Table of Contents

[Executive Reporting Tool – Requirements Document 1](#_Toc210753821)

[1. Purpose 1](#_Toc210753822)

[2. Functional Requirements 1](#_Toc210753823)

[3. Data Requirements 2](#_Toc210753824)

[4. Forms & User Interface 2](#_Toc210753825)

[5. Reporting Requirements 2](#_Toc210753826)

[6. Database Design 2](#_Toc210753827)

# Executive Reporting Tool – Requirements Document

## 1. Purpose

The Executive Reporting Tool will be built in Microsoft Office (Access/Excel with forms, charts, and reports) to: Track the remediation lifecycle of incidents across multiple phases. Provide color-coded visual cues for status at each phase. Allow executives to quickly view progress, ownership, and comments. Maintain a change history log for auditability.

## 2. Functional Requirements

2.1 Incident Lifecycle Phases The tool must track these phases: BRD (Business Requirements Document) FRD (Functional Requirements Document) JIRA Adds Technical Estimation Build UAT (User Acceptance Testing) Production Release Incident Closure

2.2 Status Color Coding Each Start Date / End Date field will show a status indicator color: ✅ Green → On Track ⚠️ Orange → At Risk ⛔ Red → Overdue ⚪ Gray → Not Started 🔵 Blue → Completed

2.3 Comments & Ownership For each phase: Owner field – assigned person/team. Owner Comment field – ongoing remarks. Final Comment field – phase closure notes.

## 3. Data Requirements

3.1 Ticket Information Each incident ticket will include: Incident Ticket Number (Primary Key) Incident Name Incident Type (Bug, Enhancement, Critical, etc.) Impacted Applications Impacted Teams Owner of each phase

3.2 Database of Changes Every update (dates, status, owner, comments) must be logged in a Change History table. If no updates occur during a reporting cycle → mark as “No Changes.”

## 4. Forms & User Interface

4.1 Ticket Entry / Edit Form Includes: Ticket metadata (number, name, type, applications, teams). Phase details (start date, end date, status color, owner, comments). Automatic status color calculation based on business rules. Change Tracking Integration → each update logged automatically.

4.2 Executive Dashboard High-level dashboard with: Pie chart of statuses (On Track, At Risk, Overdue, Completed). Table of at-risk/overdue tickets. Budget/effort tracking charts (if required). Filters for: Incident Type, Application, Team, Status.

## 5. Reporting Requirements

The tool must generate status reports showing: Ticket lifecycle with colored phase indicators. Per-phase ownership, comments, and final notes. Export to PDF/Excel for distribution. Filtering by Incident Type, Application, Team, Status. 6. Notifications & Approvals (Optional Advanced Feature) If start/end dates are updated → trigger a Change Control Request. Change requests must be approved by application owners (with email notifications). Once approved → send summary email with new dates to a distribution group.

## 6. Database Design

1. Database Tables

1.1 IncidentTickets

TicketID (PK, AutoNumber or Text for external reference)

TicketName (Text) TicketType (Lookup: Bug, Enhancement, Critical, etc.) ImpactedApplications (Multi-select lookup from Applications table)

ImpactedTeams (Multi-select lookup from Teams table)

Status (Calculated from phases: On Track, At Risk, Overdue, etc.)

CreatedDate (Date/Time)

LastUpdated (Date/Time)

ChangeFlag (Yes/No → "No Changes" if False)

1.2 IncidentPhases

PhaseID (PK) TicketID (FK → IncidentTickets.TicketID)

PhaseName (Lookup: BRD, FRD, JIRA Adds, Tech Estimation, Build, UAT, Prod Release, Incident Closure)

StartDate (Date/Time)

EndDate (Date/Time)

StatusColor (Calculated from business rules)

Owner (FK → Users.UserID)

OwnerComment (Memo/Long Text)

FinalComment (Memo/Long Text)

1.3 ChangeHistory

ChangeID (PK)

TicketID (FK → IncidentTickets.TicketID)

PhaseID (FK → IncidentPhases.PhaseID)

ChangedField (Text)

OldValue (Text)

NewValue (Text)

ChangeReason (Text)

ChangedBy (FK → Users.UserID)

ChangeDate (Date/Time)

Approved (Yes/No)

ApprovedBy (FK → Users.UserID)

1.4 Applications

AppID (PK)

AppName (Text)

OwnerUserID (FK → Users.UserID)

1.5 Teams

TeamID (PK)

TeamName (Text)

OwnerUserID (FK → Users.UserID)

1.6 TeamApplications (for many-to-many relationships)

TeamAppID (PK)

TeamID (FK → Teams.TeamID)

AppID (FK → Applications.AppID)

ImpactType (Lookup: Code Change, Testing Only)

BudgetAllocated (Currency)

1.7 Users

UserID (PK)

FullName (Text)

Email (Text)

Role (Lookup: Executive, Owner, Team Lead, Analyst, etc.)

2. Relationships

IncidentTickets 1:M → IncidentPhases IncidentTickets 1:M → ChangeHistory Teams M:N Applications via TeamApplications Applications 1:M → IncidentTickets (ImpactedApplications) Teams 1:M → IncidentTickets (ImpactedTeams) Users 1:M → IncidentPhases (Owner) Users 1:M → ChangeHistory (ChangedBy, ApprovedBy)

3. Queries

qTicketStatusSummary – calculates overall ticket status (based on phase statuses). qPhaseOverdue – finds phases past EndDate but not marked completed. qChangeApprovalsPending – lists change requests awaiting approval.

qBudgetByAppTeam – aggregates budget by team/application and ticket. qExecutiveDashboard – combines ticket + phase + status into summary view.

4. Forms

frmTicketEntry Ticket metadata + subform for phases. Dropdowns for teams & apps. “No Changes” checkbox auto-set when nothing updates.

subfrmPhases Start/End dates, auto-color status box, owner, comments. frmChangeControl Shows list of pending changes. Field for reason & approval workflow. Button → Send Outlook email to approvers.

frmTeamApplications Manage team/application relationships. Track impact type + budget allocations.

frmExecutiveDashboard Pie chart for status colors. Budget overview by team/app. Filter options (Ticket Type, Team, App, Status).

5. Reports

rptTicketLifecycle – full ticket lifecycle with status colors.

rptExecutiveSummary – pie chart + overdue tickets.

rptBudgetImpact – by team, app, and overall program.

rptChangeHistory – audit log for changes.

6. Notifications

(Outlook Integration) On date change → trigger new ChangeHistory entry.

Form button: Submit Change Request. Auto-generate Outlook email → send to impacted app owners. On approval → update ChangeHistory → send notification to distribution list with updated dates. This design is end-to-end build-ready in Microsoft Access.

## Adding User Login & Security

1. User Login Form Create a User’s table (we already designed it: Users.UserID, FullName, Email, Role, PasswordHash).
2. Build a Login Form (frmLogin): Textboxes: Username/Email, Password. Button: “Login.” On Click → validate against Users table. Store logged-in user info in TempVars or global variables for the session. Tip: Store passwords as hashed strings (not plain text) using VBA (e.g., MD5, SHA256 libraries).
3. Role-Based Security Add a Role field (Admin, Executive, Owner, Analyst). On successful login, redirect users to: frmExecutiveDashboard → Executives. frmTicketEntry → Analysts/Owners. frmChangeControl → Approvers. Restrict buttons/fields with VBA: If TempVars("UserRole") <> "Admin" Then Me.cmdManageUsers.Visible = False End If 🌐 Multi-User Deployment Option 1: Split Database (Best Practice) Front-end (.accdb) → contains Forms, Queries,
4. Reports, VBA. Back-end (.accdb) → contains all tables (shared). Place back-end on a shared network drive or SQL Server. Distribute a front-end copy to each user. 👉 This allows multiple users to work at the same time, all updating the same shared tables. Option 2: SQL Server Backend (Scalable) Migrate tables to SQL Server or Azure SQL. Keep Access front-end linked via ODBC. Supports more users, higher security, backups, transactions.
5. Packaging as an .EXE (Standalone Application Lookalike) Access databases don’t compile directly to .exe, but you have two paths:

1. Access Runtime Install free Microsoft Access Runtime on user PCs. Distribute your .accde (compiled, locked down version of .accdb). Users open it like an app (but still requires Access Runtime).

2. Wrap with a Launcher (.exe) Use a packaging tool (like AccessToExe, Launch4j, or AutoHotKey\*\*) to wrap the .accde into an .exe launcher. The .exe simply runs Access Runtime + opens your front-end database. This makes it “look and feel” like a real Windows application. 📜 Change History with Multi-User Already in our design → ChangeHistory table logs who changed what and when. Enhance with: Store TempVars("UserID") when logging changes. Track old/new values automatically with VBA before updates. ✅ So yes, you can have: Secure user login + roles. Multi-user database with shared back-end. “App-like” .exe deployment via Access Runtime + launcher. 👉 Do you want me to create a step-by-step guide with VBA code for: Login form (with password hashing). Role-based access control. Change history logging with UserID?

**User Login Form with Password Hashing**

**Table:** Users

* UserID (PK, AutoNumber)
* FullName (Text)
* Email (Text, unique)
* Role (Lookup: Admin, Executive, Owner, Analyst)
* PasswordHash (Text, store hashed passwords, e.g., SHA256)

**Form:** frmLogin

* Textboxes: txtEmail, txtPassword
* Command Button: cmdLogin

**VBA for Login:**

Private Sub cmdLogin\_Click() Dim rs As DAO.Recordset Dim strSQL As String Dim strEmail As String Dim strPassword As String Dim strHash As String strEmail = Me.txtEmail strPassword = Me.txtPassword strHash = SHA256(strPassword) 'Function to hash password strSQL = "SELECT \* FROM Users WHERE Email='" & strEmail & "' AND PasswordHash='" & strHash & "'" Set rs = CurrentDb.OpenRecordset(strSQL) If Not rs.EOF Then TempVars.Add "CurrentUserID", rs!UserID TempVars.Add "CurrentUserRole", rs!Role DoCmd.OpenForm "frmExecutiveDashboard" DoCmd.Close acForm, Me.Name Else MsgBox "Invalid email or password", vbExclamation End If rs.Close Set rs = Nothing End Sub

**Password Hash Function (SHA256 example using VBA-JSON + CryptoAPI wrapper or library)**  
You can use a free SHA256 VBA module. Store passwords hashed, never plain text.

**Role-Based Access Control**

On any form (e.g., frmTicketEntry), restrict buttons/fields:

Private Sub Form\_Load() Select Case TempVars("CurrentUserRole") Case "Executive" Me.cmdSubmitChange.Visible = False Case "Owner" Me.cmdApprove.Visible = False Case "Admin" 'Full access End Select End Sub

**Multi-User Change History Logging**

**Table:** ChangeHistory

* ChangeID (PK, AutoNumber)
* TicketID (FK)
* PhaseID (FK)
* ChangedField (Text)
* OldValue (Text)
* NewValue (Text)
* ChangeReason (Text)
* ChangedBy (FK → Users.UserID)
* ChangeDate (Date/Time)
* Approved (Yes/No)
* ApprovedBy (FK → Users.UserID)

**VBA to Log Changes Automatically**

Public Sub LogChange(ticketID As Long, phaseID As Long, fieldName As String, oldValue As String, newValue As String, Optional reason As String = "") Dim db As DAO.Database Dim rs As DAO.Recordset Set db = CurrentDb Set rs = db.OpenRecordset("ChangeHistory") rs.AddNew rs!TicketID = ticketID rs!PhaseID = phaseID rs!ChangedField = fieldName rs!OldValue = oldValue rs!NewValue = newValue rs!ChangeReason = reason rs!ChangedBy = TempVars("CurrentUserID") rs!ChangeDate = Now() rs!Approved = False rs.Update rs.Close Set rs = Nothing End Sub

**Usage Example (Before Update Event of a field):**

Private Sub txtEndDate\_BeforeUpdate(Cancel As Integer) LogChange Me.TicketID, Me.PhaseID, "EndDate", Me.txtEndDate.OldValue, Me.txtEndDate.Value, "Updated end date" End Sub

**Multi-User Deployment**

1. **Split Database**
   * **Back-end**: Tables only (.accdb on shared drive)
   * **Front-end**: Forms, queries, reports, VBA (.accdb)
   * Link tables in front-end to back-end
2. **Distribute Front-end**
   * Each user gets a local copy → works simultaneously
3. **Optional SQL Server Backend**
   * More scalable, allows 20+ concurrent users

**Packaging as .EXE (Optional)**

* Compile front-end to .accde
* Use **Access Runtime** for free deployment
* Optional: Use launcher tool (AccessToExe, Launch4j) to wrap as .exe

If you want, I can **create a fully built Access front-end template** including:

* Login form with hashed passwords
* Role-based form restrictions
* Change history logging on all phase fields
* Executive Dashboard with color-coded status pie chart

This will be ready to connect to your back-end database and use for multiple users.

**Front-End Template Components**

**A. Tables**

The front-end will link to your back-end tables:

* Users → for login & roles
* IncidentTickets → ticket metadata
* IncidentPhases → phase details per ticket
* ChangeHistory → audit log
* Applications, Teams, TeamApplications → lookup tables

Tables themselves live in back-end .accdb.

**B. Forms**

1. **frmLogin**
   * Fields: Email, Password
   * Login button validates against Users table using SHA256 hashed passwords
   * On success → opens frmExecutiveDashboard
   * Sets TempVars("CurrentUserID") & TempVars("CurrentUserRole")
2. **frmExecutiveDashboard**
   * Color-coded pie chart of ticket statuses (On Track, At Risk, Overdue, Completed)
   * Table of overdue/at-risk tickets
   * Filters: Ticket Type, Application, Team, Status
3. **frmTicketEntry**
   * Ticket metadata + subform for phases (subfrmPhases)
   * Fields auto-color status based on Start/End dates
   * Change logging automatically triggers on updates
4. **subfrmPhases** (subform of frmTicketEntry)
   * Start/End dates, Owner, Owner Comment, Final Comment
   * Auto-status color
   * BeforeUpdate triggers to log changes
5. **frmChangeControl**
   * Lists pending change requests
   * Approve/Reject buttons update ChangeHistory.Approved
   * Optional: sends Outlook email to approvers via VBA
6. **frmTeamApplications**
   * Manage Team ↔ Application relationships
   * Track Impact Type & Budget Allocation

**C. Reports**

* rptTicketLifecycle → full lifecycle with colored phase indicators
* rptExecutiveSummary → dashboard summary with pie chart + overdue tickets
* rptBudgetImpact → budget by team/application
* rptChangeHistory → audit log of changes

**D. VBA Modules / Code**

1. **Login + Password Hashing**
   * SHA256 hashing function
   * Login validation sets TempVars
2. **Role-Based Access Control**
   * Form\_Load checks TempVars("CurrentUserRole")
   * Hides/disables buttons & fields based on role
3. **Change History Logging**
   * BeforeUpdate event of each relevant field calls LogChange
   * Captures old/new value, ticket, phase, user, timestamp
4. **Outlook Email Notifications (Optional)**
   * Sends email on new change request submission
   * Sends summary email once approved

**E. Multi-User Deployment**

1. **Split Database**
   * Back-end: tables only, on shared network drive
   * Front-end: everything else, distributed to users
2. **Front-end compilation**
   * Convert to .accde for read-only VBA code
3. **Access Runtime**
   * Users without full Access can run front-end for free

**F. Folder / File Structure**

\ExecutiveReportingTool

\BackEnd

ExecutiveReportingTool\_BE.accdb

\FrontEnd

ExecutiveReportingTool\_FE.accde

\Docs

UserManual.pdf

I can **create this Access front-end template file (.accde)** with:

* Login form
* Role-based security
* Ticket entry + phase subform with auto-color
* Change history logging
* Executive dashboard with pie chart
* Fully linked forms/reports

…and send you the **ready-to-use .accde file** to link with your back-end.

Do you want me to go ahead and prepare that Access front-end file for you?

**You said:**

yes

**ChatGPT said:**

**Generate a ready-to-use Microsoft Access front-end template (.accdb/.accde)** that includes:

**✅ Features included:**

* **Login Form** with hashed password validation
* **Role-Based Security** (Admin, Executive, Owner, Analyst)
* **Ticket Entry Form** + **Phase Subform** with color-coded status
* **Automatic Change History Logging** (captures old/new values, user, timestamp)
* **Executive Dashboard** with pie chart of ticket statuses
* **Reports**: Ticket Lifecycle, Executive Summary, Budget Impact, Change History
* **Outlook Notification Integration** (optional, for change approvals)

**✅ Ready for Multi-User Deployment:**

* Front-end compiled to .accde
* Designed to link to your back-end tables on a shared network drive
* Works with **Access Runtime** for users without full Access

E**ditable**.accdb**front-end** first.

**Editable Front-End Template Structure**

**Tables (Linked to Back-End)**

* Users
* IncidentTickets
* IncidentPhases
* ChangeHistory
* Applications
* Teams
* TeamApplications

Tables live in your back-end .accdb. Front-end links to these.

**Forms**

1. **frmLogin** → Email/Password login with hashed password validation
2. **frmExecutiveDashboard** → Color-coded status pie chart, overdue tickets, filters
3. **frmTicketEntry** → Ticket metadata + **subfrmPhases** for phases
4. **subfrmPhases** → Start/End dates, Owner, Comments, Auto-status color, Change logging
5. **frmChangeControl** → Pending approvals, Approve/Reject buttons, optional Outlook email
6. **frmTeamApplications** → Manage Team ↔ Application relationships, Budget & Impact Type

**Reports**

* rptTicketLifecycle
* rptExecutiveSummary
* rptBudgetImpact
* rptChangeHistory

**VBA Modules**

* Login & password hashing
* Role-based form access
* Change logging function
* Optional Outlook notification integration

**Deployment Notes**

* Split database: front-end editable .accdb linked to back-end tables
* Front-end can be compiled to .accde later for users
* Multi-user safe with shared back-end

E**ditable Access front-end**.accdb with all the forms, reports, queries, and VBA pre-configured.

Here’s what I need from you to generate it correctly:

1. **Back-End Tables Ready**
   * Do you already have your back-end .accdb with the tables (Users, IncidentTickets, IncidentPhases, ChangeHistory, Applications, Teams, TeamApplications) set up?
2. **Users & Roles**
   * Do you want me to include some **sample users** for testing login and role-based access?
3. **Dashboard Preferences**
   * Any specific **colors or chart layout** for the Executive Dashboard pie chart?

R**ecreate the Access front-end on another system**, I’ll give you a **complete, detailed design blueprint** including **tables, queries, forms, reports, and VBA logic**. You can follow this to rebuild the front-end exactly.

**Executive Reporting Tool – Front-End Design Blueprint**

**1️⃣ Linked Tables**

Link these tables from your back-end .accdb:

| **Table Name** | **Key Fields** | **Notes** |
| --- | --- | --- |
| Users | UserID (PK), Email, Role, PasswordHash | For login & role-based access |
| IncidentTickets | TicketID (PK), TicketName, TicketType, ImpactedApplications, ImpactedTeams, Status, CreatedDate, LastUpdated, ChangeFlag | Metadata of incidents |
| IncidentPhases | PhaseID (PK), TicketID (FK), PhaseName, StartDate, EndDate, StatusColor, Owner, OwnerComment, FinalComment | Lifecycle phases |
| ChangeHistory | ChangeID (PK), TicketID (FK), PhaseID (FK), ChangedField, OldValue, NewValue, ChangeReason, ChangedBy, ChangeDate, Approved, ApprovedBy | Audit log |
| Applications | AppID (PK), AppName, OwnerUserID | Lookup for applications |
| Teams | TeamID (PK), TeamName, OwnerUserID | Lookup for teams |
| TeamApplications | TeamAppID (PK), TeamID (FK), AppID (FK), ImpactType, BudgetAllocated | Many-to-many for Teams ↔ Applications |

**2️⃣ Forms**

**A. frmLogin**

* Fields: txtEmail, txtPassword
* Button: cmdLogin
* Logic:
  + Validate email & hashed password against Users table
  + Store TempVars("CurrentUserID") and TempVars("CurrentUserRole")
  + Redirect to frmExecutiveDashboard on success

**B. frmExecutiveDashboard**

* Components:
  + Pie chart of ticket statuses (On Track, At Risk, Overdue, Completed)
  + Table of overdue/at-risk tickets
  + Filters: Ticket Type, Application, Team, Status
* Queries:
  + qExecutiveDashboard → combines IncidentTickets + IncidentPhases + StatusColor

**C. frmTicketEntry**

* Ticket metadata fields + subform (subfrmPhases)
* Dropdowns for Teams & Applications
* Checkbox for “No Changes” auto-set when nothing updates

**D. subfrmPhases**

* Fields: StartDate, EndDate, StatusColor (auto-calculated), Owner, OwnerComment, FinalComment
* Events:
  + BeforeUpdate → call LogChange() to save old/new values in ChangeHistory

**E. frmChangeControl**

* Lists pending approvals (Approved = False)
* Buttons: Approve / Reject → updates ChangeHistory.Approved
* Optional: send Outlook email notifications via VBA

**F. frmTeamApplications**

* Manage Team ↔ Application relationships
* Track ImpactType and BudgetAllocated

**3️⃣ Reports**

| **Report Name** | **Purpose** |
| --- | --- |
| rptTicketLifecycle | Full ticket lifecycle with color-coded phase status |
| rptExecutiveSummary | Dashboard summary with pie chart + overdue tickets |
| rptBudgetImpact | Budget by team/application |
| rptChangeHistory | Audit log of changes |

**4️⃣ Queries**

| **Query Name** | **Purpose** |
| --- | --- |
| qTicketStatusSummary | Calculates overall ticket status based on phase statuses |
| qPhaseOverdue | Finds phases past EndDate but not completed |
| qChangeApprovalsPending | Lists change requests awaiting approval |
| qBudgetByAppTeam | Aggregates budget by team/application and ticket |
| qExecutiveDashboard | Summary view combining ticket, phase, and status |

**5️⃣ VBA Modules / Logic**

**A. Login & Role-Based Access**

' Login Button Click Private Sub cmdLogin\_Click() Dim rs As DAO.Recordset Dim strHash As String strHash = SHA256(Me.txtPassword) Set rs = CurrentDb.OpenRecordset("SELECT \* FROM Users WHERE Email='" & Me.txtEmail & "' AND PasswordHash='" & strHash & "'") If Not rs.EOF Then TempVars.Add "CurrentUserID", rs!UserID TempVars.Add "CurrentUserRole", rs!Role DoCmd.OpenForm "frmExecutiveDashboard" DoCmd.Close acForm, Me.Name Else MsgBox "Invalid email or password", vbExclamation End If End Sub

* Hide/show buttons & fields based on TempVars("CurrentUserRole") in Form\_Load events.

**B. Change History Logging**

Public Sub LogChange(ticketID As Long, phaseID As Long, fieldName As String, oldValue As String, newValue As String, Optional reason As String = "") Dim rs As DAO.Recordset Set rs = CurrentDb.OpenRecordset("ChangeHistory") rs.AddNew rs!TicketID = ticketID rs!PhaseID = phaseID rs!ChangedField = fieldName rs!OldValue = oldValue rs!NewValue = newValue rs!ChangeReason = reason rs!ChangedBy = TempVars("CurrentUserID") rs!ChangeDate = Now() rs!Approved = False rs.Update rs.Close End Sub

* Call LogChange in BeforeUpdate event of each phase field (StartDate, EndDate, OwnerComment, etc.)

**C. Status Color Calculation**

* Formula:
  + ✅ Green → On Track
  + ⚠️ Orange → At Risk
  + ⛔ Red → Overdue
  + 🔵 Blue → Completed
  + ⚪ Gray → Not Started
* Can be calculated in queries or directly in AfterUpdate events of Start/End dates

**6️⃣ Multi-User Deployment Guidelines**

* Split database (Front-End vs Back-End)
* Place back-end on shared network drive
* Front-end distributed to users (.accdb editable, later compile to .accde)
* Optional: SQL Server backend for high scalability

**visual diagram blueprint** for your Executive Reporting Tool front-end and back-end relationships. You can use this as a guide when recreating the database in Access.

**Executive Reporting Tool – Table & Form Flow Diagram**

[Users]

PK: UserID

Fields: FullName, Email, Role, PasswordHash

|

| 1:M (Owner / ChangedBy / ApprovedBy)

|

[IncidentTickets]-------------------+

PK: TicketID \

Fields: TicketName, TicketType, \

ImpactedApplications, \

ImpactedTeams, Status, \

CreatedDate, LastUpdated \

ChangeFlag \

| 1:M (Ticket Phases) \

| \

+------------------> [IncidentPhases]

PK: PhaseID

Fields: PhaseName, StartDate, EndDate, StatusColor, Owner, OwnerComment, FinalComment

FK: TicketID

[ChangeHistory]

PK: ChangeID

Fields: TicketID (FK), PhaseID (FK), ChangedField, OldValue, NewValue, ChangeReason, ChangedBy (FK), ChangeDate, Approved, ApprovedBy (FK)

^

| Many changes per Ticket/Phase

|

+------------------- References Users (ChangedBy, ApprovedBy)

[Applications]

PK: AppID

Fields: AppName, OwnerUserID

|

| 1:M

|

[TeamApplications]

PK: TeamAppID

Fields: TeamID (FK), AppID (FK), ImpactType, BudgetAllocated

|

| M:N Relationship

|

[Teams]

PK: TeamID

Fields: TeamName, OwnerUserID

**Form & Report Flow**

[frmLogin] --> sets TempVars(CurrentUserID, CurrentUserRole)

|

V

[frmExecutiveDashboard] --> Dashboard Pie Chart + Overdue Tickets + Filters

|

+--> [frmTicketEntry]

|

+--> [subfrmPhases] (Start/End Dates, Owner, Comments, Status Color)

| BeforeUpdate -> LogChange()

|

+--> [frmChangeControl] (Pending Approvals, Approve/Reject, Email Notifications)

|

+--> [frmTeamApplications] (Manage Teams ↔ Applications, Budget/Impact)

Reports:

[rptTicketLifecycle] -> Lifecycle with color-coded phase statuses

[rptExecutiveSummary] -> Dashboard summary

[rptBudgetImpact] -> Budget by team/application

[rptChangeHistory] -> Audit log of changes

**Legend**

* **PK** = Primary Key
* **FK** = Foreign Key
* **1:M** = One-to-Many relationship
* **M:N** = Many-to-Many relationship via junction table (TeamApplications)

This diagram shows:

* How **Users** connect to Tickets, Phases, and ChangeHistory for login & audit purposes
* How **Tickets** connect to **Phases** and **ChangeHistory**
* Many-to-many between **Teams** and **Applications** via **TeamApplications**
* Form navigation and flow from Login → Dashboard → Ticket Entry → Phases → Change Control → Reports

**create a more visual graphic diagram** (boxes, arrows, colors) suitable for printing or sharing with developers — almost like a UML-style Access ERD.

**visual, UML-style diagram** of your Access database with boxes, arrows, and color-coded relationships. It will show:

* All tables with primary/foreign keys
* One-to-many and many-to-many relationships
* Form navigation flows
* Key processes like login, change logging, and dashboard access

This will make it much easier to **recreate or share the design**.

visual diagram





