# Table of Contents

# 1. Data Types

## 1.1 User

| Field | Data Type | Nullable |
|---|---|---|
| Employee ID | Integer | Not Null |
| First Name | String | Not Null |
| Last Name | String | Not Null |
| Audit View Flag | String | Null |
| Last 4 SSN | String | Not Null |
| District Number | Integer | Not Null |

## 1.2 Audit Report

| Field | Data Type | Nullable |
|---|---|---|
| Time Stamp | DateTime | Not Null |
| Employee ID | Integer | Not Null |
| Report Name | String | Not Null |
| First Name | String | Not Null |
| Last Name | String | Not Null |

## 1.3 Holiday

| Field | Data Type | Nullable |
|---|---|---|
| Employee ID | Integer | Not Null |
| Holiday Date | DateTime | Not Null |
| Holiday Name | String | Not Null |

## 1.4 District

| Field | Data Type | Nullable |
|---|---|---|
| District Number | String | Not Null |

## 1.5 Sales Data

| Field | Data Type | Nullable |
|---|---|---|
| Sale Date | Datetime | Not Null |
| Product Quantity | Integer | Null |

## 1.6 User-Sales Data

| Field | Data Type | Nullable |
|---|---|---|
| Employee ID | Integer | Null |
| Sale Date | DateTime | Null |

## 1.7 Sales Data-Holiday

| Field | Data Type | Nullable |
|---|---|---|
| Holiday Date | DateTime | Null |

| | | |
|---|---|---|
| Sale Date | DateTime | Null |

### 1.8 Sales-Data Reports

| Field | Data Type | Nullable |
|---|---|---|
| Sale Date | Datetime | Null |
| Report Name | String | Null |

### 1.9 Reports

| Field | Data Type | Nullable |
|---|---|---|
| Sale Date | Datetime | Null |
| Report Name | String | Null |

### 1.10 Product

| Field | Data Type | Nullable |
|---|---|---|
| PID | Integer | Not Null |
| Product Name | String | Not Null |
| Retail Price | Decimal | Not Null |
| Manufacturer Name | String | Not Null |
| Store Number | Integer | Not Null |
| Category Name | String | Not Null |

### 1.11 Category

| Field | Data Type | Nullable |
|---|---|---|
| Category Name | String | Not Null |

### 1.12 Product-Category

| Field | Data Type | Nullable |
|---|---|---|
| PID | Integer | Null |
| Category Name | String | Not Null |

### 1.13 Discount

| Field | Data Type | Nullable |
|---|---|---|
| PID | Integer | Not Null |
| Discount Date | DateTime | Not Null |
| Discount Price | Decimal | Not Null |

### 1.14 Manufacturer

| Field | Data Type | Nullable |
|---|---|---|
| Manufacturer Name | String | Not Null |

### 1.15 Store

| Field | Data Type | Nullable |
|---|---|---|
| Store Number | Integer | Not Null |

| | | |
|---|---|---|
| Phone Number | String | Not Null |
| City Name | String | Not Null |
| District Number | Integer | Not Null |

### 1.16 City

| Field | Data Type | Nullable |
|---|---|---|
| City Name | String | Not Null |
| State | String | Not Null |
| Population | Integer | Not Null |

## 2. Business Logic Constraints

User

- New users have to be manually configured by the database administrator to set up their account
- Use's district assignment will be manually configured by the database administrator

Product

- All products are available and sold at all stores
- The price of a product is same across all stores
- Each product is assigned to one manufacturer
- Each product belongs to one or more category

Holiday

- If a date is already in database as a holiday, a user can't add it any more
- Only users are assigned to all districts can add holidays

Main Menu

- The following statistics should be shown for all data in the data warehouse regardless of the user's access level: a welcome message with the user's full name, count of stores, cities, districts manufacturers, products and holidays
- Reports are available based on report type and user's access level:
    - General reports are available to all users
    - District reports should return data based on the district(s) assigned to the logged-in user
    - Corporate reports are only available to users who have been granted access to all districts

Audit Log

- Only users with audit log permission can view audit log
- Users who are allowed to view the audit log within the data warehouse UI will have a special flag set on their account

## 3. Task Decomposition with Abstract Code

### 3.1 Login

Task Decomp

- The login form is an input to the log in task, the login task reads from the database to confirm that the combination of email address and password is correct and if it logs in the user.
- Purpose: Allow a user to securely log into the system.
- Type: Write operation.
- Access Control: Requires user credentials (username and password).
- Consistency: Strong consistency is required.
- Order: Single step operation.
- Frequency: On-demand (whenever a user wants to log in).
- Schema Requirements: Authentication and session management constructs

*Subtasks*

1. **Authenticate User**
    - **Description**: Verify user credentials against the stored data.
    - **Access Control**: Open to all users attempting to log in.
    - **Consistency Requirements**: Strong consistency; immediate feedback is required.
    - **Schema**:
        - `Username`: String.
        - `PasswordHash`: String.
    - **Read-Only**: No.
2. **Create User Session**
    - **Description**: Generate a session token upon successful authentication.
    - **Access Control**: Restricted to authenticated users.
    - **Consistency Requirements**: Strong consistency; session must be created immediately.
    - **Schema**:
        - `Login`: Unique identifier for the user.

Abstract Code

```
Function authenticateUser(username, password)
  If isValidUsername(username) AND isValidPassword(password) THEN
    user <- getUser(username)
    IF user EXISTS THEN
      IF verifyPassword(user.PasswordHash, password) THEN
        displayWelcomeMessage(user)
        createUserSession(user)
        return True
      ELSE
        displayErrorMessage("Invalid password.")
        return False
      END IF
    ELSE
      displayErrorMessage("Username not found.")
      return False
```

```
    END IF
  ELSE
    displayErrorMessage("Invalid input format.")
    return False
  END IF
END Function

Function isValidUsername(username)
  return (username IS NOT NULL AND username IS String)
END Function

Function isValidPassword(password)
  return (password IS NOT NULL AND password IS String)
END Function

Function getUser(username)
  // Query database to fetch user by username
  return database.query("SELECT * FROM Users WHERE Username = username")
END Function

Function verifyPassword(storedHash, inputPassword)
  // Verify inputPassword against storedHash
  return hashFunction(inputPassword) == storedHash
END Function

Function displayWelcomeMessage(user)
  print("Welcome, " + user.FirstName + " " + user.LastName + "!")
END Function

Function displayErrorMessage(message)
  print("Error: " + message)
END Function

Function createUserSession(user)
  sessionToken <- generateSessionToken()
  saveSession(user.Username, sessionToken)
  print("Session created for user: " + user.Username)
END Function

Function generateSessionToken()
  // Generate a unique session token
  return UUID()
END Function

Function saveSession(username, sessionToken)
  // Save session token in the database for the user
```

```
    database.execute("INSERT INTO Sessions (Username, SessionToken) VALUES (username, sessionToken)")
END Function
```

## 3.2 Main Menu

Task Decomp

*Overview*

- **Purpose**: Provide a user interface for navigation to various sections of the application, including viewing statistics and listing reports.
- **Type**: Read-only operation.
- **Access Control**: Enabled by the user's login.
- **Consistency**: Eventual consistency is acceptable.
- **Order**: Single step operation.
- **Frequency**: On-demand (whenever a user accesses the main menu).
- **Schema Requirements**: User role and permissions constructs.

*Subtasks*

1. **Retrieve User Role and Permissions**
    - **Description**: Fetch the user's role and permissions to display relevant menu options.
    - **Access Control**: Restricted to authenticated users.
    - **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
    - **Schema**:
        - `Login`: Unique identifier for the user.
    - **Read-Only**: Yes.
2. **Generate Menu Options**
    - **Description**: Create the menu interface based on the user's role and permissions.
    - **Access Control**: Restricted to authenticated users.
    - **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
    - **Schema**:
        - View Statistics/List of Reports: List of strings.
    - **Read-Only**: Yes.

Abstract Code

```
Function mainMenu(user)
    rolesPermissions <- retrieveUserRoleAndPermissions(user.Username)
    menuOptions <- generateMenuOptions(rolesPermissions)
    displayMenu(menuOptions)
    choice <- getUserInput()
    navigateToTask(choice)
END Function
```

```
Function retrieveUserRoleAndPermissions(username)
   // Fetch user role and permissions from the database
   return database.query("SELECT Role, Permissions FROM UserRoles WHERE Username =
username")
END Function

Function generateMenuOptions(rolesPermissions)
   menuOptions <- []
   IF rolesPermissions.Role == "Admin" THEN
      menuOptions.append("1. View General Reports")
      menuOptions.append("2. View District Reports")
      menuOptions.append("3. View Corporate Reports")
      menuOptions.append("4. Manage Holidays")
      menuOptions.append("5. View Audit Log")
   ELSE IF rolesPermissions.Role == "User" THEN
      menuOptions.append("1. View General Reports")
      menuOptions.append("2. View District Reports")
   END IF
   menuOptions.append("0. Exit")
   return menuOptions
END Function

Function displayMenu(menuOptions)
   print("Main Menu")
   FOR option IN menuOptions DO
      print(option)
   END FOR
END Function

Function getUserInput()
   input <- readLine()
   return input
END Function

Function navigateToTask(choice, user)
   SWITCH choice DO
      CASE "1":
         viewGeneralReports()
      CASE "2":
         viewDistrictReports()
      CASE "3":
         viewCorporateReports()
      CASE "4":
         manageHolidays()
      CASE "5":
         viewAuditLog()
```

```
        CASE "0":
            exitProgram()
        DEFAULT:
            displayErrorMessage("Invalid choice. Please select a valid option.")
            mainMenu(user) // Re-display menu after invalid choice
    END SWITCH
END Function

Function viewGeneralReports()
    print("Generating General Reports...")
    // Implementation for viewing general reports
END Function

Function viewDistrictReports()
    print("Generating District Reports...")
    // Implementation for viewing district reports
END Function

Function viewCorporateReports()
    print("Generating Corporate Reports...")
    // Implementation for viewing corporate reports
END Function

Function manageHolidays()
    print("Managing Holidays...")
    // Implementation for managing holidays
END Function

Function viewAuditLog()
    print("Viewing Audit Log...")
    // Implementation for viewing audit log
END Function

Function exitProgram()
    print("Exiting the program. Goodbye!")
    exit()
END Function

Function displayErrorMessage(message)
    print("Error: " + message)
END Function
```

## 3.3 View Report

Task Decomp

*Overview*

- **Purpose**: Allow a user to view various reports.

- **Type**: Read-only operation.
- **Access Control**: Enabled by the user's login and appropriate permissions.
- **Consistency**: Eventual consistency is acceptable.
- **Order**: Can be done in any order.
- **Frequency**: On-demand (whenever a user views a report).
- **Schema Requirements**: Report data constructs.

*Subtasks*

1. **Fetch Report Metadata**
    - **Description**: Retrieve metadata about available reports.
    - **Access Control**: Restricted to authenticated users with appropriate permissions.
    - **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
    - **Schema**:
        - View Reports: Unique identifier for the report.
    - **Read-Only**: Yes.
2. **Retrieve Report Data**

- **Description**: Fetch the actual data for the selected report.
- **Access Control**: Restricted to authenticated users with appropriate permissions.
- **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
- **Schema**:
    - `ReportID`: Unique identifier for the report.
    - `Data`: JSON or other structured format.
- **Read-Only**: Yes.

Abstract Code

```
Function viewReports(user)
   reportMetadata <- fetchReportMetadata(user.Username)
   displayReportOptions(reportMetadata)
   reportChoice <- getUserInput()
   reportData <- retrieveReportData(reportChoice, user.Username, user.District)
   displayReportData(reportData)
END Function

Function fetchReportMetadata(username)
   // Ensure the user has permissions to view reports
   permissions <- getUserPermissions(username)
   IF permissions.contains("VIEW_REPORTS") THEN
      // Fetch metadata about available reports
      return database.query("SELECT ReportID, ReportName FROM Reports WHERE
Permissions <= permissions")
   ELSE
```

```
    displayErrorMessage("You do not have permission to view reports.")
    return []
  END IF
END Function

Function getUserPermissions(username)
  // Fetch user permissions from the database
  return database.query("SELECT Permissions FROM UserRoles WHERE Username =
username")
END Function

Function displayReportOptions(reportMetadata)
  print("Available Reports:")
  FOR report IN reportMetadata DO
    print(report.ReportID + ". " + report.ReportName)
  END FOR
END Function

Function getUserInput()
  input <- readLine()
  return input
END Function

Function retrieveReportData(reportID, username)
  // Ensure the user has permissions to view the selected report
  permissions <- getUserPermissions(username)
  IF permissions.contains("VIEW_REPORT_" + reportID) THEN
    // Fetch the actual report data
    IF reportID.Type IS “District Report” THEN
      return database.query("SELECT Data FROM ReportData WHERE ReportID = reportID
      AND ReportDisctrict = user.District”)
    ELSE
      return database.query("SELECT Data FROM ReportData WHERE ReportID = reportID")
    END IF
  ELSE
    displayErrorMessage("You do not have permission to view this report.")
    return null
  END IF
END Function

Function displayReportData(reportData)
  IF reportData IS NOT null THEN
    print("Report Data:")
    print(reportData)
  ELSE
    displayErrorMessage("Failed to retrieve report data.")
  END IF
```

```
END Function

Function displayErrorMessage(message)
  print("Error: " + message)
END Function
```

## 3.4 Holiday Form

Task Decomp

*Overview*

- **Purpose**: Allow a user to request and manage holiday leave.
- **Type**: Write and read operations.
- **Access Control**: Enabled by the user's login.
- **Consistency**: Strong consistency for write operations, eventual consistency for read operations.
- **Order**: Can be done in any order.
- **Frequency**: On-demand (whenever a user manages holiday leave).
- **Schema Requirements**: Holiday request and approval constructs.

*Subtasks*

1. **Submit Holiday Request**
    - **Description**: Allow a user to submit a new holiday request.
    - **Access Control**: Restricted to authenticated users.
    - **Consistency Requirements**: Strong consistency; request must be submitted immediately.
    - **Schema**:
        - `Holiday`: Unique identifier for the user.
    - **Read-Only**: No.
2. **View Holiday Requests**
    - **Description**: Allow a user to view their submitted holiday requests.
    - **Access Control**: Restricted to authenticated users.
    - **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
    - **Schema**:
        - `Holiday`: Unique identifier for the user.
    - **Read-Only**: Yes.

Abstract Code

```
Function manageHolidays(user)
  While True DO
    displayHolidayMenu()
    choice <- getUserInput()
    SWITCH choice DO
      CASE "1":
        submitHolidayRequest(user)
```

```
        CASE "2":
          viewHolidayRequests(user)
        CASE "0":
          exitHolidayManagement()
        DEFAULT:
          displayErrorMessage("Invalid choice. Please select a valid option.")
      END SWITCH
    END WHILE
END Function

Function displayHolidayMenu()
    print("Holiday Management Menu")
    print("1. Submit Holiday Request")
    print("2. View Holiday Requests")
    print("0. Exit")
END Function

Function submitHolidayRequest(user)
    IF isAuthenticated(user) THEN
        holidayDate <- getHolidayDateInput()
        holidayReason <- getHolidayReasonInput()
        IF isValidHolidayDate(holidayDate) AND isValidHolidayReason(holidayReason) THEN
            saveHolidayRequest(user, holidayDate, holidayReason)
            print("Holiday request submitted successfully.")
        ELSE
            displayErrorMessage("Invalid holiday date or reason.")
        END IF
    ELSE
        displayErrorMessage("User is not authenticated.")
    END IF
END Function

Function isAuthenticated(user)
    return user IS NOT NULL
END Function

Function getHolidayDateInput()
    print("Enter holiday date (YYYY-MM-DD):")
    return readLine()
END Function

Function getHolidayReasonInput()
    print("Enter holiday reason:")
    return readLine()
END Function

Function isValidHolidayDate(holidayDate)
```

```
    // Validate the date format and check if it is a future date
    return isValidDateFormat(holidayDate) AND isFutureDate(holidayDate)
END Function

Function isValidHolidayReason(holidayReason)
    // Validate the reason is not empty
    return holidayReason IS NOT NULL AND holidayReason != ""
END Function

Function saveHolidayRequest(user, holidayDate, holidayReason)
    // Save the holiday request to the database with strong consistency
    database.execute("INSERT INTO HolidayRequests (UserID, HolidayDate, HolidayReason)
VALUES (user.UserID, holidayDate, holidayReason)")
END Function

Function viewHolidayRequests(user)
    IF isAuthenticated(user) THEN
        holidayRequests <- fetchHolidayRequests(user)
        displayHolidayRequests(holidayRequests)
    ELSE
        displayErrorMessage("User is not authenticated.")
    END IF
END Function

Function fetchHolidayRequests(user)
    // Fetch holiday requests from the database with eventual consistency
    return database.query("SELECT HolidayDate, HolidayReason, Status FROM
HolidayRequests WHERE UserID = user.UserID")
END Function

Function displayHolidayRequests(holidayRequests)
    print("Your Holiday Requests:")
    FOR request IN holidayRequests DO
        print("Date: " + request.HolidayDate + ", Reason: " + request.HolidayReason + ", Status:
" + request.Status)
    END FOR
END Function

Function exitHolidayManagement()
    print("Exiting Holiday Management.")
    return
END Function

Function displayErrorMessage(message)
    print("Error: " + message)
END Function
```

```
Function isValidDateFormat(date)
   // Validate the date format (YYYY-MM-DD)
   return date.matches("\\d{4}-\\d{2}-\\d{2}")
END Function

Function isFutureDate(date)
   // Check if the date is in the future
   today <- getCurrentDate()
   return date > today
END Function

Function getCurrentDate()
   // Get the current date
   return currentDate()
END Function
```

## 3.5 View Audit Logs

Task Decomp

### *Overview*

- **Purpose**: Allow a user to view system audit logs for security and compliance purposes.
- **Type**: Read-only operation.
- **Access Control**: Enabled by the user's login and appropriate permissions.
- **Consistency**: Eventual consistency is acceptable.
- **Order**: Can be done in any order.
- **Frequency**: On-demand (whenever a user views audit logs).
- **Schema Requirements**: Audit log constructs.

### *Subtasks*

1. **Fetch Audit Log Metadata**
    - **Description**: Retrieve metadata about available audit logs.
    - **Access Control**: Restricted to authenticated users with appropriate permissions.
    - **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
    - **Schema**:
        - `LogID`: Unique identifier for the log entry.
    - **Read-Only**: Yes.
2. **Retrieve Audit Log Details**
    - **Description**: Fetch the details of the selected audit log entry.
    - **Access Control**: Restricted to authenticated users with appropriate permissions.
    - **Consistency Requirements**: Eventual consistency; slight delays are acceptable.
    - **Schema**:

        - `LogID`: Unique identifier for the log entry.
    - **Read-Only**: Yes.

Abstract Code

```
Function viewAuditLogs(user)
  While True DO
    auditLogMetadata <- fetchAuditLogMetadata(user.Username)
    displayAuditLogOptions(auditLogMetadata)
    logChoice <- getUserInput()
    IF logChoice == "0" THEN
      exitAuditLogView()
    ELSE
      auditLogDetails <- retrieveAuditLogDetails(logChoice, user.Username)
      displayAuditLogDetails(auditLogDetails)
    END IF
  END WHILE
END Function

Function fetchAuditLogMetadata(username)
  // Ensure the user has permissions to view audit logs
  permissions <- getUserPermissions(username)
  IF permissions.contains("VIEW_AUDIT_LOGS") THEN
    // Fetch metadata about available audit logs
    return database.query("SELECT LogID, LogSummary FROM AuditLogs ORDER BY
Timestamp DESC LIMIT 100")
  ELSE
    displayErrorMessage("You do not have permission to view audit logs.")
    return []
  END IF
END Function

Function getUserPermissions(username)
  // Fetch user permissions from the database
  return database.query("SELECT Permissions FROM UserRoles WHERE Username =
username")
END Function

Function displayAuditLogOptions(auditLogMetadata)
  print("Available Audit Logs:")
  FOR log IN auditLogMetadata DO
    print(log.LogID + ". " + log.LogSummary)
  END FOR
  print("0. Exit")
END Function
```

```
Function getUserInput()
  input <- readLine()
  return input
END Function

Function retrieveAuditLogDetails(logID, username)
  // Ensure the user has permissions to view the selected audit log entry
  permissions <- getUserPermissions(username)
  IF permissions.contains("VIEW_AUDIT_LOGS") THEN
    // Fetch the details of the selected audit log entry
    return database.query("SELECT * FROM AuditLogs WHERE LogID = logID")
  ELSE
    displayErrorMessage("You do not have permission to view this audit log.")
    return null
  END IF
END Function

Function displayAuditLogDetails(auditLogDetails)
  IF auditLogDetails IS NOT null THEN
    print("Audit Log Details:")
    print("LogID: " + auditLogDetails.LogID)
    print("Timestamp: " + auditLogDetails.Timestamp)
    print("User: " + auditLogDetails.User)
    print("Action: " + auditLogDetails.Action)
    print("Details: " + auditLogDetails.Details)
  ELSE
    displayErrorMessage("Failed to retrieve audit log details.")
  END IF
END Function

Function exitAuditLogView()
  print("Exiting Audit Log View.")
  return
END Function

Function displayErrorMessage(message)
  print("Error: " + message)
END Function
```