:** 39](#_Toc121349881)

[**CHAPTER 4: SQL QUERIES** 50](#_Toc121349882)

[**LIST OF OPEN AND IN PROGRESS COMPLAINTS** 50](#_Toc121349883)

[**TOP COMPLAINT TYPES AND THEIR COUNT** 50](#_Toc121349884)

[**LATE REIMBURSEMENTS** 51](#_Toc121349885)

[**DETAILS OF CARD ISSUANCE REQUEST** 52](#_Toc121349886)

[**EXPENDITURE MONITORING** 52](#_Toc121349887)

[**CARD STATISTICS- 1** 53](#_Toc121349888)

[**CARD STATISTICS- 2** 54](#_Toc121349889)

[**SUPPORT TEAM DETAILS** 55](#_Toc121349890)

[**APPROVED DEPARTMENT CLAIMS** 57](#_Toc121349891)

[**DISPLAY POLICY AND EMPLOYEE COUNT** 59](#_Toc121349892)

[**COMPLAINT RESOLUTION TIME** 60](#_Toc121349893)

[**LOCATION AND DESCRIPTION SPECIFIC EXPENDITURE SUMMARY** 61](#_Toc121349894)

[**Chapter 5: TRIGGERS, PROCEDURES AND ADVANCED SQL** 62](#_Toc121349895)

[**USER VALIDATIONS PACKAGE** 62](#_Toc121349896)

[**REIMBURSEMENT TRIGGER** 65](#_Toc121349897)

[**REIMBURSEMENT PROCEDURE** 66](#_Toc121349898)

[**DERIVE END DATE WITH TRIGGER** 66](#_Toc121349899)

[**EMPLOYEE ID SEQUENCE GENERATION TRIGGER** 67](#_Toc121349900)

[**DEPARTMENT ID SEQUENCE GENERATION TRIGGER** 67](#_Toc121349901)

[**COMPLAINT ID SEQUENCE GENERATION TRIGGER** 68](#_Toc121349902)

[**TRANSACTION ID SEQUENCE GENERATION TRIGGER** 68](#_Toc121349903)

[**CONTRACT ID SEQUENCE GENERATION TRIGGER** 69](#_Toc121349904)

[**LOGIN ID SEQUENCE GENERATION TRIGGER** 69](#_Toc121349905)

[**CHECK CARD EXPENSE LIMIT WITH TRIGGER** 70](#_Toc121349906)

[**CONSTRAINING RELATIONSHIP TRIGGER** 71](#_Toc121349907)

[**DEPARTMENT COUNT TRIGGER** 72](#_Toc121349908)

[**GENERATE TOTAL EXPENSE INCURRED** 72](#_Toc121349909)

[**MANAGER WILL ALSO MANAGE THEMSELF** 73](#_Toc121349910)

[**INDEXING CC\_EMPLOYEES WITH CLIENT COMPANY ID** 73](#_Toc121349911)

[**PARTITION CLIENT COMPANIES BY TYPE** 73](#_Toc121349912)

[**CHAPTER 6: FRONT-END** 74](#_Toc121349913)

[**HOME PAGE** 74](#_Toc121349914)

[**USER LOGIN PAGE** 74](#_Toc121349915)

[**SIGN UP PAGE** 75](#_Toc121349916)

[**MANAGER’S HOME PAGE VIEW** 75](#_Toc121349917)

[**EMPLOYEES VIEW FOR MANAGER** 75](#_Toc121349918)

[**NEW EMPLOYEE ADDITION IN EMPLOYEES VIEW** 76](#_Toc121349919)

[**VIEW EMPLOYEES IN EMPLOYEES VIEW** 76](#_Toc121349920)

[**REIMBURSEMENTS PAGE** 77](#_Toc121349921)

[**NEW REIMBURSEMENT FORM IN REIMBURSEMENTS** 77](#_Toc121349922)

[**REIMBURSEMENT CLAIMS ADDITION, UPDATION AND DELETION IN REIMBURSEMENTS** 78](#_Toc121349923)

[**REIMBURSEMENT CLAIMS APPROVAL AND REJECTION IN REIMBURSEMENTS** 79](#_Toc121349924)

[**EXPENSES** 79](#_Toc121349925)

[**ASSIGN CARDS** 80](#_Toc121349926)

[**COMPLAINTS** 80](#_Toc121349927)

[**NEW COMPLAINTS FORM** 81](#_Toc121349928)

[**COMPLAINTS MANAGEMENT IN COMPLAINTS** 81](#_Toc121349929)

[**TOP COMPLAINT TYPES AND COUNT** 82](#_Toc121349930)

[**SUPPORT TEAM INFORMATION** 82](#_Toc121349931)

[**REIMBURSEMENTS VIEW FOR EMPLOYEES** 83](#_Toc121349932)

[**CARD DETAILS INFORMATION PAGE FOR EMPLOYEES** 83](#_Toc121349933)

[**COMPLAINTS VIEW FOR EMPLOYEE’S LOGIN** 84](#_Toc121349934)

[**SUPPORT TEAM INFORMATION FOR EMPLOYEE’S LOGIN** 84](#_Toc121349935)

[UI REPRESENTATION OF COMPLEX QUERIES IN STATISTICS 85](#_Toc121349936)

[**CHAPTER 7: IMPLEMENTATION PLAN** 88](#_Toc121349937)

[**PLAN OF ACTION:** 88](#_Toc121349938)

[**EXPENSES ALONG WITH DESCRIPTION FOR THE IMPLEMENTATION OF FULL PROJECT** 88](#_Toc121349939)

[**APPENDIX A** 90](#_Toc121349940)

# **CHAPTER 1: TRIPACTIONS EXPENSE MANAGEMENT**

TripActions is looking for expanding their revenue stream by introducing TripActions Liquid – an expense management and corporate card solution for their platform. The current database design will not be able to hold these new features as it will add further complexity if added without careful consideration of the issues like redundant data and loss of valuable information. To accommodate this new module, redesigning of their database is required for the smooth integration to incorporate relevant user functionalities.

Onboarding the client companies and partnered banks, keeping a track of the contracts signed between Trip Actions and Client companies, providing support for any queries that the clients have and finally tracking the payments is extremely curial.

## **REQUIREMENT ANALYSIS**

TripActions (TA) is coming up with a tool by which a company can track their expenses including their employees’ corporate cards’ expenditures on a dashboard. Companies can partner with TripActions to purchase this expense management system to track their expenses.

TA employee details such as employee ID, employee first name, second name, email IDs, gender, address, date of joining, and phone numbers are stored. Some of the employees can either be Administrators who handle client contracts or Agents who handle customers’ complaints/requests by providing the solutions. Some of the Administrators are managers who manage the agents. The administrators who are not managers monitor the contracts signed with the client companies. Few TA employees are neither of them. TA employee’s login credentials such as login ID, username and password are maintained. A team of TA employees called Support team is formed constituting of two or more agents and one or more manager to provide service for every client company. Every support team has its unique ID and name.

TA maintains records of the client companies that are registered with them to use this expense management and corporate card solution. All the details regarding contract ID, duration, start, and end date of contract are recorded. Client company signs the contract for a certain duration of time and one client company can opt for only one policy based on the contract signed.

For the client companies (customers) that are partnered with TA, it stores the company’s name, type of company (e.g., Tech, Manufacturing, Banking, Retail, etc.), email IDs, and phone numbers. Client Companies has different branches, so we store the branch ID, location, and contact. Client companies have departments such as Sales, Finance, Board of Directors, HR, Marketing, Product, Customer Success, Operations, Engineering and many more. TA records its client companies’ departments’ details such as ID, count of employees, name and total expense incurred.

Client company stores its employee details specific to departments like employee ID, first name, last name, gender, address, email ID, phone number, date of birth and date of joining. Few of the employees are managers who manage the employees and keep track of the expense budget allocated to the teams. For every contract signed, TA stores the login details specific for employees such as login ID, login username and Password for their client companies. Client company employee’s login details is defined by roles assigned to them which also describes the functionality and the view of the expense management tool. Role details such as the role ID, description and name are stored. When client companies sign up for the TA, the client company’s employees get subscription to the TA corporate card named as “TripActions liquid” which is accomplished due to tie up with multiple banks offering credit cards. A subscription of the corporate cards can be given only to a few selected employees which can or cannot be renewed during the contract, storing start date, validity and end date. A card can be assigned to any of the employee only if they are part of the client company that has signed up for that particular policy plan by client company. One corporate card can be reassigned to another employee based on subscription validity. All the tie-up bank’s details like name of the bank and bank ID are maintained. The bank branch details are also stored such as location, branch manager’s details (name, email ID and contact number) and routing number are recorded.

There are various policies provided by TA. The policy ID, name, tier, description and pricing plan. Policy names offered by TA include Growth plan and Professional plan with varied benefits. Growth plan is for small companies which consists of less than 50 employees and is free of cost. For every additional employee added to the policy under Growth plan is charged extra $25. Professional plan is for larger companies having more than 100 employees and additional corporate deals are recorded. Policy can offer multiple benefits to their valuable customers. TA records all the benefits that they can offer and stores the benefit ID, details and coupon code.

Client companies can subscribe to multiple types of cards offered by TA based on the contract signed and issue them to their employees according to their requirements. Once the corporate cards are issued, TA wants to track the details of every card such as card number, card type, category, issue date, expiry date and CVV.

In order to track their client companies’ expenses, TA records all the card expenses generated on the Card that the employees possess. Details like the Transaction ID, Date, Time, Location, description, and Amount generated by the employees of the client companies while using the TA liquid corporate card are stored. The expenses incurred from other than corporate card is tracked in Reimbursement that has been requested by the client company employees and stores details such as ID, the date of expense, location, amount, mode of expense (like corporate card, personal, cash, online payment etc.), status and type of expense incurred.

Card repayment details associated with the card once the payment is due for it are tracked such as the repayment ID, amount to be paid and payment date. The client companies get their invoices and TA which in turn pays the bank. TA offers client companies to repay the total amount partially before due date and all details are stored in payment date and amount paid before the due date. Bank generates separate invoice for every client company and sends it to the TA manager who handles the bills. Invoice received by TA from bank includes invoice ID, total amount and date of invoice issued. Later TA manager generates the invoice for every client company by adding the commission/brokerage as per the subsidy by storing invoiceID, totalAmount, dueDate, dateOfIssue and status. Brokerage detail such as brokerage amount is added. The invoice sent to the client company by TA has details such as invoice ID, total amount and date of issued.

TA keeps the status information of every contract which can be either in active or cancelled status. If a client company is unhappy with TA services and decides to terminate the contract, TA keeps a note of the cancellation details of the contract by recording date of cancellation and reason associated with the cancellation. For Active status of contracts, TA stores creditScores of their client companies.

TA would also like to keep track of the complaints or support requests they receive from their client’s employees. For the record, they’d like to save the complaint ID, complaint type (e.g., sign in/login issues, data mismatch, etc.), status (pending or completed or open), filed date, resolved date and complaint description. TA have designated customer support agents to manage these complaints/support requests that they receive.

# **Chapter 2: ER DIAGRAM AND DATA DICTIONARY**

## **DATA DICTIONARY (Conceptual / ER Modeling)**

NOTE:Under each entity,

* Determines the attributes within that entity
* Subclass of a superclass entity
* Attribute under a subclass
* Attributes under composite attributes
* Subclass under a subclass

|  |  |  |
| --- | --- | --- |
| **Schema Construct** | **Construct Description** | **Other Information** |
| **BANKS** | Entity Class, to model bank information |  |
| * bankID | Identifying number of the bank | Identifying Attribute |
| * bName | Bank name | Attribute should not contain NULL values |
| **BANK BRANCHES** | Entity class that models the bank branches |  |
| * branchManagerDetails * emailID * name * phoneNum | Branch manager details that contain their name, email ID and phone number | Composite Attribute |
| * location | Address at which the bank branch is located |  |
| * routingNum | Identifying number of a bank branch | Identifying attribute |
| **BENEFITS** | Entity class, to model the benefits that are provided by the policies |  |
| * benefitDetails | All details about each benefit are enlisted. E.g: Offers or deals which provides some concession |  |
| * benefitID | Unique Identifier associated with a benefit | Identifying Attribute |
| * couponCode | A code that is used to avail the benefit | Attribute cannot have NULL values |
| **BROKERAGES** | Weak entity class, that models the brokerage amount details associated with TripActions’ invoices that are verified by TripActions’ Managers |  |
| * brokerageAmount | The amount (in $) owed by TripActions’ to make payments for the invoice. 7% of total amount per invoice is added to this attribute. | Attribute cannot be a negative value |
| **CARDS** | Entity class to model the cards that TripActions can offer to the client companies employees |  |
| * cardNum | Identifying number for cards | Identifying attribute |
| * cardType | Type of card. E.g – Mastercard, VISA, Amex | Attribute should not contain NULL values |
| * category | Category the card belongs to. E.g – Gold, Silver, Platinum, Black, etc. | Attribute should not contain NULL values |
| * cvv | Security code for the cards | Attribute should not contain NULL values |
| * eDate | Expiry date of the card | Attribute should not contain NULL values |
| * iDate | The date the card gets issued from the bank to the client company |  |
| **CARD EXPENSES** | Entity class to model the expenses recorded by corporate cards registered for client companies employees |  |
| * amount | Amount (in $) associated with the expense | Attribute should not be a negative value |
| * expenseDate | Date on which the transaction was made |  |
| * description | Contains the details associated with the expense |  |
| * location | Location at which the transaction was performed |  |
| * expenseTime | Time at which the transaction was done |  |
| * transactionID | Identifying number associated with the expense | Identifying attribute |
| **CARD SUBSCRIPTIONS** | Weak entity class to model the card subscription details when an employee enrolls for a card |  |
| * endDate | The end date of the subscription of the card to a particular employee | Attribute should not contain NULL values |
| * startDate | Identifying attribute to save the date when the card got issued to the employee | Partially Identifying attribute |
| * validity | The duration of the subscription of the card that is availed to the employee in Days | Derived attribute |
| **CC BRANCHES** | Entity class that models the client companies’ branches information |  |
| * branchID | Identifying number of the branch of client company | Identifying attribute |
| * contactNumber | Contact number of the client company |  |
| * location | Location at which the client company is located |  |
| **CC DEPARTMENTS** | Entity class to model the departments within the client companies |  |
| * deptID | Identifying number of the department within the client company | Identifying attribute |
| * deptName | Name of the department in the client company |  |
| * empCount | Number of employees of the client company present in a specific department |  |
| * totalExpenseIncurred | Total Expense incurred by the department quarterly ($) | Attribute should not contain negative values |
| **CC EMP LOGINS** | Entity class to model the login details for those employees in the client companies to access the expense management tool by TripActions |  |
| * loginID | Identifying attribute for client company’s employees to login | Identifying attribute |
| * password | Password used to login to the expense management tool used by the client company’s employees | Attribute should not contain NULL values |
| * userName | Username used by the employee when they need to login. | Attribute should be UNIQUE and NOT NULL |
| **CC EMPLOYEES** | Entity class that models employees that belong to the client company | Superclass |
| * address | The address of the employee who belongs to the client company |  |
| * dateOfBirth | Date of Birth of the employee |  |
| * dateOfJoining | Date of joining of the employee into the client company | Attribute should not contain NULL values |
| * emailID | Email address(es) of the employee | Multivalued attribute |
| * empID | Identifying number of the employee from client company | Identifying Attribute |
| * gender | Gender of the employee working in the client company |  |
| * fName | First name of the employee who belongs to the client company |  |
| * lName | Last name of the employee who belongs to the client company |  |
| * phoneNum | Phone number(s) of the employee | Multivalued attribute |
| **CC MANAGERS** | Entity class that models the managers information who manages the employees of client companies | Subclass |
| * assignedBudget | Budget that is assigned to the manager for the expenses of the employees in his team (in $) | Attribute cannot contain negative values |
| **CC INVOICES** | Entity class to model the invoice data generated by TripActions that is circulated to client companies |  |
| * dateOfIssue | The date at which the invoice is generated | Attribute cannot be a NULL value |
| * invoiceID | Identifying number generated when an invoice is created | Identifying attribute |
| * totalAmount | Total amount (in $) that is recorded in the invoice based on the expenses | Attribute should not contain NULL values and Not negative |
| * dueDate | Date by which the amount has to be paid by client companies | Attribute should not contain NULL values |
| * status | Status of the invoice bill if it has been repaid or not till date | By default status is set to ‘Pending’ |
| **CLIENT COMPANIES** | Entity class, to model client companies’ details that have a contract with TripActions |  |
| * name | Client company name that is associated with TripActions | Attribute should not be null |
| * companyID | Identifying number for client company | Identifying attribute |
| * emailID | EmailID(s) of the client company | Multivalued Attribute |
| * phoneNum | Phone number(s) of the client company | Multivalued Attribute |
| * type | The type of industry the client company belongs to. Eg., Tech, Manufacturing, Banking, Retail, etc. |  |
| **COMPLAINTS** | Entity class, to model the complaints received by TripActions by their clients |  |
| * complaintDescription | Description of the complaint |  |
| * complaintID | Identifying number for the complaints | Identifying attribute |
| * status | Status of the complaint received, e.g, open, resolved, in progress, etc | Attribute should not contain NULL values.  Default value set to ‘Open’ |
| * type | The type of complaint received. Eg, sign in/login issues, data mismatch, etc. |  |
| * fileDate | The date on which the complaint was filed | Atrribute should not contain NULL values |
| * resolveDate | The date at which the complaint was resolved |  |
| **CONTRACTS** | Entity class to model the contract information between TripActions and its Client Companies |  |
| * contractID | Identifying number of the contract | Identifying attribute |
| * duration | The duration (in months) for which the contract exists between the client company and TripActions | By default it is 12 months |
| * endDate | The date on which contract ends | Derived Attribute and can not be null |
| * startDate | The contract start date | Start date can not be null |
| * **ACTIVE** | Subclass entity to model the active contracts that exists between TripActions and its client companies |  |
| * creditScore | This attribute tracks the client company’s financial payment trends and assigns a score accordingly (to verify if TripActions wants to continue being in contract with them) | Attribute should not contain NULL values |
| * **CANCELLED** | Subclass entity to model the cancelled/expired contracts that would have existed between TripActions and their client companies |  |
| * dateOfCancellation | Date at which the contract was cancelled/expired | Attribute should not contain NULL values |
| * reasonForCancellation | A brief description as to why the contract got cancelled or contracts which got expired and not renewed are tracked by TripActions |  |
| * **POLICIES** | Entity class to model the policies provided by TripActions for their customers | Superclass |
| * policyDescription | Brief details about the policy |  |
| * policyID | Identifying attribute for policy | Identifying Attribute |
| * policyName | Name of the policy |  |
| * policyTier | Policies are ranked according to what they offer (Eg: Emerald, Gold) |  |
| * pricingPlan | Pricing that the customer needs to plan if they sign up for the policy |  |
| * **GROWTH PLANS** | Entity class to model the growth plan offered by TripActions’ policy to small companies with ( < 50 employees ) | Subclass |
| * additionalPaymentPerNewUser | Contains the amount paid for every new user that is added under the growth plan (in $) | Attribute should not contain negative values |
| * **PROFESSIONAL PLANS** | Entity class that models the plan that is provided by TripActions for large companies ( >50 employees) | Subclass |
| * additionalCorporateDeals | Deals description that is altered and provided to each large company TripActions is associated with |  |
| **REIMBURSEMENTS** | Entity class to model reimbursements (employees are reimbursed if they use their private payment methods and not the corporate card) of the employees of client companies |  |
| * amount | Payment amount in $ towards reimbursement | Attribute should not contain a 0 or negative values |
| * expenseDate | The date at which the expense was made that had been requested for reimbursement | Attribute should not contain NULL values |
| * location | Location of payment |  |
| * modeOfExpense | It contains details about how the payment was exactly made (personal card/ cash/ zelle) |  |
| * reimbursementID | It is the unique auto generated number produced by the TripActions software for the reimbursement | Identifying attribute |
| * status | Attribute that tracks the reimbursement status. | Attribute should not contain NULL values.  This will have a by default value of “pending”. |
| * typeOfExpenseIncurred | The type of expense incurred. E.g, - Travel, food, etc. |  |
| **REPAYMENTS** | Entity class that models all the payments towards the invoices the client companies must pay to TripActions. This payment amount is the total expense amount that is generated towards the company by corporate cards |  |
| * amountPaid | Amount that is due (in $) for the payments | Attribute cannot contain negative values |
| * paymentDate | The date on which the repayment was made |  |
| * repaymentID | Unique number associated with repayment | Identifying attribute |
| **ROLES** | Entity class to model the roles associated to the employees in the client companies accessing the expense management tool that will have different views for each role |  |
| * description | Description about the role and what privileges it has being a TripActions user. |  |
| * roleID | Identifying number for the role | Identifying attribute |
| * name | The name of the role (e.g, manager, department employees) | Attribute should be UNIQUE and not contain NULL values |
| **SUPPORT TEAMS** | Entity class to model the data of the Support Team of Trip Actions that is formed with TA Managers and TA Customer support agents |  |
| * supportTeamID | Identifying attribute for a support team |  |
| * name | Name of the support team | Attribute should not contain NULL values |
| **TA EMPLOYEES** | Entity class to model all the employees working for TripActions | Superclass |
| * address | Address of the TripActions employee |  |
| * gender | Gender of employee |  |
| * fName | First name of the employee |  |
| * lName | Last name of the employee |  |
| * dateOfJoining | The date at which the employee joined TripActions | Attribute should not contain NULL values |
| * emailID | Email ID(s) of the employee | Multivalued attribute |
| * empID | Identifying number for employee | Identifying Attribute |
| * phoneNum | Contact number(s) of employee in TripActions | Multivalued attribute |
| * **TA ADMINISTRATORS** | Entity subclass that is linked to TripActions employee’s entity class to model the data of employees that work as administrators | Subclass |
| * **TA MANAGERS** | Entity subclass to model those managers who manage the invoices incurred by their client companies and also manage the TripActions Customer Support agents | Subclass within a subclass |
| * **TA CUSTOMER SUPPORT AGENTS** | Entity subclass that is linked to TripActions employees entity class to model the customer support agents that work for TripActions and handle the complaints they receive | Subclass |
| **TA INVOICES** | Entity class to model the invoice that is generated by the bank and issued to TripActions |  |
| * dateOfIssue | Date at which the invoice was issued |  |
| * invoiceID | Identifying number for the invoice that has been issued by the bank to TripActions | Identifying attribute |
| * totalAmount | Amount (in $) to be paid that is included in the invoice | Attribute should not have negative values |
| **TA LOGINS** | Entity class to model login details for TripActions employees |  |
| * loginID | Identifying attribute for TripActions’ employee login. | Identifying attribute |
| * password | password used by TripActions employee for logging into the TripActions portal | Attribute cannot be NULL |
| * userName | Username associated with the TripActions employee when they login. | Attribute should be UNIQUE and NOT NULL |

|  |  |
| --- | --- |
| Administers | Relationship that models a relation between TA ADMINISTRATORS to CONTRACTS that are signed by Client Companies |
| Are Part Of | Relationship that models a relation between cards and contracts. Cards are a part of contracts |
| Belongs To | Relationship that models a relation between banks entity and bank branches entity. Bank branches belong to banks |
| Binded By | Relationship that models a relation between policies and contracts. A contract is binded by policies present in it |
| Can Apply For | Relationship that models a relation between cc employees entity and reimbursements entity. Employees can apply for reimbursements |
| Circulate | Relationship that models the relation of client companies invoices being circulated by Trip Action managers |
| Comprises Of | Relationship that models relation between cc branches and client companies. Client companies comprise of a number of client company branches (cc branches) |
| Consist Of | Relationship that models relation between client companies entity and it's departments - cc departments entity. Client companies have a number of departments |
| Consists | Relationship that models a relation between client company departments - cc departments entity and its employees - cc employees entity. Departments consist of employees |
| Define | Relationship that models a relation between roles entity and cc emp logins. The kind of login is defined by the role assigned |
| Enroll For | Relationship that models a constrained relation between cards entity, card subscriptions entity and client companies entity. |
| Forms | Relationship that maps the aggregation relation of a support team forming by combining TA customer support agents, and managers |
| Generate | Relationship that models a relation between card expenses entity and cards entity. Expenses are generated on the card. |
| Generates | Relationship that models a relation between TA invoice entity and banks entity. Banks generate invoices. |
| Handled By | Relationship that maps a relation between cc invoice entity, client companies and repayment entities. Clients companies will handle the cc invoice that is given to them and it is recorded in repayment |
| Have | Relationship that models the TA Employees have TA logins to login to the expense management portal |
| Have Access To | Relationship that models a relation between cc emp logins and cc employees. Employees have access to their employee logins |
| Manages | Relationship that models the relation of Trip Actions managers managing their customer support agents |
| CC\_Manages | Relationship that models a relation between cc employees entity and cc managers. Managers mange the employees |
| Offers | Relationship that models a relation between policies entity and benefits entity. Policies offer one or multiple benefits |
| Provide Service To | Relationship that maps a relation between the support team entity and client company entity. Support team provides service to a client company |
| Raise | Relationship that models a relation between client company employees entity and complaints entity. The employees of client company can raise complaints |
| Resolves | Relationship that models the relation where Trip Actions customer support agents resolve the complaints filed to them |
| Sign | Relationship that models a relation between client company and a contract being signed up |
| Verify | Relationship that maps a relation between TA managers, TA invoices and brokerage. TA managers verify if the brokerage is added to be TA invoices |
| Work For | Relationship that maps a relationship between cc employees entity and client companies entity. Cc employees work for their client company |

ER Diagram Link- [TripActionsER](https://emailarizona-my.sharepoint.com/:u:/g/personal/nidhis_arizona_edu/ESX1P0o-lyFKrNEiH8wQpcwB9zpufS-KChmPhEnKbGsqvg)



# **CHAPTER 3: RELATIONAL SCHEMA AND DATA DICTIONARY**

## **ER TO RELATIONAL (TABLES BEFORE NORMALIZATION)**

Strong Entities:

|  |
| --- |
| * BANKS (bankID, bName) |
| * BANK\_BRANCHES (routingNum, phoneNum, location, emailID, name, bankID)   Foreign Key (bankID) references BANKS (bankID) |
| * BENEFITS (benefitID, benefitDetails, couponCode) |
| * CARDS (cardNum, cardType, iDate, cvv, category, eDate, bankID, contractID)   Foreign Key (bankID) references BANKS (bankID)  Foreign Key (contractID) references CONTRACTS (contractID) |
| * CARD\_EXPENSES (transactionID, expenseDate, expenseTime, location, description, amount, cardNum)  Foreign Key (cardNum) references CARDS (cardNum) |
| * CC\_BRANCHES (branchID, location, contactNumber, companyID)  Foreign key (companyID) references CLIENT\_COMPANIES (companyID) |
| * CC\_DEPARTMENTS (deptID, empCount, totalExpenseIncurred, deptName, companyID)   Foreign key (companyID) references CLIENT\_COMPANIES (companyID) |
| * CC\_EMP\_LOGINS (loginID, password, userName) |
| * CC\_EMPLOYEES (empID, fName, lName, address, dateOfJoining, dateOfBirth, gender, companyID, loginID)   Foreign Key (companyID) references CLIENT\_COMPANIES (companyID)  Foreign Key (loginID) references CC\_EMP\_LOGINS (loginID)   CC\_MANAGERS (managerID, assignedBudget)  Foreign Key (managerID) references CC\_EMPLOYEES (empID) |
| * CC\_EMPLOYEES\_EMAILS (empID, emailID)   Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * CC\_EMPLOYEES\_PHONES (empID, phoneNum)   Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * CC\_INVOICES (invoiceID, totalAmount, dateOfIssue, dueDate, status, companyID, managerID)  Foreign key (companyID) references CLIENT\_COMPANIES (companyID)  Foreign key (managerID) references TA\_MANAGERS (managerID) |
| * CLIENT\_COMPANIES (companyID, name, type) |
| * CLIENT\_COMPANIES\_EMAILID (companyID, emailID) * Foreign Key (companyID) references CLIENT\_COMPANIES (companyID) |
| * CLIENT\_COMPANIES\_PHONENUM (companyID, phoneNum)   Foreign Key (companyID) REFERENCES CLIENT\_COMPANIES (companyID) |
| * COMPLAINTS (complaintID, type, filedDate, resolvedDate, status, complaintDescription, empID)   Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * CONTRACTS (contractID, startDate, duration, endDate, policyID, companyID)  Foreign Key (policyID) references POLICIES (policyID)  Foreign Key (companyID) references CLIENT\_COMPANIES (companyID)   ACTIVE (contractID, creditScore)  Foreign Key (contractID) references CONTRACTS (contractID)  CANCELLED (contractID, dateOfCancellation, reasonForCancellation)  Foreign Key (contractID) references CONTRACTS (contractID) |
| * POLICIES (policyID, policyName, policyDescription, policyTier, pricingPlan)   GROWTH\_PLANS (policyID, additionalPaymentPerNewUser)  Foreign Key (policyID) references POLICIES (policyID)  PROFESSIONAL\_PLANS (policyID, additionalCorporateDeals)  Foreign Key (policyID) references POLICIES (policyID) |
| * REIMBURSEMENTS (reimbursementID, expenseDate, location, amount, modeOfExpense, typeOfExpenseInccured, status, empID)   Foreign Key(empID) references CC\_EMPLOYEES (empID) |
| * ROLES (roleID, name, description) |
| * SUPPORT\_TEAMS (supportTeamID, name, companyID)   Foreign key (companyID) references CLIENT\_COMPANIES (companyID) |
| * TA\_EMPLOYEES (empID, fName, lName, dateOfJoining, gender, address, loginID)  Foreign Key (loginID) REFERENCES TA\_LOGINS (loginID)   TA\_ADMINISTRATORS (empID)  Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID)  TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)  Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID)    TA\_MANAGERS (empID)  Foreign Key (empID) REFERENCES TA\_ADMINISTRATORS (empID) |
| * TA\_EMPLOYEES\_EMAILID (empID, emailID)  Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID) |
| * TA\_EMPLOYEES\_PHONE\_NUM (empID, phoneNum)  Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID) |
| * TA\_INVOICES(invoiceID, totalAmount, dateOfIssue, bankID)   Foreign Key (bankID) references BANKS (bankID) |
| * TA\_LOGINS (loginID, userName, password) |

Relationships and Weak Entities:

|  |
| --- |
| * ADMINISTERS (empID, contractID)  Foreign Key (contractID) references CONTRACTS (contractID)  Foreign Key (empID) references TA\_ADMINISTRATORS (empID) |
| * BROKERAGES (invoiceID, empID, brokerageAmount)   Foreign Key (invoiceID) references TA\_INVOICES (invoiceID)  Foreign Key (empID) references TA\_MANAGERS (empID) |
| * CARD\_SUBSCRIPTION (empID, cardNum, startDate, endDate, validity)   Foreign Key (empID) references CC\_EMPLOYEES (empID)  Foreign Key (cardNum) references CARDS (cardNum) |
| * CC\_MANAGES (managerID, employeeID)   Foreign Key (managerID) references CC\_MANAGERS (managerID)  Foreign Key (employeeID) references CC\_EMPLOYEES (empID) |
| * CONSISTS (deptID, empID)   Foreign Key (deptID) references CC\_DEPARTMENTS (deptID)  Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * DEFINE (roleID, loginID)   Foreign Key (roleID) references ROLES (roleID)  Foreign Key (loginID) references CC\_EMP\_LOGINS (loginID) |
| * FORMS\_MANAGERS (supportTeamID, empID)  Foreign Key (supportTeamID) references SUPPORT\_TEAMS (supportTeamID)  Foreign Key (empID) references TA\_MANAGERS (empID) |
| * FORMS\_SUPPORT\_AGENTS (supportTeamID, empID)  Foreign Key (supportTeamID) references SUPPORT\_TEAMS (supportTeamID)  Foreign Key (empID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID) |
| * MANAGES (managerID, supportAgentID)  Foreign Key (managerID) references TA\_MANAGERS (empID)  Foreign Key (supportAgentID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID) |
| * OFFERS (policyID, benefitID)  Foreign Key (policyID) references POLICIES (policyID)  Foreign Key (benefitID) references BENEFITS (benefitID) |
| * REPAYMENTS (repaymentID, invoiceID, companyID, amountPaid, paymentDate)   Foreign key (invoiceID) references CC\_INVOICES (invoiceID)  Foreign key (companyID) references CLIENT\_COMPANIES (companyID) |
| * RESOLVES (empID, complaintID)  Foreign Key (empID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)  Foreign Key (complaintID) references COMPLAINTS (complaintID) |
| * ROLE\_LOGIN\_DEFINITION (roleID, loginID)   Foreign Key (roleID) references ROLES (roleID)  Foreign Key (loginID) references CC\_EMP\_LOGINS (loginID) |

## **NORMALIZATION AND FUNCTIONAL DEPENDENCIES:**

**BANKS**

FD1: bankID -> bname

Original Table:

BANKS (bankID, bName)

Normalized Table:

BANKS (bankID, bName)

**BANK\_BRANCHES**

FD1: routingNum -> phoneNum, location, emailID, name

FD2: routingNum -> bankID

Original Table:

BANK\_BRANCHES (routingNum, phoneNum, location, emailID, name, bankID)

Normalized table:

BANK\_BRANCHES (routingNum, phoneNum, location, emailID, name, bankID)

Foreign Key (bankID) references BANKS (bankID)

**BENEFITS:**

FD1: BenefitID -> benefitDetails, couponCode

Original Table:

BENEFITS (benefitID, benefitDetails, couponCode)

Normalized Table:

BENEFITS (benefitID, benefitDetails, couponCode)

**CARDS**

FD1: cardNum -> cardType, iDate, cvv, category, eDate

FD2: cardNum -> bankID, contractID

Original Table:

CARDS (cardNum, cardType, iDate, cvv, category, eDate, bankID, contractID)

Normalized Table:

CARDS (cardNum, cardType, iDate, cvv, category, eDate, bankID, contractID)

Foreign Key (bankID) references BANKS (bankID)  
 Foreign Key (contractID) references CONTRACTS (contractID)

**CARD\_EXPENSES**

FD1: transactionID -> expenseDate, expenseTime, location, description, amount

FD2: transactionID -> cardNum

Original Table:

CARD\_EXPENSES (transactionID, expenseDate, expenseTime, location, description, amount, cardNum)

Normalized Table:

CARD\_EXPENSES (transactionID, expenseDate, expenseTime, location, description, amount, cardNum)  
 Foreign Key (cardNum) references CARDS (cardNum)

**CC\_BRANCHES**

FD1: branchID -> location, contatNumber, companyID

Original Table:

CC\_BRANCHES (branchID, location, contactNumber, companyID)

Normalized Table:

CC\_BRANCHES (branchID, location, contactNumber, companyID)  
 Foreign key companyID references CLIENT\_COMPANIES (companyID)

**CC\_DEPARTMENTS**

FD1: deptID –> empCount, totalExpenseIncurred, deptName, companyID

Original Table:

CC\_DEPARTMENTS (deptID, empCount, totalExpenseIncurred, deptName, companyID)

Normalized Table:

CC\_DEPARTMENTS (deptID, empCount, totalExpenseIncurred, deptName, companyID)

Foreign key (companyID) references CLIENT\_COMPANIES (companyID)

**CC\_EMP\_LOGINS**

FD1: loginID -> userName, password

FD2: userName -> password

Original Table:

CC\_EMP\_LOGINS (loginID, password, userName)

Normalized Table:

CC\_EMP\_LOGINS\_USERNAME (loginID, userName)

CC\_EMP\_LOGINS\_PASSWORD (userName, password)

Foreign Key (userName) REFERENCES CC\_EMP\_LOGINS\_USERNAME (userName)

**CC\_EMPLOYEES**

FD1: empID -> fName, lName, address, dateOfJoining, dateOfBirth, gender, companyID, loginID

FD2: emailID -> empID

FD3: phoneNum -> empID

ORIGINAL TABLE:

CC\_EMPLOYEES (empID, fName, lName, address, dateOfJoining, dateOfBirth, gender, companyID emailID, phoneNum, loginID)

NORMALIZED TABLE:

CC\_EMPLOYEES (empID, fName, lName, address, dateOfJoining, dateOfBirth, gender, companyID, loginID)

Foreign Key (companyID) references CLIENT\_COMPANIES (companyID)

Foreign Key (loginID) references CC\_EMP\_LOGINS\_USERNAME (loginID)

CC\_EMPLOYEES\_EMAILS (empID, emailID)

Foreign Key (empID) references CC\_EMPLOYEES (empID)

CC\_EMPLOYEES\_PHONES (empID, phoneNum)

Foreign Key (empID) references CC\_EMPLOYEES (empID)

**CC\_MANAGERS**

FD1: managerID -> assignedBudget

ORIGINAL TABLE:

CC\_MANAGERS (managerID, assignedBudget)

Foreign Key (managerID) references CC\_EMPLOYEES (empID)

NORMALIZED TABLE:

CC\_MANAGERS (managerID, assignedBudget)

Foreign Key (managerID) references CC\_EMPLOYEES (empID)

**CC\_INVOICES**

FD1: invoiceID -> totalAmount, dateOfIssue, status, dueDate, companyID, managerID

Original Table:

CC\_INVOICES (invoiceID, totalAmount, dateOfIssue, status, dueDate, companyID, managerID)

Normalized Table:

CC\_INVOICES (invoiceID, totalAmount, dateOfIssue, status, dueDate, companyID, managerID)  
 Foreign key (companyID) references CLIENT\_COMPANIES (companyID)  
 Foreign key (managerID) references TA\_MANAGERS (managerID)

**CLIENT\_COMPANIES**

FD1: companyID -> name, type

FD2: emailID -> companyID

FD3: phoneNum -> companyID

ORIGINAL TABLE:

CLIENT\_COMPANIES (companyID, name, type, emailID, phoneNum)

NORMALIZED TABLE:

CLIENT\_COMPANIES (companyID, name, type)

CLIENT\_COMPANIES\_EMAILID (companyID, emailID)

Foreign Key (companyID) REFERENCES CLIENT\_COMPANIES (companyID)

CLIENT\_COMPANIES\_PHONENUM (companyID, phoneNum)

Foreign Key (companyID) REFERENCES CLIENT\_COMPANIES (companyID)

**COMPLAINTS**

FD1: complaintID -> type, filedDate, resolvedDate, status, complaintDescription, empID

Original Table:

COMPLAINTS (complaintID, type, filedDate, resolvedDate, status, complaintDescription, empID)

Normalized Table:

COMPLAINTS (complaintID, type, filedDate, resolvedDate, status, complaintDescription, empID)

Foreign key (empID) references CC\_EMPLOYEES (empID)

**CONTRACTS:**

FD1: contractID -> startDate, duration, endDate, policyID, companyID

Original Table:

CONTRACTS (contractID, startDate, duration, endDate, policyID, companyID)

Normalized Table:

CONTRACTS (contractID, startDate, duration, endDate, policyID, companyID)  
 Foreign Key (policyID) references POLICIES (policyID)  
 Foreign Key (companyID) references CLIENT\_COMPANIES (companyID)

**ACTIVE:**

FD1: contractID -> creditScore

Original Table:

ACTIVE (contractID, creditScore)

Normalized Table:

ACTIVE (contractID, creditScore)  
 Foreign Key (contractID) references CONTRACTS (contractID)

**CANCELLED**

FD1: contractID -> dateOfCancellation, reasonForCancellation

Original Table:  
 CANCELLED (contractID, dateOfCancellation, reasonForCancellation)   
Normalized Table:

CANCELLED (contractID, dateOfCancellation, reasonForCancellation)  
 Foreign Key (contractID) references CONTRACTS (contractID)

**POLICIES:**

FD1: policyID -> policyName, PolicyDescription, policyTier, pricingPlan

Original Table:

POLICIES (policyID, policyName, policyDescription, policyTier, pricingPlan)

Normalized Table:

POLICIES (policyID, policyName, policyDescription, policyTier, pricingPlan)

**GROWTH\_PLANS:**

FD1: policyID -> additionalPaymentPerNewUser

Original Table:  
 GROWTH\_PLANS (policyID, additionalPaymentPerNewUser)

Normalized Table:

GROWTH\_PLANS (policyID, additionalPaymentPerNewUser)  
 Foreign Key (policyID) references POLICIES (policyID)

**PROFESSIONAL PLANS:**

FD1: policyID -> additionalCorporateDeals

Original Table:

PROFESSIONAL\_PLANS (policyID, additionalCorporateDeals)

Normalized Table:

PROFESSIONAL\_PLANS (policyID, additionalCorporateDeals)  
 Foreign Key (policyID) references POLICIES (policyID)

**REIMBURSEMENTS:**

FD1: reimbursementID -> invoice, location, amount, modeOfExpense,typeOfExpenseIncurred

FD2: reimbursementID -> empID

Original Table:

REIMBURSEMENTS (reimbursementID, expenseDate, location, amount, modeOfExpense, typeOfExpenseInccured, status, empID)

Normalized Table:

REIMBURSEMENTS (reimbursementID, expenseDate, location, amount, modeOfExpense, typeOfExpenseInccured, status, empID)

Foreign Key(empID) references CC\_EMPLOYEES (empID)

**ROLES:**

FD1: roleID -> name, description

Original Table:

ROLES (roleID, name, description)

Normalized Table:

ROLES (roleID, name, description)

**SUPPORT\_TEAMS:**

FD1: supportTeamID -> name

FD2: supportTeamID -> companyID

Original Table:

SUPPORT\_TEAMS (supportTeamID, name, companyID)

Normalized Table:

SUPPORT\_TEAMS (supportTeamID, name, companyID)

Foreign key (companyID) references CLIENT\_COMPANIES (companyID)

**TA\_EMPLOYEES**

FD1: empID -> gender, fName, lName, address, dateOfJoining,loginID

FD2: emailID -> empID

FD3: phoneNum -> empID

Original Table:

TA\_EMPLOYEES (empID, gender, fName, lName, address, dateOfJoining, emailID, phoneNum,loginID)

Normalized Table:

TA\_EMPLOYEES (empID, fName, lName, dateOfJoining, gender, address, loginID)  
 Foreign Key (loginID) REFERENCES TA\_LOGINS (loginID)  
 TA\_EMPLOYEES\_EMAILID (empID, emailID)  
 Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID)

TA\_EMPLOYEES\_PHONE\_NUM (empID, phoneNum)  
 Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID)

**TA\_ADMINISTRATORS**

Original Table:  
 TA\_ADMINISTRATORS (empID)

Normal Table:  
 TA\_ADMINISTRATORS (empID)  
 Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID)

**TA\_CUSTOMER\_SUPPORT\_AGENTS**

Original Table:

TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)

Normalized Table:

TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)  
 Foreign Key (empID) REFERENCES TA\_EMPLOYEES (empID)

**TA\_MANAGERS**

Original Table:  
 TA\_MANAGERS (empID)

Normalized Table:  
 TA\_MANAGERS (empID)  
 Foreign Key (empID) REFERENCES TA\_ADMINISTRATORS (empID)

**TA\_INVOICES**

FD1: invoiceID -> totalAmount, dateOfIssue

invoiceID -> bankID

Original Table:

TA\_INVOICES(invoiceID, totalAmount, dateOfIssue, bankID)

Normalized Table:

TA\_INVOICES(invoiceID, totalAmount, dateOfIssue, bankID)

Foreign Key (bankID) references BANKS (bankID)

**TA\_LOGINS**

FD1: loginID -> userName, password

userName -> password

Original Table:

TA\_LOGINS (loginID, userName, password)

Normalized Table:

TA\_LOGINS\_USERNAME (loginID, userName)

TA\_LOGINS\_PASSWORD (userName, password)

Foreign Key (userName) references TA\_LOGINS\_USERNAME (userName)

**Weak Entities and Relationships**

**ADMINISTERS**

FD1: contractID, empID -> contractID, empID

Original Table:

ADMINISTERS (empID, contractID)

Normalized Table:

ADMINISTERS (empID, contractID)  
 Foreign Key (contractID) references CONTRACTS (contractID)  
 Foreign Key (empID) references TA\_ADMINISTRATORS (empID)

**BROKERAGES**

FD1: invoiceID, empID -> brokerageAmount

Original Table:

BROKERAGES (invoiceID, empID, brokerageAmount)

Normalized Table:

BROKERAGES (invoiceID, empID, brokerageAmount)

Foreign Key (invoiceID) references TA\_INVOICES (invoiceID)

Foreign Key (empID) references TA\_MANAGERS (empID)

**CARD\_SUBSCRIPTIONS**

FD1: empID,cardNum, startDate -> endDate, validity

Original Table:

CARD\_SUBSCRIPTIONS (empID, cardNum, startDate, endDate, validity)

Normalized Table:

CARD\_SUBSCRIPTIONS (empID, cardNum, startDate, endDate, validity)

Foreign Key (empID) references CC\_EMPLOYEES (empID)

Foreign Key (cardNum) references CARDS (cardNum)

**CC\_MANAGES**

FD1: employeeID, managerID -> employeeID, managerID

Original Table:

CC\_MANAGES (managerID, employeeID)

Normalized Table:

CC\_MANAGES (managerID, employeeID)

Foreign Key (managerID) references CC\_MANAGERS (managerID)

Foreign Key (employeeID) REFERENCES CC\_EMPLOYEES (empID)

**CONSISTS**

FD1: deptID, empID -> deptID, empID

Original Table:

CONSISTS (deptID, empID)

Normalized Table:

CONSISTS (deptID, empID)

Foreign Key (deptID) references CC\_DEPARTMENTS (deptID)

Foreign Key (empID) references CC\_EMPLOYEES (empID)

**DEFINE**

FD1: roleID, loginID -> roleID, loginID

Original Table:

DEFINE (roleID, loginID­)

Normalized Table:

DEFINE (roleID, loginID)

Foreign Key (roleID) references ROLES (roleID)

Foreign Key (loginID) references CC\_EMP\_LOGINS\_USERNAME (loginID)

**FORMS\_MANAGERS**

FD1: supportTeamID, empID -> supportTeamID, empID

Original Table:

FORMS\_MANAGERS (supportTeamID, empID)

Normalized Table:

FORMS\_MANAGERS (supportTeamID, empID)  
 Foreign Key (supportTeamID) REFERENCES SUPPORT\_TEAMS (supportTeamID)  
 Foreign Key (empID) REFERENCES TA\_MANAGERS (empID)

**FORMS\_SUPPORT\_AGENTS**

FD1: empID, supportTeamID -> empID, supportTeamID

Original Table:

FORMS\_SUPPORT\_AGENTS (supportTeamID, empID)

Normalized Table:

FORMS\_SUPPORT\_AGENTS (supportTeamID, empID)  
 Foreign Key (supportTeamID) REFERENCES SUPPORT\_TEAMS (supportTeamID)  
 Foreign Key (empID) REFERENCES TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)

**MANAGES**

FD1: managerID, supportAgentID -> managerID, supportAgentID

ORIGINAL TABLE:

MANAGES(managerID, supportAgentID)

NORMALIZED TABLE:

MANAGES(managerID, supportAgentID)  
 Foreign Key (managerID) REFERENCES TA\_MANAGERS (empID)  
 Foreign Key (supportAgentID) REFERENCES TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)

**OFFERS**

FD1: benefitID, policyID -> benefitID, policyID

Original Table:

OFFERS (policyID, benefitID)

Normalized Table:

OFFERS (policyID, benefitID)  
 Foreign Key (policyID) references POLICIES (policyID)  
 Foreign Key (benefitID) references BENEFITS (benefitID)

**REPAYMENTS:**

FD1: repaymentID, invoiceID, companyID -> amountPaid, paymentDate

Original Table:

REPAYMENTS (repaymentID, invoiceID, companyID, amountPaid, paymentDate)

Normalized Table:

REPAYMENTS (repaymentID, invoiceID, companyID, amountPaid, paymentDate)

Foreign key (invoiceID) references CC\_INVOICES (invoiceID)

Foreign key (companyID) references CLIENT\_COMPANIES (companyID)

**RESOLVES**

FD1: complaintID, empID -> complaintID, empID

Original Table:

RESOLVES (empID, complaintID)

Normalized Table:

RESOLVES (empID, complaintID)

Foreign Key (empID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)

Foreign Key (complaintID) references COMPLAINTS (complaintID)

**ROLE\_LOGIN\_DEFINITION**

FD1: roleID, loginID -> roleID, loginID

Original Table:

ROLE\_LOGIN\_DEFINITION (roleID, loginID)

Normalized Table:

ROLE\_LOGIN\_DEFINITION (roleID, loginID)

Foreign Key (roleID) references ROLES (roleID)

Foreign Key (loginID) references CC\_EMP\_LOGINS (loginID)

## **TABLES AFTER NORMALIZATION**

Strong Entities

|  |
| --- |
| * BANKS (bankID, bName) |
| * BANK\_BRANCHES (routingNum, phoneNum, location, emailID, name, bankID)   Foreign Key (bankID) references BANKS (bankID) |
| * BENEFITS (benefitID, benefitDetails, couponCode) |
| * CARDS (cardNum, cardType, iDate, cvv, category, eDate, bankID, contractID)  Foreign Key (bankID) references BANKS (bankID)  Foreign Key (contractID) references CONTRACTS (contractID) |
| * CARD\_EXPENSES (transactionID, expenseDate, expenseTime, location, description, amount, cardNum)  Foreign Key (cardNum) references CARDS (cardNum) |
| * CC\_BRANCHES (branchID, location, contactNumber, companyID)  Foreign key (companyID) references CLIENT\_COMPANIES (companyID) |
| * CC\_DEPARTMENTS (deptID, empCount, totalExpenseIncurred, deptName, companyID)   Foreign key (companyID) references CLIENT\_COMPANIES (companyID) |
| * CC\_EMP\_LOGINS\_USERNAME (loginID, userName) |
| * CC\_EMP\_LOGINS\_PASSWORD (userName, password)   Foreign Key (userName) references CC\_EMP\_LOGINS\_USERNAME (userName) |
| * CC\_EMPLOYEES (empID, fName, lName, address, dateOfJoining, dateOfBirth, gender, companyID, loginID)   Foreign Key (companyID) references CLIENT\_COMPANIES (companyID)  Foreign Key (loginID) references CC\_EMP\_LOGINS\_USERNAME (loginID)   CC\_MANAGERS (managerID, assignedBudget)  Foreign Key (managerID) references CC\_EMPLOYEES (empID) |
| * CC\_EMPLOYEES\_EMAILS (empID, emailID)   Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * CC\_EMPLOYEES\_PHONES (empID, phoneNum)   Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * CC\_INVOICES (invoiceID, totalAmount, dateOfIssue, dueDate, status, companyID, managerID)  Foreign key (companyID) references CLIENT\_COMPANIES (companyID)  Foreign key (managerID) references TA\_MANAGERS (managerID) |
| * CLIENT\_COMPANIES (companyID, name, type) |
| * CLIENT\_COMPANIES\_EMAILID (companyID, emailID)   Foreign Key (companyID) references CLIENT\_COMPANIES (companyID) |
| * CLIENT\_COMPANIES\_PHONENUM (companyID, phoneNum)   Foreign Key (companyID) references CLIENT\_COMPANIES (companyID) |
| * COMPLAINTS (complaintID, type, filedDate, resolvedDate, status, complaintDescription, empID)   Foreign key (empID) references CC\_EMPLOYEES (empID) |
| * CONTRACTS (contractID, startDate, duration, endDate, policyID, companyID)  Foreign Key (policyID) references POLICIES (policyID)  Foreign Key (companyID) references CLIENT\_COMPANIES (companyID)   ACTIVE (contractID, creditScore)  Foreign Key (contractID) references CONTRACTS (contractID)  CANCELLED (contractID, dateOfCancellation, reasonForCancellation)  Foreign Key (contractID) references CONTRACTS (contractID) |
| * POLICIES (policyID, policyName, policyDescription, policyTier, pricingPlan)   GROWTH\_PLANS (policyID, additionalPaymentPerNewUser)  Foreign Key (policyID) references POLICIES (policyID)  PROFESSIONAL\_PLANS (policyID, additionalCorporateDeals)  Foreign Key (policyID) references POLICIES (policyID) |
| * REIMBURSEMENTS (reimbursementID, expenseDate, location, amount, modeOfExpense, typeOfExpenseInccured, empID, status)   Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * ROLES (roleID, name, description) |
| * SUPPORT\_TEAMS (supportTeamID, name, companyID)   Foreign Key (companyID) references CLIENT\_COMPANIES (companyID) |
| * TA\_EMPLOYEES (empID, fName, lName, dateOfJoining, gender, address, loginID)  Foreign Key (loginID) references TA\_LOGINS (loginID)   TA\_ADMINISTRATORS (empID)  Foreign Key (empID) references TA\_EMPLOYEES (empID)   TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)  Foreign Key (empID) references TA\_EMPLOYEES (empID)   TA\_MANAGERS (empID)  Foreign Key (empID) references TA\_ADMINISTRATORS (empID) |
| * TA\_EMPLOYEES\_EMAILID (empID, emailID)  Foreign Key (empID) references TA\_EMPLOYEES (empID) |
| * TA\_EMPLOYEES\_PHONE\_NUM (empID, phoneNum)  Foreign Key (empID) references TA\_EMPLOYEES (empID) |
| * TA\_INVOICES (invoiceID, totalAmount, dateOfIssue, bankID)   Foreign Key (bankID) references BANKS (bankID) |
| * TA\_LOGINS\_USERNAME (loginID, userName) |
| * TA\_LOGINS\_PASSWORD (userName, password)   Foreign Key (userName) references TA\_LOGINS\_USERNAME (userName) |

Relationships and Weak Entities

|  |
| --- |
| * ADMINISTERS (empID, contractID)  Foreign Key (contractID) references CONTRACTS (contractID)  Foreign Key (empID) references TA\_ADMINISTRATORS (empID) |
| * BROKERAGES (invoiceID, empID, brokerageAmount)   Foreign Key (invoiceID) references TA\_INVOICES (invoiceID)  Foreign Key (empID) references TA\_MANAGERS (empID) |
| * CARD\_SUBSCRIPTIONS (empID, cardNum, startDate, endDate, validity)   Foreign Key (empID) references CC\_EMPLOYEES (empID)  Foreign Key (cardNum) references CARDS (cardNum) |
| * CC\_MANAGES (managerID, employeeID)   + Foreign Key (managerID) references CC\_MANAGERS (managerID)   + Foreign Key (employeeID) references CC\_EMPLOYEES (empID) |
| * CONSISTS (deptID, empID)   + Foreign Key (deptID) references CC\_DEPARTMENTS (deptID)   + Foreign Key (empID) references CC\_EMPLOYEES (empID) |
| * DEFINE (roleID, loginID)   Foreign Key (roleID) references ROLES (roleID)  Foreign Key (loginID) references CC\_EMP\_LOGINS (loginID) |
| * FORMS\_MANAGERS (supportTeamID, empID)  Foreign Key (supportTeamID) references SUPPORT\_TEAMS (supportTeamID)  Foreign Key (empID) references TA\_MANAGERS (empID) |
| * FORMS\_SUPPORT\_AGENTS (supportTeamID, empID)  Foreign Key (supportTeamID) references SUPPORT\_TEAMS (supportTeamID)  Foreign Key (empID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID) |
| * MANAGES (managerID, supportAgentID)  Foreign Key (managerID) references TA\_MANAGERS (empID)  Foreign Key (supportAgentID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID) |
| * OFFERS (policyID, benefitID)  Foreign Key (policyID) references POLICIES (policyID)  Foreign Key (benefitID) references BENEFITS (benefitID) |
| * RESOLVES (empID, complaintID)  Foreign Key (empID) references TA\_CUSTOMER\_SUPPORT\_AGENTS (empID)  Foreign Key (complaintID) references COMPLAINTS (complaintID) |
| * ROLE\_LOGIN\_DEFINITION (roleID, loginID)   + Foreign Key (roleID) references ROLES (roleID)   + Foreign Key (loginID) references CC\_EMP\_LOGINS (loginID) |

## **RELATIONAL DATA DICTIONARY:**

|  |  |  |
| --- | --- | --- |
| **Schema Construct** | **Data type** | **Constraint** |
| **BANKS** | **Relation representing the entity class BANKS** | |
| * bankID | varchar2 (4) | Primary Key |
| * bName | varcahr2 (10) | Not NULL |
| FD: bankID -> bname | | |

|  |  |  |
| --- | --- | --- |
| **BANK\_BRANCHES** | **Relation representing the entity class BANK BRANCHES** | |
| * bankID | Varchar2 (4) |  |
| * emailID | Varchar2 (10) |  |
| * location | Varchar2 (20) |  |
| * name | Varchar2 (10) |  |
| * phoneNum | number (10,0) |  |
| * routingNum | Number (8,0) | Primary Key |
| FD: routingNum -> phoneNum, location, emailID, name  routingNum -> bankID | | |

|  |  |  |
| --- | --- | --- |
| **BENEFITS** | **Relation representing the entity class BENEFITS** | |
| * benefitDetails | Varchar2 (10) |  |
| * benefitID | Varchar2 (4) | Primary Key |
| * couponCode | Varchar2 (8) | Not NULL |
| FD: BenefitID -> benefitDetails, couponCode | | |

|  |  |  |
| --- | --- | --- |
| **CARDS** | **Relation representing the entity class CARDS** | |
| * bankID | Varchar2 (4) | Foreign Key references BANKS |
| * cardNum | Number (16) | Primary Key |
| * cardType | Varchar2 (10) | Not NULL |
| * category | Varchar2 (10) | Not NULL |
| * contractID | Varchar2 (4) | Foreign Key references CONTRACTS |
| * cvv | Number (3) | Not NULL |
| * eDate | Date | Not NULL |
| * iDate | Date |  |
| FDs: cardNum -> cardType, iDate, cvv, category, eDate  cardNum -> bankID, contractID | | |

|  |  |  |
| --- | --- | --- |
| **CARD\_EXPENSES** | **Relation representing the entity class CARD\_EXPENSES** | |
| * amount | Number (8,2) | CHECK (amount >= 0) |
| * cardNum | Number (16) | Foreign Key references CARDS |
| * description | Varchar2 (10) |  |
| * expenseDate | Date |  |
| * expenseTime | Varchar2 (10) |  |
| * location | Varchar2 (10) |  |
| * transactionID | Varchar2 (4) | Primary Key |
| FDs: transactionID -> expenseDate, expenseTime, location, description, amount  transactionID -> cardNum | | |

|  |  |  |
| --- | --- | --- |
| **CC\_BRANCHES** | **Relation representing the entity class CC BRANCHES** | |
| * branchID | Varchar2 (4) | Primary Key |
| * companyID | Varchar2 (4) | Foreign Key references CLIENT\_COMPANIES |
| * contactNumber | Number (10) |  |
| * location | Varchar2 (10) |  |
| FD: branchID -> location, contatNumber, companyID | | |

|  |  |  |
| --- | --- | --- |
| **CC\_DEPARTMENTS** | **Relation representing the entity class CC DEPARTMENTS** | |
| * companyID | Varchar2 (4) | Foreign Key references CLIENT\_COMPANIES |
| * deptID | Varchar2 (8) | Primary Key |
| * deptName | Varchar2 (20) |  |
| * empCount | Number (4) |  |
| * totalExpenseIncurred | Number (11,2) | CHECK (totalExpenseIncurred >= 0) |
| FD: deptID –> empCount, totalExpenseIncurred, deptName, companyID | | |

|  |  |  |
| --- | --- | --- |
| **CC\_EMP\_LOGINS\_PASSWORD** | **Relation representing the entity class CC EMP LOGINS** | |
| * password | Varchar2 (10) | Not NULL |
| * userName | Varchar2 (10) | Primary Key, foreign key references CC\_EMP\_LOGINS\_USERNAME |
| FD: userName -> password | | |

|  |  |  |
| --- | --- | --- |
| **CC\_EMP\_LOGINS\_USERNAME** | **Relation representing the entity class CC EMP LOGINS** | |
| * loginID | Varchar2 (8) | Primary Key |
| * userName | Varchar2 (10) | UNIQUE |
| FD: loginID -> userName, password | | |

|  |  |  |
| --- | --- | --- |
| **CC\_EMPLOYEES** | **Relation representing the entity class CC\_EMPLOYEES** | |
| * address | Varchar2 (50) | CHECK (amount >= 0) |
| * companyID | Varchar2 (4) | Foreign Key references CLIENT\_COMPANIES |
| * dateOfBirth | Date |  |
| * dateOfJoining | Date | Not NULL |
| * empID | Varchar2 (4) | Primary Key |
| * fName | Varchar2 (50) |  |
| * gender | Varchar2 (10) |  |
| * lName | Varchar2 (50) |  |
| * loginID | Varchar2 (8) | Foreign key references CC\_EMP\_LOGINS\_USERNAME |
| FD: empID -> fName, lName, address, dateOfJoining, dateOfBirth, gender, companyID, loginID | | |

|  |  |  |
| --- | --- | --- |
| **CC\_MANAGERS** | **Relation representing the entity sub-class CC MANAGERS** | |
| * assignedBudget | Number(7) | CHECK (assignedBudget >= 0) |
| * managerID | Varchar2 (4) | Primary Key, Foreign Key references CC\_EMPLOYEES |
| FD: managerID -> assignedBudget | | |

|  |  |  |
| --- | --- | --- |
| **CC\_EMPLOYEES\_EMAILS** | **Relation representing the Email multi-valued attribute for CC EMPLOYEES** | |
| * emailID | Varchar2 (50) | Primary Key |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references CC\_EMPLOYEES |
| FD: emailID -> empID | | |

|  |  |  |
| --- | --- | --- |
| **CC\_EMPLOYEES\_PHONES** | **Relation representing the phoneNum multi-valued attribute for CC EMPLOYEES** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references CC\_EMPLOYEES |
| * phoneNum | Number (10) | Primary Key |
| FD: phoneNum -> empID | | |

|  |  |  |
| --- | --- | --- |
| **CC\_INVOICES** | **Relation representing the entity class CC INVOICES** | |
| * companyID | Varchar2 (4) | Foreign key references CLIENT\_COMPANIES |
| * dateOfIssue | Date | Not NULL |
| * dueDate | Date |  |
| * invoiceID | Varchar2 (6) | Primary Key |
| * managerID | Varchar2 (4) | Foreign Key references TA\_MANAGERS |
| * status | Varchar2 (10) | DEFAULT 'Pending’, Not NULL |
| * totalAmount | Number (9,2) | CHECK (totalAmount >= 0) |
| FD: invoiceID -> totalAmount, dateOfIssue, status, dueDate, companyID, managerID | | |

|  |  |  |
| --- | --- | --- |
| **CLIENT\_COMPANIES** | **Relation representing the entity class CLIENT COMPANIES** | |
| * companyID | Varchar2 (4) | Primary Key |
| * name | Varchar2 (20) | Not NULL |
| * type | Varchar2 (10) |  |
| FD: companyID -> name, type | | |

|  |  |  |
| --- | --- | --- |
| **CLIENT\_COMPANIES\_EMAILID** | **Relation representing the Email multi-valued attribute for CLIENT COMPANIES** | |
| * companyID | Varchar2 (4) | Primary Key, Foreign Key references CLIENT\_COMPANIES |
| * emailID | Varchar2 (50) | Primary Key |
| FD: emailID -> companyID | | |

|  |  |  |
| --- | --- | --- |
| **CC\_EMPLOYEES\_PHONES** | **Relation representing the phoneNum multi-valued attribute for CLIENT COMPANIES** | |
| * companyID | Varchar2 (4) | Primary Key, Foreign Key references CLIENT\_COMPANIES |
| * phoneNum | Number (10) | Primary Key |
| FD: phoneNum -> companyID | | |

|  |  |  |
| --- | --- | --- |
| **COMPLAINTS** | **Relation representing the entity class COMPLAINTS** | |
| * complaintDescription | Varchar2 (50) |  |
| * complaintID | Varchar2 (4) | Primary Key |
| * fileDate | date | Not NULL |
| * resolveDate | date |  |
| * status | Varchar2 (20) | Not NULL, DEFAULT ‘Open’, CHECK (status in ('Open', 'Resolved','In progress')) |
| * type | Varchar2 (25) |  |
| FD: complaintID -> type, filedDate, resolvedDate, status, complaintDescription | | |

|  |  |  |
| --- | --- | --- |
| **CONTRACTS** | **Relation representing the entity class CONTRACTS** | |
| * contractID | Varchar2 (4) | Primary Key |
| * companyID | Varchar2 (4) | Foreign key, references CLIENT\_COMPANIES |
| * duration | Number (3) |  |
| * endDate | date | Not NULL |
| * policyID | Varchar2 (4) | Foreign key, references POLICIES |
| * startDate | date | Not NULL |
| FD: contractID -> starteDate, duration, endDate, policyID, companyID | | |

|  |  |  |
| --- | --- | --- |
| **ACTIVE** | **Relation representing the entity sub-class of CONTRACTS (ACTIVE Contracts)** | |
| * contractID | Varchar2 (4) | Primary Key, Foreign Key references CONTRACTS |
| * creditScore | Number (4) | Not NULL |
| FD: contractID -> creditScore | | |

|  |  |  |
| --- | --- | --- |
| **CANCELLED** | **Relation representing the entity sub-class of CONTRACTS (CANCELLED Contracts)** | |
| * contractID | Varchar2 (4) | Primary Key, Foreign Key references CONTRACTS |
| * dateOfCancellation | date | Not NULL |
| * reasonForCancellation | Varchar2 (50) |  |
| FD: contractID -> dateOfCancellation, reasonForCancellation | | |

|  |  |  |
| --- | --- | --- |
| **POLICIES** | **Relation representing the entity class POLICIES** | |
| * policyDescription | Varchar2 (50) |  |
| * policyID | Varchar2 (4) | Primary Key |
| * policyName | Varchar2 (25) |  |
| * policyTier | Varchar2 (50) |  |
| * pricingPlan | Number (5) |  |
| FD: policyID -> policyName, PolicyDescription, policyTier, pricingPlan | | |

|  |  |  |
| --- | --- | --- |
| **GROWTH\_PLANS** | **Relation representing the entity sub-class of POLICIES (GROWTH\_PLAN Policy)** | |
| * additionalPaymentPerNewUser | Number (9,2) | CHECK (additionalPaymentPerNewUser >= 0) |
| * policyID | Varchar2 (4) | Primary Key, Foreign Key references POLICIES |
| FD: policyID -> additionalPaymentPerNewUser | | |

|  |  |  |
| --- | --- | --- |
| **PROFESSIONAL\_PLANS** | **Relation representing the entity sub-class of POLICIES (PROFESSIONAL\_PLAN Policy)** | |
| * additionalCorporateDeals | Varchar2 (50) |  |
| * policyID | Varchar2 (4) | Primary Key, Foreign Key references POLICIES |
| FD: policyID -> additionalCorporateDeals | | |

|  |  |  |
| --- | --- | --- |
| **REIMBURSEMENTS** | **Relation representing the entity class REIMBURSEMENTS** | |
| * amount | Number (9,2) | CHECK (amount > 0) |
| * empID | Varchar2 (4) | Foreign Key, references CC\_EMPLOYEES |
| * expenseDate | date | Not NULL |
| * location | Varchar2 (50) |  |
| * modeOfExpense | Varchar2 (50) | Such as personal card/ cash/ zelle, etc |
| * reimbursementID | Varchar2 (4) | Primary Key |
| * status | Varchar2 (20) | DEFAULT ‘Pending’, Not NULL |
| * typeOfExpenseIncurred | Varchar2 (30) |  |
| FDs: reimbursementID -> invoice, location, amount, modeOfExpense,typeOfExpenseIncurred  reimbursementID -> empID | | |

|  |  |  |
| --- | --- | --- |
| **ROLES** | **Relation representing the entity class ROLES** | |
| * description | Varchar2 (50) |  |
| * name | Varchar2 (20) | UNIQUE, Not NULL |
| * roleID | Varchar2 (4) | Primary Key |
| FD: roleID -> name, description | | |

|  |  |  |
| --- | --- | --- |
| **SUPPORT\_TEAMS** | **Relation representing the entity class SUPPORT\_TEAMS** | |
| * name | Varchar2 (20) | Not NULL |
| * supportTeamID | Varchar2 (4) | Primary Key |
| FD: supportTeamID -> name | | |

|  |  |  |
| --- | --- | --- |
| **TA\_EMPLOYEES** | **Relation representing the entity class TA\_EMPLOYEES** | |
| * address | Varchar2 (50) |  |
| * dateOfJoining | date | Not NULL |
| * empID | Varchar2 (4) | Primary Key |
| * fName | Varchar2 (50) |  |
| * gender | Varchar2 (10) |  |
| * lName | Varchar2 (50) |  |
| * loginID | Varchar2 (50) | Foreign Key references TA\_LOGINS\_USERNAME |
| FDs: empID -> gender, fName, lName, address, dateOfJoining,loginID | | |

|  |  |  |
| --- | --- | --- |
| **TA\_ADMINISTRATORS** | **Relation representing the entity class TA\_ADMINISTRATORS** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_EMPLOYEES |

|  |  |  |
| --- | --- | --- |
| **TA\_CUSTOMER\_SUPPORT\_AGENTS** | **Relation representing the entity class TA\_CUSTOMER\_SUPPORT\_AGENTS** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_EMPLOYEES |

|  |  |  |
| --- | --- | --- |
| **TA\_MANAGERS** | **Relation representing the entity class TA\_MANAGERS** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_EMPLOYEES |

|  |  |  |
| --- | --- | --- |
| **TA\_EMPLOYEES\_EMAILID** | **Relation representing the entity class TA\_EMPLOYEES\_EMAILID** | |
| * emailID | Varchar2 (50) | Primary Key |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_EMPLOYEES |
| FD: emailID -> empID | | |

|  |  |  |
| --- | --- | --- |
| **TA\_EMPLOYEES\_PHONE\_NUM** | **Relation representing the entity class TA\_EMPLOYEES\_PHONE\_NUM** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_EMPLOYEES |
| * phoneNum | varchar2 (12) | Primary Key |
| FD: phoneNum -> empID | | |

|  |  |  |
| --- | --- | --- |
| **TA\_INVOICES** | **Relation representing the entity class TA\_INVOICES** | |
| * bankID | Varchar2 (4) | Foreign Key references BANKS |
| * dateOfIssue | date |  |
| * invoiceID | Varchar2 (6) | Primary Key |
| * totalAmount | Number (9,2) |  |
| FD: invoiceID -> totalAmount, dateOfIssue  invoiceID -> bankID | | |

|  |  |  |
| --- | --- | --- |
| **TA\_LOGINS\_PASSWORD** | **Relation representing the entity class TA\_LOGINS\_PASSWORD** | |
| * password | Varchar2 (50) | Not NULL |
| * userName | Varchar2 (25) | Primary Key, Foreign Key references TA\_LOGINS\_USERNAME |
| FD: userName -> password | | |

|  |  |  |
| --- | --- | --- |
| **TA\_LOGINS\_USERNAME** | **Relation representing the entity class TA\_LOGINS\_USERNAME** | |
| * loginID | Varchar2 (50) | Primary Key |
| * userName | Varchar2 (25) | UNIQUE |
| FD: loginID -> userName, password | | |

**WEAK ENTITIES AND RELATIONSHIPS**

|  |  |  |
| --- | --- | --- |
| **ADMINISTERS** | **Relation representing the relationship ADMINISTERS** | |
| * contractID | Varchar2 (4) | Primary Key, Foreign Key references CONTRACTS |
| * empID | Varchar2 (4) | Primary Key, foreign Key references TA\_ADMINISTRATORS |
| Primary Key Constraint: contractID, empID FD: contractID, empID -> contractID, empID | | |

|  |  |  |
| --- | --- | --- |
| **BROKERAGES** | **Relation representing the weak entity class BROKERAGES** | |
| * brokerageAmount | Number (6,2) | CHECK (brokerageAmount >= 0) |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_MANAGERS |
| * invoiceID | Varchar2 (4) | Primary Key, foreign Key references TA\_INVOICES |
| Primary Key Constraint: empID, invoiceID FD: empID, invoiceID -> brokerageAmount | | |

|  |  |  |
| --- | --- | --- |
| **CARD\_SUBSCRIPTIONS** | **Relation representing the weak entity class CARD\_SUBSCRIPTIONS** | |
| * cardNum | Number (16) | Primary Key, Foreign Key references CARDS |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references CC\_EMPLOYEES |
| * endDate | Date | Not NULL |
| * startDate | Date | Primary Key |
| * validity | Number (4) |  |
| Primary Key Constraint: cardNum, empID, startDate  FD: empID,cardNum -> startDate, endDate, validity | | |

|  |  |  |
| --- | --- | --- |
| **CC\_MANAGES** | **Relation representing the relationship CC\_MANAGES** | |
| * employeeID | Varchar2 (4) | Primary Key, Foreign Key references CC\_EMPLOYEES |
| * managerID | Varchar2 (4) | Primary Key, Foreign Key references CC\_MANAGERS |
| Primary Key Constraint: employeeID, managerID FD: employeeID, managerID -> employeeID, managerID | | |

|  |  |  |
| --- | --- | --- |
| **CONSISTS** | **Relation representing the relationship CONSISTS** | |
| * deptID | Varchar2 (4) | Primary Key, Foreign Key references CC\_DEPARTMENTS |
| * empID | Varchar2 (4) | Primary Key, foreign Key references CC\_EMPLOYEES |
| Primary Key Constraint: deptID, empID FD: deptID, empID -> deptID, empID | | |

|  |  |  |
| --- | --- | --- |
| **FORMS** | **Relation representing the relationship FORMS** | |
| * benefitID | Varchar2 (4) | Primary Key, Foreign Key references BENEFITS |
| * policyID | Varchar2 (4) | Primary Key, foreign Key references POLICIES |
| Primary Key Constraint: benefitID, policyID FD: benefitID, policyID -> benefitID, policyID | | |

|  |  |  |
| --- | --- | --- |
| **FORMS\_MANAGERS** | **Relation representing the relationship FORMS between TA\_MANAGERS and SUPPORT\_TEAMS** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_MANAGERS |
| * supportTeamID | Varchar2 (4) | Primary Key, foreign Key references SUPPORT\_TEAMS |
| Primary Key Constraint: empID, supportTeamID FD: empID, supportTeamID -> empID, supportTeamID | | |

|  |  |  |
| --- | --- | --- |
| **FORMS\_SUPPORT\_AGENTS** | **Relation representing the relationship FORMS between TA\_CUSTOMER\_SUPPORT\_AGENTS and SUPPORT\_TEAMS** | |
| * empID | Varchar2 (4) | Primary Key, Foreign Key references TA\_CUSTOMER\_SUPPORT\_AGENTS |
| * supportTeamID | Varchar2 (4) | Primary Key, foreign Key references SUPPORT\_TEAMS |
| Primary Key Constraint: empID, supportTeamID FD: empID, supportTeamID -> empID, supportTeamID | | |

|  |  |  |
| --- | --- | --- |
| **MANAGES** | **Relation representing the relationship MANAGES** | |
| * managerID | Varchar2 (4) | Primary Key, Foreign Key references TA\_MANAGERS |
| * supportAgentID | Varchar2 (4) | Primary Key, foreign Key references TA\_CUSTOMER\_SUPPORT\_AGENTS |
| Primary Key Constraint: managerID, supportAgentID FD: managerID, supportAgentID -> managerID, supportAgentID | | |

|  |  |  |
| --- | --- | --- |
| **OFFERS** | **Relation representing the relationship OFFERS** | |
| * benefitID | Varchar2 (4) | Primary Key, Foreign Key references BENEFITS |
| * policyID | Varchar2 (4) | Primary Key, foreign Key references POLICIES |
| Primary Key Constraint: benefitID, policyID FD: benefitID, policyID -> benefitID, policyID | | |

|  |  |  |
| --- | --- | --- |
| **RESOLVES** | **Relation representing the relationship RESOLVES** | |
| * complaintID | Varchar2 (4) | Primary Key, Foreign Key references COMPLAINTS |
| * empID | Varchar2 (4) | Primary Key, foreign Key references TA\_CUSTOMER\_SUPPORT\_AGENTS |
| Primary Key Constraint: complaintID, empID FD: complaintID, empID -> complaintID, empID | | |

|  |  |  |
| --- | --- | --- |
| **ROLE\_LOGIN\_DEFINITION** | **Relation representing the relationship DEFINE** | |
| * loginID | Varchar2 (4) | Primary Key, Foreign Key references CC\_EMP\_LOGINS\_USERNAME |
| * roleID | Varchar2 (4) | Primary Key, foreign Key references ROLES |
| Primary Key Constraint: loginID, roleID FD: loginID, roleID -> loginID, roleID | | |

|  |  |  |
| --- | --- | --- |
| **REPAYMENTS** | **Relation representing the entity class REPAYMENTS** | |
| * amountPaid | Number (9,2) | CHECK (amount >= 0) |
| * paymentDate | date |  |
| * repaymentID | Varchar2 (4) | Primary Key |
| FD: repaymentID -> amountPaid, paymentDate | | |

## **TABLE CREATION SCRIPTS FOR TRIPACTIONS DATABASE:**

-- Create table command for BANKS table

CREATE TABLE BANKS (

bankID VARCHAR2(4),

bName VARCHAR2(10) NOT NULL,

CONSTRAINT banks\_pk PRIMARY KEY (bankID)

);

-- Create table command for BANK\_BRANCHES table

CREATE TABLE BANK\_BRANCHES (

routingNum NUMBER(8),

phoneNum NUMBER(10),

location VARCHAR2(20),

emailID VARCHAR2(10),

name VARCHAR2(10),

bankID VARCHAR2(4),

CONSTRAINT bank\_branches\_pk PRIMARY KEY (routingNum),

CONSTRAINT bank\_branches\_fk FOREIGN KEY (bankID) REFERENCES BANKS (bankID) ON DELETE SET NULL

);

-- Create table command for BENEFITS table

CREATE TABLE BENEFITS (

benefitID VARCHAR2(4),

benefitDetails VARCHAR2(10),

couponCode VARCHAR2(8) NOT NULL,

CONSTRAINT benefits\_pk PRIMARY KEY (benefitID)

);

-- Create table command for BROKERAGES table

CREATE TABLE BROKERAGES (

invoiceID varchar2(4),

empID varchar2(4),

brokerageAmount number(6,2),

CONSTRAINT brokerage\_pk PRIMARY KEY (invoiceID, empID),

CONSTRAINT brokerage\_fk1 FOREIGN KEY (invoiceID) REFERENCES TA\_INVOICES (invoiceID) ON DELETE SET NULL,

CONSTRAINT brokerage\_fk2 FOREIGN KEY (empID) REFERENCES TA\_MANAGERS (empID) ON DELETE SET NULL,

CONSTRAINT brokerage\_chk CHECK (brokerageAmount >= 0)

);

-- Create table command for CARDS table

CREATE TABLE CARDS (

cardNum NUMBER(16),

cardType VARCHAR2(10) NOT NULL,

iDate DATE,

cvv NUMBER(3) NOT NULL,

category VARCHAR2(10) NOT NULL,

eDate DATE NOT NULL,

constraint cards\_pk PRIMARY KEY (cardNum),

CONSTRAINT cards\_fk1 FOREIGN KEY (bankID) REFERENCES BANKS (bankID) ON DELETE SET NULL,

CONSTRAINT cards\_fk2 FOREIGN KEY (contractID) REFERENCES CONTRACTS (contractID) ON DELETE SET NULL

);

-- Create table command for CARD\_EXPENSES table

CREATE TABLE CARD\_EXPENSES (

transactionID VARCHAR2(4),

expenseDate DATE,

expenseTime VARCHAR2(8),

location VARCHAR2(10),

description VARCHAR2(10),

amount NUMBER(8,2),

cardNum NUMBER(16),

CONSTRAINT card\_expenses\_pk PRIMARY KEY (transactionID),

CONSTRAINT card\_expenses\_fk FOREIGN KEY (cardNum) REFERENCES CARDS (cardNum) ON DELETE SET NULL,

CONSTRAINT card\_expenses\_chk CHECK (amount >= 0)

);

-- Create table command for CARD\_SUBSCRIPTIONS table

CREATE TABLE CARD\_SUBSCRIPTIONS (

empID VARCHAR2(4),

cardNum NUMBER(12),

startDate DATE,

endDate DATE NOT NULL,

validity NUMBER(4)

CONSTRAINT card\_subscription\_pk PRIMARY KEY (empID, cardNum, startDate),

CONSTRAINT card\_subscription\_fk1 FOREIGN KEY (empID) REFERENCES CC\_EMPLOYEES(empID) ON DELETE CASCADE,

CONSTRAINT card\_subscription\_fk2 FOREIGN KEY (cardNum) REFERENCES CARDS (cardNum) ON DELETE SET NULL

);

-- Create table command for CC\_BRANCHES table

CREATE TABLE CC\_BRANCHES (

    branchID VARCHAR2(4),

    location VARCHAR2(10),

    contactNumber NUMBER(10),

    company\_id varchar2(4),

    CONSTRAINT ccbranches\_pk PRIMARY KEY (branchID),

    CONSTRAINT ccbranches\_fk1 FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL

);

-- Create table command for CC\_DEPARTMENTS table

CREATE TABLE CC\_DEPARTMENTS (

deptID VARCHAR2(8),

empCount NUMBER(4),

totalExpenseIncurred NUMBER(11,2) ,

deptName varchar2(20),

companyID varchar2(4),

CONSTRAINT cc\_departments\_pk PRIMARY KEY (deptID),

CONSTRAINT cc\_departments\_fk FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL,

CONSTRAINT cc\_departments\_chk CHECK (totalExpenseIncurred >= 0)

);

-- Create table command for CC\_EMPLOYEES table

CREATE TABLE CC\_EMPLOYEES (

empID varchar2(4),

fName VARCHAR2(50),

lName VARCHAR2(50),

address varchar2(50),

dateOfJoining date NOT NULL,

dateOfBirth date,

gender varchar2(10),

companyID varchar2(4),

loginID varchar2(8),

CONSTRAINT cc\_employees\_pk PRIMARY KEY (empID),

CONSTRAINT cc\_employees\_fk1 FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL,

CONSTRAINT cc\_employees\_fk2 FOREIGN KEY (loginID) REFERENCES CC\_EMP\_LOGINS\_USERNAME (loginID) ON DELETE SET NULL

);

-- Create table command for CC\_MANAGERS table

CREATE TABLE CC\_MANAGERS (

managerID varchar2(4),

assignedBudget number(7),

CONSTRAINT cc\_managers\_pk PRIMARY KEY (managerID),

CONSTRAINT cc\_managers\_fk FOREIGN KEY (managerID) REFERENCES CC\_EMPLOYEES (empID) ON DELETE SET NULL,

CONSTRAINT cc\_managers\_chk CHECK (assignedBudget >= 0)

);

-- Create table command for CC\_INVOICES table

CREATE TABLE CC\_INVOICES (

invoiceID varchar2(6),

totalAmount number(9,2) NOT NULL,

dateOfIssue date NOT NULL,

companyID varchar2(4),

managerID varchar2(4),

status varchar2(10) DEFAULT 'Pending’ NOT NULL

dueDate date,

CONSTRAINT cc\_invoices\_pk PRIMARY KEY (invoiceID),

CONSTRAINT cc\_invoices\_fk1 FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL,

CONSTRAINT cc\_invoices\_fk2 FOREIGN KEY (managerID) REFERENCES TA\_MANAGERS (managerID) ON DELETE SET NULL,

CONSTRAINT cc\_invoices\_chk CHECK (totalAmount >= 0)

);

-- Create table command for CLIENT\_COMPANIES table

CREATE TABLE CLIENT\_COMPANIES (

companyID VARCHAR2(4),

name VARCHAR2(20) NOT NULL,

type VARCHAR2(10),

CONSTRAINT client\_companies\_pk PRIMARY KEY (companyID)

);

-- Create table command for COMPLAINTS table

CREATE TABLE COMPLAINTS (

complaintID varchar2(4),

type varchar2(25),

fileDate date NOT NULL,

resolveDate date,

status varchar2(20) DEFAULT 'Open' NOT NULL,

complaintDescription varchar2(50),

CONSTRAINT complaints\_pk PRIMARY KEY (complaintID),

CONSTRAINT complaints\_chk CHECK (status in (‘Open’,’Resolved’,’In progress’))

ENABLE NOVALIDATE,

CONSTRAINT complaints\_FK FOREIGN KEY (EMPID)  
      REFERENCES CC\_EMPLOYEES (EMPID)  ON DELETE CASCADE;

-- Create table command for CONTRACTS table

CREATE TABLE CONTRACTS (

contractID varchar2(4),

startDate date NOT NULL,

duration number(3) DEFAULT 12,

endDate date not null,

policyID varchar2(4),

companyID varchar2(4),

CONSTRAINT contracts\_pk PRIMARY KEY (contractID),

CONSTRAINT contracts\_fk1 FOREIGN KEY (policyID) REFERENCES POLICIES (policyID) ON DELETE SET NULL,

CONSTRAINT contracts\_fk2 FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL

);

-- Create table command for ACTIVE table

CREATE TABLE ACTIVE (

contractID VARCHAR2(4),

creditScore NUMBER(4) NOT NULL,

CONSTRAINT active\_pk PRIMARY KEY (contractID),

CONSTRAINT active\_fk FOREIGN KEY (contractID) REFERENCES CONTRACTS (contractID) ON DELETE SET NULL

);

-- Create table command for CANCELLED table

CREATE TABLE CANCELLED (

contractID VARCHAR2(4),

dateOfCancellation date NOT NULL,

reasonForCancellation VARCHAR2(50),

CONSTRAINT cancelled\_pk PRIMARY KEY (contractID),

CONSTRAINT cancelled\_fk FOREIGN KEY (contractID) REFERENCES CONTRACTS (contractID) ON DELETE SET NULL

);

-- Create table command for POLICIES table

CREATE TABLE POLICIES (

policyID varchar2(4),

policyName varchar2(25),

policyDescription varchar2(50),

policyTier varchar2(50),

pricingPlan NUMBER(5),

CONSTRAINT policies\_pk PRIMARY KEY (policyID)

);

-- Create table command for GROWTH\_PLANS table

CREATE TABLE GROWTH\_PLANS (

policyID varchar2(4),

additionalPaymentPerNewUser number(9,2),

CONSTRAINT growth\_plans\_pk PRIMARY KEY (policyID),

CONSTRAINT growth\_plans\_fk FOREIGN KEY (policyID) REFERENCES POLICIES (policyID) ON DELETE SET NULL,

CONSTRAINT growth\_plans\_chk CHECK (additionalPaymentPerNewUser >= 0)

);

-- Create table command for PROFESSIONAL\_PLANS table

CREATE TABLE PROFESSIONAL\_PLANS (

policyID varchar2(4),

additionalCorporateDeals varchar2(50),

CONSTRAINT professional\_plans\_pk PRIMARY KEY (policyID),

CONSTRAINT professional\_plans\_fk FOREIGN KEY (policyID) REFERENCES POLICIES (policyID) ON DELETE SET NULL

);

-- Create table command for REIMBURSEMENTS table

CREATE TABLE REIMBURSEMENTS (

reimbursementID varchar2(4),

expenseDate date NOT NULL,

location varchar2(50),

amount number(9,2),

modeOfExpense varchar2(50)

typeOfExpenseIncurred varchar2(50),

empID varchar2(4),

status varchar2(20) DEFAULT 'Pending' NOT NULL,

CONSTRAINT reimbursements\_pk PRIMARY KEY (reimbursementID),

CONSTRAINT reimbursements\_fk FOREIGN KEY (empID) REFERENCES CC\_EMPLOYEES (empID) ON DELETE SET NULL,

CONSTRAINT reimbursements\_chk CHECK (amount > 0)

);

-- Create table command for REPAYMENTS table

CREATE TABLE REPAYMENTS (

repaymentID varchar2(4),

amountPaid number(9,2),

paymentDate date,

CONSTRAINT repayments\_pk PRIMARY KEY (repaymentID),

CONSTRAINT repayments\_chk CHECK (amount >= 0)

);

-- Create table command for ROLES table

CREATE TABLE ROLES (

roleID varchar2(4),

name varchar2(20) NOT NULL,

description varchar2(50),

CONSTRAINT roles\_pk PRIMARY KEY (roleID),

CONSTRAINT roles\_uq UNIQUE (name)

);

-- Create table command for SUPPORT\_TEAMS table

CREATE TABLE SUPPORT\_TEAMS (

supportTeamID varchar2(4),

name varchar2(20) NOT NULL,

CONSTRAINT support\_teams\_pk PRIMARY KEY (supportTeamID)

);

-- Create table command for TA\_EMPLOYEES table

CREATE TABLE TA\_EMPLOYEES (

empID VARCHAR2(4),

fName VARCHAR2(50),

lName VARCHAR2(50),

dateOfJoining date NOT NULL,

gender VARCHAR2(10),

address VARCHAR2(50),

loginID VARCHAR2(50),

CONSTRAINT ta\_employees\_pk PRIMARY KEY (empID),

CONSTRAINT ta\_employees\_fk FOREIGN KEY (loginID) REFERENCES TA\_LOGINS\_USERNAME (loginID) ON DELETE SET NULL

);

-- Create table command for TA\_ADMINISTRATORS table

CREATE TABLE TA\_ADMINISTRATORS (

empID VARCHAR2(4),

CONSTRAINT ta\_administrators\_pk PRIMARY KEY (empID),

CONSTRAINT ta\_administrators\_fk FOREIGN KEY(empID) REFERENCES TA\_EMPLOYEES (empID) ON DELETE SET NULL

);

-- Create table command for TA\_MANAGERS table

CREATE TABLE TA\_MANAGERS (

empID VARCHAR2(4),

CONSTRAINT ta\_managers\_pk PRIMARY KEY (empID),

CONSTRAINT ta\_managers\_fk FOREIGN KEY(empID) REFERENCES TA\_ADMINISTRATORS (empID) ON DELETE SET NULL

);

-- Create table command for TA\_CUSTOMER\_SUPPORT\_AGENTS table

CREATE TABLE TA\_CUSTOMER\_SUPPORT\_AGENTS (

empID VARCHAR2(4),

CONSTRAINT ta\_customer\_support\_pk PRIMARY KEY (empID),

CONSTRAINT ta\_customer\_support\_fk FOREIGN KEY (empID) REFERENCES TA\_EMPLOYEES (empID) ON DELETE SET NULL

);

-- Create table command for TA\_INVOICES table

CREATE TABLE TA\_INVOICES (invoiceID varchar2(6),

totalAmount NUMBER(9,2),

dateOfIssue DATE,

bankID VARCHAR2(4),

CONSTRAINT ta\_invoices\_pk PRIMARY KEY (invoiceID),

CONSTRAINT ta\_invoices\_fk FOREIGN KEY (bankID) REFERENCES BANKS (bankID) ON DELETE SET NULL,

CONSTRAINT ta\_invoices\_chk CHECK (totalAmount >= 0)

);

-- Create table command for TA\_LOGINS\_USERNAME table

CREATE TABLE TA\_LOGINS\_USERNAME (

loginID varchar2(50),

userName varchar2(25),

CONSTRAINT ta\_logins\_username\_pk PRIMARY KEY (loginID),

CONSTRAINT ta\_logins\_username\_uq UNIQUE (userName)

);

-- Create table command for TA\_LOGIN\_PASSWORD table

CREATE TABLE TA\_LOGINS\_PASSWORD (

userName varchar2(25),

password varchar2(50) NOT NULL,

CONSTRAINT ta\_logins\_password\_pk PRIMARY KEY (userName),

CONSTRAINT ta\_logins\_password\_fk FOREIGN KEY (userName) REFERENCES TA\_LOGINS\_USERNAME(userName) ON DELETE CASCADE

);

-- Create table command for ADMINISTERS table

CREATE TABLE ADMINISTERS (

empID varchar2(4),

contractID varchar2(4),

CONSTRAINT administers\_pk PRIMARY KEY (empID, contractID),

CONSTRAINT administers\_fk1 FOREIGN KEY (empID) REFERENCES TA\_ADMINISTRATORS (empID) ON DELETE SET NULL,

CONSTRAINT administers\_fk2 FOREIGN KEY (contractID) REFERENCES CONTRACTS (contractID) ON DELETE SET NULL

);

-- Create table command for CC\_EMP\_LOGINS\_PASSWORD table

CREATE TABLE CC\_EMP\_LOGINS\_PASSWORD (

userName VARCHAR2(10),

password VARCHAR2(10) NOT NULL,

CONSTRAINT cc\_emp\_logins\_password\_pk PRIMARY KEY (userName),

CONSTRAINT cc\_emp\_logins\_password\_fk FOREIGN KEY (userName) REFERENCES CC\_EMP\_LOGINS\_USERNAME (userName) ON DELETE CASCADE

);

-- Create table command for CC\_EMP\_LOGINS\_USERNAME table

CREATE TABLE CC\_EMP\_LOGINS\_USERNAME (

loginID VARCHAR2(8),

userName VARCHAR2(10),

CONSTRAINT cc\_emp\_logins\_username\_pk PRIMARY KEY (loginID),

CONSTRAINT cc\_emp\_logins\_username\_uq UNIQUE (userName)

);

-- Create table command for CC\_EMPLOYEES\_EMAILS table

CREATE TABLE CC\_EMPLOYEES\_EMAILS (

empID varchar2(4),

emailID varchar2(50),

CONSTRAINT cc\_employees\_emails\_pk PRIMARY KEY (empID, emailID),

CONSTRAINT cc\_employees\_emails\_fk FOREIGN KEY (empID) REFERENCES CC\_EMPLOYEES (empID) ON DELETE SET NULL

);

-- Create table command for CC\_EMPLOYEES\_PHONES table

CREATE TABLE CC\_EMPLOYEES\_PHONES (

empID varchar2(4),

phoneNum varchar2(12),

CONSTRAINT cc\_employees\_phones\_pk PRIMARY KEY (empID, phoneNum),

CONSTRAINT cc\_employees\_phones\_fk FOREIGN KEY (empID) REFERENCES CC\_EMPLOYEES (empID) ON DELETE SET NULL

);

-- Create table command for CC\_MANAGES table

CREATE TABLE CC\_MANAGES (

managerID varchar2(4),

employeeID varchar2(4),

CONSTRAINT cc\_manages\_pk PRIMARY KEY (managerID, employeeID),

CONSTRAINT cc\_manages\_fk1 FOREIGN KEY (managerID) REFERENCES CC\_MANAGERS (managerID) ON DELETE SET NULL,

CONSTRAINT cc\_manages\_fk2 FOREIGN KEY (employeeID) REFERENCES CC\_EMPLOYEES (empID) ON DELETE CASCADE

);

-- Create table command for CLIENT\_COMPANIES\_EMAILID table

CREATE TABLE CLIENT\_COMPANIES\_EMAILID (

companyID varchar2(4),

emailID varchar2(50),

CONSTRAINT client\_companies\_emailid\_pk PRIMARY KEY (companyID, emailID),

CONSTRAINT client\_companies\_emailid\_fk FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL

);

-- Create table command for CLIENT\_COMPANIES\_PHONENUM table

CREATE TABLE CLIENT\_COMPANIES\_PHONENUM (

companyID varchar2(4),

phoneNum varchar2(12),

CONSTRAINT client\_companies\_phonenum\_pk PRIMARY KEY (companyID, phoneNum),

CONSTRAINT client\_companies\_phonenum\_fk FOREIGN KEY (companyID) REFERENCES CLIENT\_COMPANIES (companyID) ON DELETE SET NULL

);

-- Create table command for CONSISTS table

CREATE TABLE CONSISTS(

empID VARCHAR2(4),

deptID VARCHAR2(4),

CONSTRAINT cc\_consists\_fk1 FOREIGN KEY (empID) REFERENCES CC\_EMPLOYEES (empID) ON DELETE SET NULL,

CONSTRAINT cc\_consists\_fk2 FOREIGN KEY (deptID) REFERENCES CC\_DEPARTMENTS (deptID) ON DELETE SET NULL

);

-- Create table command for DEFINE table

CREATE TABLE DEFINE (  
roleid varchar(4),  
loginid varchar2(8),  
CONSTRAINT define\_pk PRIMARY KEY (roleid,loginid),  
CONSTRAINT define\_fk1 FOREIGN KEY (roleid) REFERENCES ROLES (roleid) ON DELETE CASCADE,  
CONSTRAINT define\_fk2 FOREIGN KEY (loginid) REFERENCES CC\_EMP\_LOGINS\_USERNAME (loginid) ON DELETE CASCADE);

-- Create table command for FORMS\_MANAGERS table

CREATE TABLE FORMS\_MANAGERS (

supportTeamID varchar2(4),

empID varchar2(4),

CONSTRAINT forms\_managers\_pk PRIMARY KEY (supportTeamID, empID),

CONSTRAINT forms\_managers\_fk1 FOREIGN KEY (supportTeamID) REFERENCES SUPPORT\_TEAMS (supportTeamID) ON DELETE SET NULL,

CONSTRAINT forms\_managers\_fk2 FOREIGN KEY (empID) REFERENCES TA\_MANAGERS (empID) ON DELETE SET NULL

);

-- Create table command for FORMS\_SUPPORT\_AGENTS table

CREATE TABLE FORMS\_SUPPORT\_AGENTS (

supportTeamID varchar2(4),

empID varchar2(4),

CONSTRAINT forms\_support\_agents\_pk PRIMARY KEY (supportTeamID, empID),

CONSTRAINT forms\_support\_agents\_fk1 FOREIGN KEY (supportTeamID) REFERENCES SUPPORT\_TEAMS (supportTeamID) ON DELETE SET NULL,

CONSTRAINT forms\_support\_agents\_fk2 FOREIGN KEY (empID) REFERENCES TA\_CUSTOMER\_SUPPORT\_AGENETS (empID) ON DELETE SET NULL

);

-- Create table command for OFFERS table

CREATE TABLE OFFERS (

policyID varchar2(4),

benefitID varchar2(4),

CONSTRAINT forms\_pk PRIMARY KEY (policyID, benefitID),

CONSTRAINT forms\_fk1 FOREIGN KEY (policyID) REFERENCES POLICIES (policyID) ON DELETE SET NULL,

CONSTRAINT forms\_fk2 FOREIGN KEY (benefitID) REFERENCES BENEFITS (benefitID) ON DELETE SET NULL

);

-- Create table command for MANAGES table

CREATE TABLE MANAGES (

managerID varchar2(4),

supportAgentID varchar2(4),

CONSTRAINT manages\_pk PRIMARY KEY (managerID, supportAgentID),

CONSTRAINT manages\_fk1 FOREIGN KEY (managerID) REFERENCES TA\_MANAGERS (empID) ON DELETE SET NULL,

CONSTRAINT manages\_fk2 FOREIGN KEY (supportAgentID) REFERENCES TA\_CUSTOMER\_SUPPORT\_AGENTS (empID) ON DELETE SET NULL

);

-- Create table command for RESOLVES table

CREATE TABLE RESOLVES (

empID varchar2(4),

complaintID varchar2(4),

CONSTRAINT resolves\_pk PRIMARY KEY (empID, complaintID),

CONSTRAINT resolves\_fk1 FOREIGN KEY (empID) REFERENCES TA\_CUSTOMER\_SUPPORT\_AGENTS (empID) ON DELETE SET NULL,

CONSTRAINT resolves\_fk2 FOREIGN KEY (complaintid) REFERENCES COMPLAINTS (complaintID) ON DELETE SET NULL

);

-- Create table command for ROLE\_LOGIN\_DEFINATION table

CREATE TABLE ROLE\_LOGIN\_DEFINITION (

roleID varchar2(4),

loginID varchar2(8),

CONSTRAINT role\_login\_pk PRIMARY KEY (roleID, loginID),

CONSTRAINT role\_login\_fk1 FOREIGN KEY (roleID) REFERENCES ROLES (roleID) ON DELETE SET NULL,

CONSTRAINT role\_login\_fk2 FOREIGN KEY (loginID) REFERENCES CC\_EMP\_LOGINS\_USERNAME (loginID) ON DELETE SET NULL

);

-- Create table command for TA\_EMPLOYEES\_EMAILID table

CREATE TABLE TA\_EMPLOYEES\_EMAILID (

empID varchar2(4),

emailID varchar2(50),

CONSTRAINT ta\_employees\_emailid\_pk PRIMARY KEY (empID),

CONSTRAINT ta\_employees\_emailid\_fk FOREIGN KEY (empID) REFERENCES TA\_EMPLOYEES (empID) ON DELETE SET NULL

);

-- Create table command for TA\_EMPLOYEES\_PHONE\_NUM table

CREATE TABLE TA\_EMPLOYEES\_PHONE\_NUM (

empID varchar2(4), phoneNum varchar2(12),

CONSTRAINT ta\_employees\_phone\_num\_pk PRIMARY KEY (empID),

CONSTRAINT ta\_employees\_phone\_num\_fk FOREIGN KEY (empID) REFERENCES TA\_EMPLOYEES (empID) ON DELETE SET NULL

);

# **CHAPTER 4: SQL QUERIES**

## **LIST OF OPEN AND IN PROGRESS COMPLAINTS**

Display list of complaint types under each manager with their department who have count of complaints that are open or in progress that are greater than 2. This will help managers identify the type of complaints received that are open and in progress that will help them to track and close as soon as possible.

WITH comp\_mgr\_dept AS (

SELECT COUNT(complaintid) AS "num complaints",

LISTAGG(type, ', ') WITHIN GROUP(

ORDER BY cm.managerid) AS "complaints under a manager",

cm.managerid AS "manager id",

deptname AS "department name",

cd.deptid AS "department id"

FROM complaints c

JOIN cc\_manages cm ON c.empid = cm.employeeid

JOIN consists co ON cm.managerid = co.empid

JOIN cc\_departments cd ON co.deptid = cd.deptid

JOIN cc\_managers ccm ON co.empid = ccm.managerid

JOIN cc\_employees cce ON ccm.managerid = cce.empid

WHERE ( status = 'Open' OR status = 'In progress' )

GROUP BY cm.managerid, deptname, cd.deptid

HAVING COUNT(complaintid) > 2

)

SELECT \*

FROM comp\_mgr\_dept

WHERE "manager id" = 'E102';

## **TOP COMPLAINT TYPES AND THEIR COUNT**

Display the top 3 frequently raised complaint types for a manager, show the total complaint count for that type and also the complaint count by status (Open, In progress, Resolved)

WITH complaint\_count AS (

SELECT type, COUNT(\*) AS complaint\_type\_count,

DENSE\_RANK() OVER(

ORDER BY COUNT(\*) DESC) AS count\_rank

FROM complaints

WHERE empid IN (

SELECT empid

FROM cc\_manages

WHERE managerid = 'E102'

)

GROUP BY type

), top\_complaints AS (

SELECT cc.type, status, complaint\_type\_count

FROM complaint\_count cc

JOIN complaints c ON cc.type = c.type

WHERE count\_rank <= 3

)

SELECT

type,complaint\_type\_count AS "Total Complaints", o\_c AS "Open complaints",

ip\_c AS "In progress complaints", r\_c AS "Resolved complaints"

FROM top\_complaints

PIVOT (

COUNT(\*) AS c

FOR status IN ( 'Open' AS o, 'In progress' AS ip, 'Resolved' AS r )

)

ORDER BY "Total Complaints" DESC;

## **LATE REIMBURSEMENTS**

Late reimbursement is defined as one where the status is still 'pending' one month later than the expense Date. The manager is interested in knowing the following information regarding the late reimbursements above amount ‘X’ for each of his employee:

1. The count of late reimursements
2. The average number of days by which the reimbursement is late for that employee across all late reimbursements

This insight will help the managers gain insight of whether the reimbursements are being processed on time or not.

WITH late\_reimbursements AS (

SELECT \*

FROM reimbursements

WHERE status = 'Pending'

AND months\_between(sysdate, expensedate) > 1

AND empid IN (

SELECT employeeid

FROM cc\_manages

WHERE managerid = 'E102'

)

), count\_reimbursements AS (

SELECT empid, COUNT(\*) AS total\_late\_reimbursement\_counts

FROM late\_reimbursements

GROUP BY empid

), avg\_latedays AS (

SELECT

empid, round(AVG(months\_between(sysdate, expensedate)),2) AS avg\_no\_latedays\_in\_months

FROM late\_reimbursements

GROUP BY empid

)

SELECT

lr.empid,reimbursementid, expensedate,

amount,total\_late\_reimbursement\_counts, avg\_no\_latedays\_in\_months

FROM late\_reimbursements lr

JOIN count\_reimbursements cr ON lr.empid = cr.empid

JOIN avg\_latedays al ON al.empid = lr.empid

WHERE amount > 100;

## **DETAILS OF CARD ISSUANCE REQUEST**

Due to increasing card demands, TA has created a rule for it's clients regarding card issuance request. The manager is required to provide the count of card categories they need in future such that the card category count in the request must be equivalent to the number of cards of that category expiring only in the second month from the request date otherwise the request will be denied. This query outputs the category and it’s count for the manager to issue his card issuance request.

SELECT category, COUNT(\*) AS count

FROM cards

WHERE edate > TO\_DATE(add\_months(sysdate, 1), 'DD-MON-YYYY')

AND edate < TO\_DATE(add\_months(sysdate, 2), 'DD-MON-YYYY')

AND cardnum IN (

SELECT DISTINCT cardnum

FROM card\_subscriptions cs

JOIN cc\_manages cm ON cm.employeeid = cs.empid

WHERE managerid = 'E102'

)

GROUP BY category;

## **EXPENDITURE MONITORING**

Managers wants to know all the statistics regarding his department’s budget and expenses and needs to know if he has enough budget or not. This query displays the following expenditure data:

1. Manager’s monthly assigned budget
2. Total expenditure incurred till date
3. Percent of assigned budget used
4. The remaing balance
5. A message that tells the manager if he has enough budget or not

The manager can make informed decisions regarding his department’s spending habit using the information generated by this query

WITH maintab AS (

SELECT

ccmgr.managerid, ccm.employeeid, assignedbudget,

ce.cardnum, transactionid, amount, expensedate,description

FROM cc\_manages ccm

JOIN cc\_managers ccmgr ON ccm.managerid = ccmgr.managerid

JOIN card\_subscriptions cs ON ccm.employeeid = cs.empid

JOIN card\_expenses ce ON ce.cardnum = cs.cardnum

WHERE ccmgr.managerid = 'E102'

AND ( expensedate BETWEEN trunc(sysdate, 'MM') AND sysdate )

), totalusage AS (

SELECT

managerid, assignedbudget,

SUM(amount) AS total\_expenditure, round((SUM(amount) / assignedbudget) \* 100,2) AS percent\_used

FROM maintab

GROUP BY managerid, assignedbudget

)

SELECT

managerid, assignedbudget, total\_expenditure,

percent\_used, assignedbudget - total\_expenditure AS remaining\_balance,

CASE

WHEN percent\_used > 90 THEN 'Budget limit'

WHEN percent\_used >= 50 THEN 'Atleast 50% of budget used'

ELSE 'You have enough budget'

END AS message

FROM totalusage;

## **CARD STATISTICS- 1**

For all the cards under a manager, he wants to know the card’s category and type along with the following card statistics –

1. To how many employees the cards have been subscribed to until now
2. The total number of transactions that have been carried out through that card

With this information, the manager can have insight on the usage of the cards in his department and decide whether they need more cards.

WITH main AS (

SELECT c.cardnum, cardtype, category, COUNT(empid) AS noofsubscribes

FROM cards c

LEFT OUTER JOIN card\_subscriptions cs ON c.cardnum = cs.cardnum

WHERE empid IN (

SELECT employeeid

FROM cc\_manages

WHERE managerid = 'E102'

)

GROUP BY c.cardnum, cardtype, category

)

SELECT m.cardnum, cardtype, category, noofsubscribes,

COUNT(transactionid) AS nooftransactions

FROM main m

LEFT OUTER JOIN card\_expenses ce ON ce.cardnum = m.cardnum

GROUP BY m.cardnum, noofsubscribes, cardtype, category;

## **CARD STATISTICS- 2**

For all the cards, under a manager, he needs to:

1. Who is the current user of that card (empid and name)
2. Who was the previous user of the card ( empid and name)

This information will allow the manager to track the employees to whom the cards are presently or were previously assigned.

WITH main AS (

SELECT c.cardnum, cs.empid AS currempid, LEAD(cs.empid)

OVER(PARTITION BY c.cardnum

ORDER BY startdate DESC

) AS previousempid,

RANK()

OVER(PARTITION BY c.cardnum

ORDER BY startdate DESC

) AS rankdate

FROM card\_subscriptions cs

RIGHT OUTER JOIN cards c ON cs.cardnum = c.cardnum

WHERE cs.empid IN (

SELECT employeeid

FROM cc\_manages

WHERE managerid = 'E102'

)

), currempdetails AS (

SELECT cardnum, coalesce(fname, 'Not subscribed currently') AS currentempname,coalesce(currempid, 'Not su bscribed currently') AS currentempid

FROM main m

LEFT OUTER JOIN cc\_employees cce ON cce.empid = m.currempid

WHERE rankdate = 1

), prevempdetails AS (

SELECT cardnum, coalesce(fname, 'Previously not subscribed') AS previousempname, coalesce(previousempid, 'Previously not subscribed') AS previousempid

FROM main m

LEFT OUTER JOIN cc\_employees cce ON cce.empid = m.previousempid

WHERE rankdate = 1

)

SELECT \*

FROM currempdetails ced

NATURAL JOIN prevempdetails;

## **SUPPORT TEAM DETAILS**

The following query extracts the assignedTripActions support team with their manager and support agent contact for an employee. This data can be used by the employees to contact the support team in case of any issue

WITH managers AS (

SELECT ( fname || ' ' || lname ) AS "empname", e.empid

FROM cc\_employees e

WHERE e.empid IN (

SELECT DISTINCT managerid

FROM cc\_manages

)

), support AS (

SELECT m."empname" AS "CC\_Manager\_Name", ( e.fname || ' ' || e.lname ) AS "CC\_Employee\_Name",

e.empid AS "CC\_Employee\_ID", cm.managerid AS "CC\_Manager\_ID", c.name AS "Client\_Company\_Name",

s.name AS "Support\_Team\_Name", ta.empid AS "TA\_Mgr\_ID", tcsa.empid AS "TA\_Csa\_ID",

s.supportteamid AS "Support\_Team\_ID",

COUNT(DISTINCT s.supportteamid) AS "Number of Support Teams"

FROM cc\_manages cm

JOIN cc\_employees e ON cm.employeeid = e.empid

JOIN client\_companies c ON e.companyid = c.companyid

JOIN support\_teams s ON s.companyid = c.companyid

JOIN forms\_managers fm ON fm.supportteamid = s.supportteamid

JOIN forms\_support\_agents fsa ON fsa.supportteamid = s.supportteamid

JOIN ta\_managers ta ON ta.empid = fm.empid

JOIN ta\_customer\_support\_agents tcsa ON tcsa.empid = fsa.empid

JOIN managers m ON m.empid = cm.managerid

WHERE cm.managerid != e.empid

GROUP BY m."empname", e.fname, e.lname, e.empid,

cm.managerid, c.name, s.name, ta.empid, tcsa.empid, s.supportteamid

), supportteam AS (

SELECT s."Support\_Team\_ID", s."Support\_Team\_Name", tm.fname AS "Support Team Manager Contact",

tme.emailid AS "Support Team Manager EmailID",

LISTAGG(DISTINCT tc.fname, ',') WITHIN GROUP(

ORDER BY tc.fname

) AS "Support Team Agent Contact",

LISTAGG(DISTINCT tce.emailid, ',') WITHIN GROUP(

ORDER BY tce.emailid

) AS "Support Team Agent EmailID"

FROM support s

JOIN ta\_employees tm ON s."TA\_Mgr\_ID" = tm.empid

JOIN ta\_employees tc ON s."TA\_Csa\_ID" = tc.empid

JOIN ta\_employees\_emailid tme ON s."TA\_Mgr\_ID" = tme.empid

JOIN ta\_employees\_emailid tce ON s."TA\_Csa\_ID" = tce.empid

GROUP BY s."Support\_Team\_ID", s."Support\_Team\_Name", tm.fname, tme.emailid

), employee AS (

SELECT s."Support\_Team\_ID", s."CC\_Manager\_ID",

LISTAGG(DISTINCT s."CC\_Employee\_ID", ',') WITHIN GROUP(

ORDER BY s."CC\_Employee\_ID"

) AS "CC\_Employee\_ID",

LISTAGG(DISTINCT s."CC\_Employee\_Name", ',') WITHIN GROUP(

ORDER BY s."CC\_Employee\_Name"

) AS "CC\_Employee\_Name"

FROM support s

GROUP BY s."Support\_Team\_ID", s."CC\_Manager\_ID"

)

SELECT DISTINCT

s."CC\_Manager\_ID" AS "Manager ID",

s."CC\_Manager\_Name" AS "Manager Name",

e."CC\_Employee\_ID" AS "Reporting Employee ID",

e."CC\_Employee\_Name" AS "Reporting Employee Name",

s."Support\_Team\_ID" AS "Support Team ID",

s."Support\_Team\_Name" AS "Support Team Name",

st."Support Team Manager Contact" AS "Support Team Manager Contact",

st."Support Team Agent Contact" AS "Support Team Agent Contact",

st."Support Team Manager EmailID" AS "Support Team Manager EmailID",

st."Support Team Agent EmailID" AS "Support Team Agent EmailID"

FROM supportteam st

JOIN employee e ON st."Support\_Team\_ID" = e."Support\_Team\_ID"

JOIN support s ON s."Support\_Team\_ID" = st."Support\_Team\_ID";

## **APPROVED DEPARTMENT CLAIMS**

View a manager and their department level claims (reimbursements) that have been approved per month . This is useful for a manager who wants to track their expenditure even before approving other claims made by employees working under them for that particular month. Or when in retrospect they want to look up on their expenditure made in the past few months.

WITH mngr\_dept AS (

SELECT DISTINCT

managerid,

deptid

FROM cc\_managers cm

JOIN consists c ON cm.managerid = c.empid

), reimb\_emp AS (

SELECT

to\_char(expensedate, 'Month') AS "Month",

managerid AS "Manager ID",

(

CASE

WHEN c.deptid IS NULL THEN

'Total for the Month'

ELSE

c.deptid

END

) AS "Total Expenditure through Reimbursements made per month",

SUM(amount) AS "Amount Reimbursed"

FROM reimbursements r

JOIN consists c ON r.empid = c.empid

JOIN mngr\_dept md ON md.deptid = c.deptid

WHERE status = 'Approved'

GROUP BY

ROLLUP(to\_char(expensedate, 'Month'),

managerid,

c.deptid)

)

SELECT \*

FROM reimb\_emp

WHERE "Month" IS NOT NULL

AND "Manager ID" IS NOT NULL

AND "Total Expenditure through Reimbursements made per month" IS NOT NULL

AND "Manager ID" = 'E102';

WITH mngr\_dept AS (

SELECT DISTINCT managerid, deptid

FROM cc\_managers cm

JOIN consists c ON cm.managerid = c.empid

), reimb\_emp AS (

SELECT to\_char(expensedate, 'Month') AS "Month", managerid AS "Manager ID",

( CASE

WHEN c.deptid IS NULL THEN 'Total for the Month'

ELSE c.deptid

END

) AS "Total Expenditure through Reimbursements made per month",

SUM(amount) AS "Amount Reimbursed"

FROM reimbursements r

JOIN consists c ON r.empid = c.empid

JOIN mngr\_dept md ON md.deptid = c.deptid

WHERE status = 'Approved'

GROUP BY

ROLLUP(to\_char(expensedate, 'Month'), managerid, c.deptid)

)

SELECT \*

FROM reimb\_emp

WHERE

"Month" IS NOT NULL

AND "Manager ID" IS NOT NULL

AND "Total Expenditure through Reimbursements made per month" IS NOT NULL

AND "Manager ID" = 'E102';

## **DISPLAY POLICY AND EMPLOYEE COUNT**

TripActions allows the managers to check the count and the policy which their company has opted for. TripActions in turn will display their current policy tier along with the upgrade prompt incase the limit for that current tier is reached.

WITH policyandemployeecount AS (

SELECT p.policyid AS policyid, COUNT(empid) AS "Count of Employees"

FROM cc\_employees cce

JOIN client\_companies cc ON cc.companyid = cce.companyid

JOIN contracts c ON cc.companyid = c.companyid

JOIN policies p ON p.policyid = c.policyid

WHERE cce.companyid = (

SELECT companyid

FROM cc\_employees cce

JOIN cc\_managers ccm ON cce.empid = ccm.managerid

WHERE empid = 'E104'

)

AND c.contractid IN (

SELECT contractid

FROM active

)

GROUP BY p.policyid

)

SELECT policyid AS "Policy ID", "Count of Employees",

CASE

WHEN ( policyid IN (

SELECT policyid

FROM growth\_plans

) ) THEN

CASE

WHEN ( "Count of Employees" >= 50 )

THEN 'Growth Plan. Please upgrade.' ELSE 'Growth Plan.'

END

WHEN ( policyid IN (

SELECT policyid

FROM professional\_plans

) ) THEN 'Professional Plan'

END AS "Policy"

FROM policyandemployeecount;

## **COMPLAINT RESOLUTION TIME**

Managers would like to know how long does it take (resolution time in days) for different types of complaints to be resolved. The resolution time is defined as 'Undefined' for those new complaint types which have never been resolved.

WITH main AS (

SELECT DISTINCT lower(type) AS type, complaintid, filedate, resolvedate, status, empid

FROM complaints c

JOIN cc\_manages cman ON c.empid = cman.employeeid

WHERE managerid = 'E102'

), resolvedrestime AS (

SELECT DISTINCT type, to\_char(AVG(resolvedate - filedate)) AS avg\_resolution\_time

FROM main

WHERE status = 'Resolved'

GROUP BY type

), unresolvedrestime AS (

SELECT DISTINCT type, 'Undefined' AS avg\_resolution\_time

FROM main

WHERE type NOT IN (

SELECT type

FROM resolvedrestime

)

)

SELECT \*

FROM resolvedrestime

UNION

SELECT \*

FROM unresolvedrestime;

## **LOCATION AND DESCRIPTION SPECIFIC EXPENDITURE SUMMARY**

Manager wants insights regarding location wise (cities) and description wise (shops) expenditure. This information would help the manager gain insight about which locations and shops are most used by the employees. He can provide the HR team with this information to get potential offers at those locations and shops

SELECT coalesce(location, 'All Locations Total') AS locations,

    coalesce(description, 'All Shops Total')  AS shops, SUM(amount) AS total\_amount

FROM (

SELECT DISTINCT cardnum, location, description, amount

        FROM card\_expenses ce

        WHERE cardnum IN (

                SELECT DISTINCT cardnum

                FROM card\_subscriptions

                WHERE empid IN (

                        SELECT empid

                        FROM cc\_manages

                        WHERE managerid = 'E102'

                    )

            )

    )

GROUP BY CUBE(location, description)

ORDER BY total\_amount DESC;

# **Chapter 5: TRIGGERS, PROCEDURES AND ADVANCED SQL**

## **USER VALIDATIONS PACKAGE**

Grouped all the functions related to login validations in a package. This package includes encrypting the password, checking the database and validating certain conditions while signing an employee up and also while logging in.

CREATE OR REPLACE PACKAGE user\_validations AS

FUNCTION signup\_validation(p\_empid varchar2,p\_username varchar2, p\_password varchar2, p\_emailid varchar2, p\_phoneNum varchar2)

RETURN varchar2;

FUNCTION login\_validation(p\_username varchar2, p\_password varchar2)

RETURN varchar2;

FUNCTION encrypt\_password (p\_password varchar2)

RETURN varchar2;

END user\_validations;

/

CREATE OR REPLACE PACKAGE BODY user\_validations AS

FUNCTION signup\_validation(p\_empid varchar2,p\_username varchar2, p\_password varchar2, p\_emailid varchar2, p\_phoneNum varchar2)

RETURN varchar2 AS

v\_count number;

BEGIN

SELECT COUNT(\*)

INTO v\_count

FROM cc\_employees

WHERE empid = p\_empid;

IF (v\_count != 1) THEN

RETURN 'Employee ID does not exists. Please enter a valid ID!';

ELSE

SELECT COUNT(\*)

INTO v\_count

FROM cc\_emp\_logins\_username

WHERE empid = p\_empid;

IF (v\_count != 1) THEN

SELECT COUNT(\*)

INTO v\_count

FROM cc\_emp\_logins\_username

WHERE username = p\_username;

IF (v\_count != 1) THEN

INSERT INTO cc\_emp\_logins\_username (username, empid) VALUES(p\_empid, p\_username);

INSERT INTO cc\_emp\_logins\_password VALUES(p\_username, user\_validations.ENCRYPT\_PASSWORD(p\_password));

SELECT COUNT(\*)

INTO v\_count

FROM cc\_employees\_emails

WHERE empid = p\_empid

AND emailid = p\_emailid;

IF (v\_count = 0) THEN

INSERT INTO cc\_employees\_emails values(p\_empid, p\_emailid);

END IF;

SELECT COUNT(\*)

INTO v\_count

FROM cc\_employees\_phones

WHERE empid = p\_empid

AND phoneNum = p\_phoneNum;

IF (v\_count = 0) THEN

INSERT INTO cc\_employees\_phones values(p\_empid, p\_phoneNum);

END IF;

RETURN 'Registration Done!';

ELSE

RETURN 'This user name has already taken. Please choose a different name!';

END IF;

ELSE

RETURN 'User Account already exists!';

END IF;

END IF;

END signup\_validation;

FUNCTION login\_validation(p\_username varchar2, p\_password varchar2)

RETURN varchar2 AS

v\_count number;

BEGIN

SELECT COUNT(\*)

INTO v\_count

FROM cc\_emp\_logins\_password

WHERE username = p\_username

AND password = user\_validations.ENCRYPT\_PASSWORD(p\_password);

IF (v\_count != 1) THEN

RETURN 'Invalid';

ELSE

SELECT COUNT(\*)

INTO v\_count

FROM cc\_managers

WHERE managerid = (SELECT empid

FROM cc\_emp\_logins\_username

WHERE username = p\_username);

IF (v\_count != 1) THEN

RETURN 'emptype';

ELSE

RETURN 'managertype';

END IF;

END IF;

END login\_validation;

FUNCTION encrypt\_password (p\_password varchar2)

RETURN varchar2 AS

encryption\_key varchar2(21) := '7211AC0DB030620172507';

BEGIN

RETURN DBMS\_CRYPTO.encrypt(UTL\_RAW.CAST\_TO\_RAW(p\_password), 4353, encryption\_key);

END encrypt\_password;

END user\_validations;

/

## **REIMBURSEMENT TRIGGER**

TripActions does not allow an employee to submit more than 5 reimbursements in a particular month. They also have a policy to auto-approve a reimbursement clain if the amount is less than $50. Employees can only claim reimburements for expenses made in the past 90 days. This trigger is made to implement these use-cases.

CREATE OR REPLACE TRIGGER reimbursements\_trigger

BEFORE INSERT

ON REIMBURSEMENTS

FOR EACH ROW

DECLARE

v\_rCount number;

v\_empCount number;

v\_reimbID REIMBURSEMENTS.reimbursementid%type;

BEGIN

SELECT COUNT(\*) INTO v\_empCount FROM CC\_EMPLOYEES WHERE EMPID = :new.empID;

IF (v\_empCount = 0) THEN

raise\_application\_error(-20016, 'Employee not a part of the contract. Please enter a valid empID');

END IF;

SELECT COUNT(\*)

INTO v\_rCount

FROM REIMBURSEMENTS

WHERE empID = :new.empID

AND TO\_CHAR(expenseDate, 'mm-yyyy') = TO\_CHAR(to\_date(:new.expenseDate), 'mm-yyyy')

AND STATUS != 'Declined';

IF (v\_rCount <= 5) THEN

IF (sysdate - to\_date(:new.expensedate)) >= 90 THEN

raise\_application\_error (-20017, 'Claim submission period of 90 days has been paased');

ELSIF (:new.amount <=50) THEN

:new.status := 'Auto Approved';

END IF;

ELSE

raise\_application\_error(-20018, 'Exceeded max submissions for ' || TO\_CHAR(to\_date(:new.expenseDate), 'mm-yyyy') || '(MM/YYYY). Please contact your manager for approval');

END IF;

SELECT LPAD(to\_char(reimbursements\_seq.nextval),4,'R') INTO v\_reimbID FROM dual;

:new.reimbursementid := v\_reimbID;

END;

/

## **REIMBURSEMENT PROCEDURE**

When a manager at a company enters the employee ID of someone they manage, this procedure fetches the 3 oldest reimbursement claims for that employee and sets their status to ‘Review’.

CREATE OR REPLACE PROCEDURE reimbursements\_procedure (p\_empID cc\_employees.empID%type) AS

CURSOR CCLAIM IS

SELECT \*

FROM REIMBURSEMENTS

WHERE EMPID = p\_empID

FOR UPDATE OF STATUS;

v\_idlist sys.ODCIVarchar2List;

BEGIN

WITH REVIEW AS (

SELECT

REIMBURSEMENTID, EMPID,

RANK() OVER (PARTITION BY EMPID ORDER BY EXPENSEDATE ASC) AS PRANK

FROM REIMBURSEMENTS

ORDER BY EMPID)

SELECT REIMBURSEMENTID BULK COLLECT INTO v\_idlist FROM REVIEW

WHERE EMPID = p\_empID and prank < 4;

FOR C IN CCLAIM LOOP

FOR i IN 1 .. v\_idlist.count LOOP

UPDATE REIMBURSEMENTS SET STATUS = 'Review' WHERE REIMBURSEMENTID = v\_idlist(i);

END LOOP;

END LOOP;

END;

/

## **DERIVE END DATE WITH TRIGGER**

The contract has a start date and duration in years, this trigger will calculate the end date of the contract using that data.

CREATE OR REPLACE TRIGGER contract\_enddate\_derive

BEFORE INSERT

ON CONTRACTS

FOR EACH ROW

DECLARE

BEGIN

:new.endDate := TRUNC(ADD\_MONTHS(:new.startDate, :new.duration\*12));

END;

/

## **EMPLOYEE ID SEQUENCE GENERATION TRIGGER**

TripActions wants to auto-generate empID of when inserting data of a new employee. This combination of sequence and trigger helps implement it.

CREATE SEQUENCE cc\_employees\_seq

START WITH 101

MAXVALUE 999;

CREATE OR REPLACE TRIGGER generate\_cc\_empID

BEFORE INSERT

ON CC\_EMPLOYEES

FOR EACH ROW

DECLARE

v\_numEmpID CC\_EMPLOYEES.empID%type;

BEGIN

SELECT LPAD(to\_char(cc\_employees\_seq.nextval),4,'E') INTO v\_numEmpID FROM dual;

:new.empID := v\_numEmpID;

END;

/

## **DEPARTMENT ID SEQUENCE GENERATION TRIGGER**

TripActions wants to auto-generate deptID of when inserting data of a new department. This combination of sequence and trigger helps implement it.

CREATE SEQUENCE cc\_departments\_seq

START WITH 101

MAXVALUE 999;

CREATE OR REPLACE TRIGGER generate\_departmentID

BEFORE INSERT

ON CC\_DEPARTMENTS

FOR EACH ROW

DECLARE

v\_dept\_ID CC\_DEPARTMENTS.deptid%type;

BEGIN

SELECT LPAD(to\_char(cc\_departments\_seq.nextval),4,'D') INTO v\_dept\_ID FROM dual;

:new.deptid := v\_dept\_ID;

END;

/

## **COMPLAINT ID SEQUENCE GENERATION TRIGGER**

TripActions wants to auto-generate complaintID of when inserting data of a new complaint. This combination of sequence and trigger helps implement it.

CREATE SEQUENCE complaints\_seq

START WITH 101

MAXVALUE 999;

CREATE OR REPLACE TRIGGER generate\_complaintID

BEFORE INSERT

ON COMPLAINTS

FOR EACH ROW

DECLARE

v\_comp\_ID complaints.complaintid%type;

BEGIN

SELECT LPAD(to\_char(complaints\_seq.nextval),4,'c') INTO v\_comp\_ID FROM dual;

:new.complaintid := v\_comp\_ID;

END;

/

## **TRANSACTION ID SEQUENCE GENERATION TRIGGER**

TripActions wants to auto-generate trainsactionID of when inserting data of a new transaction. This combination of sequence and trigger helps implement it.

CREATE SEQUENCE transactions\_seq

START WITH 101

MAXVALUE 999;

CREATE OR REPLACE TRIGGER generate\_transactionID

BEFORE INSERT

ON CARD\_EXPENSES

FOR EACH ROW

DECLARE

v\_trans\_ID CARD\_EXPENSES.transactionid%type;

BEGIN

SELECT LPAD(to\_char(transactions\_seq.nextval),4,'T') INTO v\_trans\_ID FROM dual;

:new.transactionid := v\_trans\_ID;

END;

/

## **CONTRACT ID SEQUENCE GENERATION TRIGGER**

TripActions wants to auto-generate contractID of when inserting data of a new contract. This combination of sequence and trigger helps implement it.

CREATE SEQUENCE contracts\_seq

START WITH 101

MAXVALUE 999;

CREATE OR REPLACE TRIGGER generate\_contractID

BEFORE INSERT

ON CONTRACTS

FOR EACH ROW

DECLARE

v\_contractID CONTRACTS.contractid%type;

BEGIN

SELECT LPAD(to\_char(contracts\_seq.nextval),4,'C') INTO v\_contractID FROM dual;

:new.contractid := v\_contractID;

END;

/

## **LOGIN ID SEQUENCE GENERATION TRIGGER**

TripActions wants to auto-generate loginID of when inserting data of a new login. This combination of sequence and trigger helps implement it.

CREATE SEQUENCE LOGINID\_SEQ

START WITH 10001

MAXVALUE 99999;

CREATE OR REPLACE TRIGGER generate\_loginID

BEFORE INSERT

ON cc\_emp\_logins\_username

FOR EACH ROW

DECLARE

v\_loginID cc\_emp\_logins\_username.loginID%type;

BEGIN

SELECT LPAD(to\_char(loginID\_seq.nextval),8,'L000') INTO v\_loginID FROM dual;

:new.loginid := v\_loginID;

END;

/

## **CHECK CARD EXPENSE LIMIT WITH TRIGGER**

TripActions maintains a list of card expenses. Each record can only be added for that current day and the expense for the entire month should not exceed $10000 oon that particular card.

CREATE OR REPLACE TRIGGER card\_expense\_limit\_check

BEFORE INSERT OR UPDATE

ON CARD\_EXPENSES

FOR EACH ROW

DECLARE

v\_maxExpense constant NUMBER(5) := 10000;

v\_totalexpense NUMBER;

BEGIN

IF (:new.expenseDate NOT LIKE SYSDATE) THEN

raise\_application\_error (-20002, 'Expense date different from today''s date');

END IF;

SELECT COALESCE(SUM(amount), 0)

INTO v\_totalexpense

FROM card\_expenses

WHERE cardnum = :new.cardnum

AND EXTRACT(MONTH FROM expenseDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM expenseDate) = EXTRACT(YEAR FROM SYSDATE);

v\_totalexpense := v\_totalexpense + :new.amount;

IF (v\_totalexpense > v\_maxExpense) THEN

raise\_application\_error (-20003, 'Exceeded max limit on the card: ' || :new.cardNum || '. The limit amount is $' || v\_maxExpense);

END IF;

END;

/

## **CONSTRAINING RELATIONSHIP TRIGGER**

TripActions keeps a track of card subscription details of the start and end date of a particular card that is assigned to a particular employee. The card can only be assigned to the employee if the card is a part of a same contract that the company the employee is working in has signed. The start and end date of the card subscription should also not lie outside of the card’s issue date and expiry date.

CREATE OR REPLACE TRIGGER card\_subscriptions\_trigger

BEFORE INSERT

ON card\_subscriptions

FOR EACH ROW

DECLARE

v\_card\_contractid varchar2(4);

v\_card\_idate date;

v\_card\_edate date;

v\_employee\_contractid varchar2(4);

BEGIN

SELECT contractid, idate, edate

INTO v\_card\_contractid, v\_card\_idate, v\_card\_edate

FROM cards

WHERE cardnum = :new.cardnum;

SELECT a.contractid

INTO v\_employee\_contractid

FROM active a

JOIN contracts c ON a.contractid = c.contractid

JOIN client\_companies cc ON c.companyid = cc.companyid

JOIN cc\_employees e ON cc.companyid = e.companyid

WHERE empid = :new.empid;

IF (v\_card\_contractid != v\_employee\_contractid) THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee''s company is not part of the same contract as the card.');

ELSIF (:new.startdate < v\_card\_idate) OR (:new.enddate > v\_card\_edate) THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Card is not active for the entire subscription duration.');

END IF;

:new.validity := :new.enddate - :new.startdate;

END;

/

## **DEPARTMENT COUNT TRIGGER**

TripActions stores the number of employees in each department. This trigger will automatically adjust the number of employees stored in that department.

CREATE OR REPLACE TRIGGER consists\_trigger

BEFORE INSERT ON CONSISTS

FOR EACH ROW

DECLARE

v\_empcount number(4);

BEGIN

SELECT COUNT(empid)

INTO v\_empcount

FROM consists

WHERE deptid = :new.deptid;

v\_empcount := v\_empcount + 1;

UPDATE cc\_departments

SET empcount = v\_empcount

WHERE deptid = :new.deptid;

END;

/

## **GENERATE TOTAL EXPENSE INCURRED**

TripActions automatically adds the sum of the card expenses to the respective department.

create or replace TRIGGER expense\_incurred\_derive

AFTER INSERT OR UPDATE

ON card\_expenses

FOR EACH ROW

DECLARE

v\_deptid cc\_departments.deptid%type;

BEGIN

SELECT DISTINCT deptid

INTO v\_deptid

FROM card\_subscriptions cs

JOIN consists c ON cs.empid = c.empid

WHERE cardnum = :new.cardnum;

UPDATE cc\_departments

SET totalexpenseincurred =

totalexpenseincurred + :new.amount WHERE deptid = v\_deptid;

END;

/

## **MANAGER WILL ALSO MANAGE THEMSELF**

TripActions only allows company to have 2 levels of hierarchy. This implies that a manager will have the managing rights on themselves.

CREATE OR REPLACE TRIGGER cc\_managers\_trigger

BEFORE INSERT ON cc\_managers

FOR EACH ROW

BEGIN

INSERT INTO cc\_manages VALUES(:new.managerid, :new.managerid);

END;

/

## **INDEXING CC\_EMPLOYEES WITH CLIENT COMPANY ID**

TripActions overtime will maintain a huge list of employees and want fast query response time when querying through a companyID in the list of employees.

CREATE INDEX cc\_employees\_companyid\_index

ON cc\_employees (companyid);

## **PARTITION CLIENT COMPANIES BY TYPE**

TripActions have a future plan of providing offers to companies based on the company type. The data analysts who will analyze this will be able to take advantage of partition pruning.

ALTER TABLE client\_companies

MODIFY PARTITION BY LIST (TYPE) (

PARTITION technology VALUES ('Technology'),

PARTITION online\_startups VALUES ('Fintech'),

PARTITION miscellaneous VALUES ('Mixed')

);

# **CHAPTER 6: FRONT-END**

Link: [**Home Page - TRIP ACTIONS (ec2-34-223-254-213.us-west-2.compute.amazonaws.com)**](http://ec2-34-223-254-213.us-west-2.compute.amazonaws.com/TripActions/)

## **HOME PAGE**

This page displays the home screen for Trip Actions.

Graphical user interface, application, Teams

Description automatically generated

## **USER LOGIN PAGE**

This is the User login page.

Graphical user interface, application

Description automatically generated

## **SIGN UP PAGE**

This is the signup form for new employee signup

Graphical user interface, application

Description automatically generated

## **MANAGER’S HOME PAGE VIEW**

This is the Manager's view of the Trip Actions home page

Graphical user interface

Description automatically generated

## **EMPLOYEES VIEW FOR MANAGER**

On this page manager can either add a new employee under him and view his current employees

Graphical user interface, text, application

Description automatically generated

## **NEW EMPLOYEE ADDITION IN EMPLOYEES VIEW**

Here the manager can add a new employee

Graphical user interface, application

Description automatically generated

## **VIEW EMPLOYEES IN EMPLOYEES VIEW**

Here the manager can view his current employees and perform CRUD operations

Graphical user interface, text, application

Description automatically generated

## **REIMBURSEMENTS PAGE**

This page allows multiple options for manager about employee claims

Graphical user interface, text, website

Description automatically generated

## **NEW REIMBURSEMENT FORM IN REIMBURSEMENTS**

This page allows manager to insert new reimbursemenets

Graphical user interface

Description automatically generated

## **REIMBURSEMENT CLAIMS ADDITION, UPDATION AND DELETION IN REIMBURSEMENTS**

Here the manager can insert update and delete the claims of employees

Graphical user interface, text

Description automatically generated with medium confidence

,

## **REIMBURSEMENT CLAIMS APPROVAL AND REJECTION IN REIMBURSEMENTS**

Here the manager can approve and reject the claims submitted by his reportees

Graphical user interface, application, Word

Description automatically generated

## **EXPENSES**

Screenshot of query "Expenditure Monitoring" . This page shows manager their assigned budget.

Graphical user interface, application

Description automatically generated

## **ASSIGN CARDS**

Managers can assign cards to their employees. This page allows manager to insert, update or delete the card details for his/her reportees

Graphical user interface, text, application

Description automatically generated

## **COMPLAINTS**

This page shows the menu for Complaints

Graphical user interface, website

Description automatically generated

## **NEW COMPLAINTS FORM**

This page allows managers to raise new complaints

Graphical user interface

Description automatically generated

## **COMPLAINTS MANAGEMENT IN COMPLAINTS**

This page allows the manager to perform CRUD operations on existing complaints

Graphical user interface, application

Description automatically generated

## **TOP COMPLAINT TYPES AND COUNT**

Screenshot of query no. "Top 3 complaint types and their count" .Top 3 types of complaints received by manager of his/her employees using WITH TIES

A computer screen capture

Description automatically generated with medium confidence

## **SUPPORT TEAM INFORMATION**

Screenshot of query no. "Support team details". This page shows the manager his/her reportees and the support contact from TripActions

Graphical user interface, text, application

Description automatically generated

## **REIMBURSEMENTS VIEW FOR EMPLOYEES**

This page shows the Employee side of reimbursement menu

Graphical user interface, application

Description automatically generated

## **CARD DETAILS INFORMATION PAGE FOR EMPLOYEES**

The card details of cards assigned to the employee

Text

Description automatically generated with low confidence

## **COMPLAINTS VIEW FOR EMPLOYEE’S LOGIN**

This page shows the employee side of complaints menu

Graphical user interface, text, website

Description automatically generated

## **SUPPORT TEAM INFORMATION FOR EMPLOYEE’S LOGIN**

This page shows the TA support details for the employee

Graphical user interface, text, application

Description automatically generated

## UI REPRESENTATION OF COMPLEX QUERIES IN STATISTICS

The statistics page shows the Complex queries

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# **CHAPTER 7: IMPLEMENTATION PLAN**

## **PLAN OF ACTION:**

Timeline

Description automatically generated

## **EXPENSES ALONG WITH DESCRIPTION FOR THE IMPLEMENTATION OF FULL PROJECT**

|  |  |  |
| --- | --- | --- |
| **EXPENSES** | **ESTIMATED COST** | **NOTES** |
| Cloud service Provider - AWS | $5500 | AWS is used to host the TripAction’s application on cloud service |
| Oracle 19c Enterprise Edition | $47500 | TripAction’s relational database model |
| Oracle Golden Gate | $8750 | This is used for real time disaster recovery management of Production system |
| Database Administrators | $15000 - $20000 | Amount required to hire a DBA to form the team for this project |
| Data Engineers | $20000 - $25000 | Amount required to hire a data engineer to manage the design and test the database |
| Application Developers | $30000 - $40000 | Amount required to hire two developers to work on implementation of the application of this project |
| Software Test Engineers | $20000 - $30000 | Amount required to hire two developers to work on testing of the application of this project |
| Trainings | $10000 - $15000 | Amount required to train the employees to enhance their skill set |

# **APPENDIX A**

* Learnt the requirements of the clients to implement their new products.
* Created the conceptual design by creating the Entity Relationship diagram to understand the entities and their relations.
* Used the logical designing to convert the ER diagram to relational tables.
* Used the normalization techniques to remove the redundancies and make the design more practical to create the physical design.
* Created the new tables in the database to understand the flow of actions.
* Implemented the client requirements by writing the SQL queries to fetch the correct results.
* Used Triggers and procedures to reduce the manual query running.
* Used sequences to auto generate the ID's of the employees.
* Learnt to create the front end with the help of asp.net, C Sharp, Web forms and HTML.
* Displayed the output of the SQL queries on the front end.
* Learnt to create a thorough report for the implementation strategy and planning.
* Researched about project estimations for our report.