Page-by-Page Navigation and SQL Queries

MehfoozPakistan Crime Pattern Analysis System
Comprehensive Database Query Documentation

December 9, 2025

Contents

1 Introduction

This document provides comprehensive documentation of every screen in the MehfoozPakistan CPAS application and the corresponding SQL queries executed. Each page section includes:

- Page purpose and functionality
- User interactions available
- SQL queries executed (SELECT, INSERT, UPDATE, DELETE)
- Query explanations and performance considerations
- Data flow and business logic

2 Authentication Pages

2.1 Officer Login Page

2.1.1 Page Screenshot

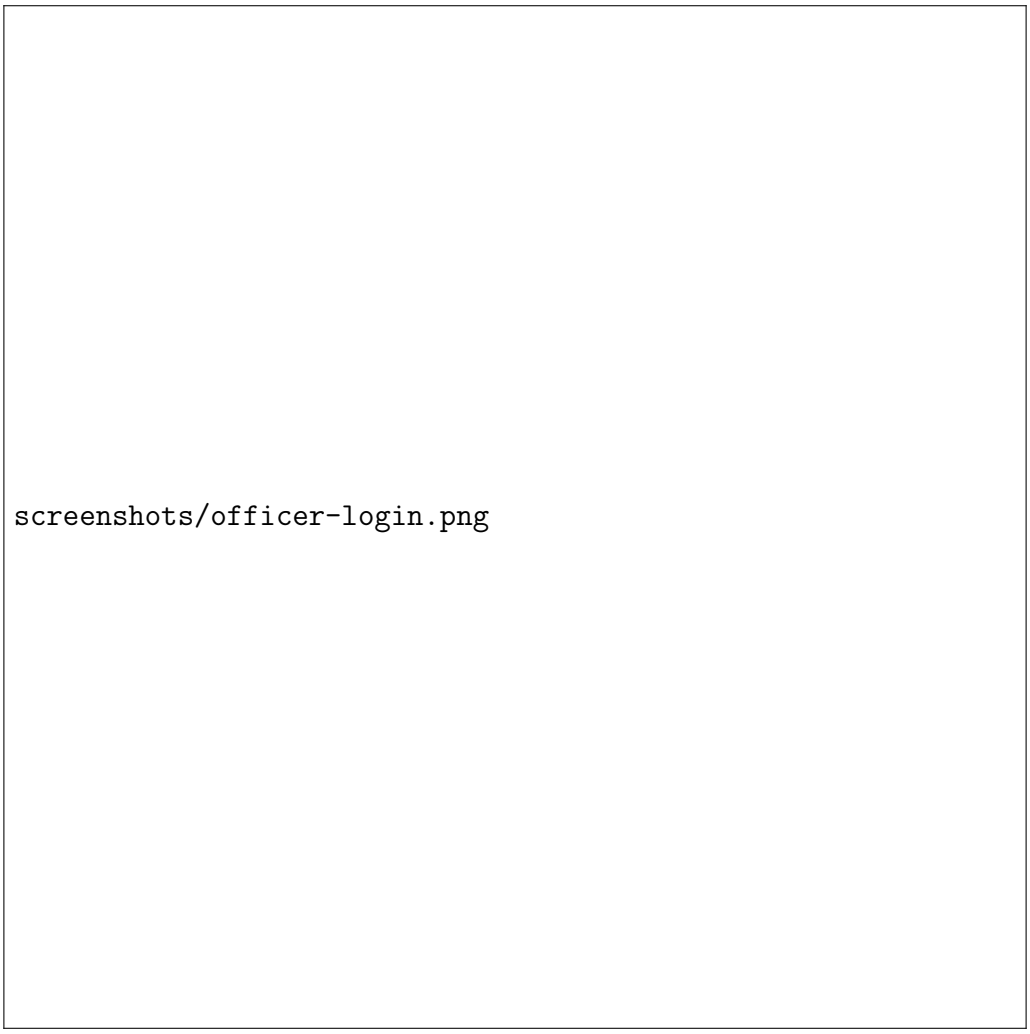

Figure 1: Officer Login Page - Badge Number and Password Authentication

2.1.2 Page Purpose

Authenticates law enforcement officers using their badge number and password, providing access to full CRUD operations on all crime management features.

2.1.3 User Interactions

- Enter badge number (Officer_ID)
- Enter password
- Submit login form
- Receive JWT token upon successful authentication

2.1.4 SQL Query: Validate Officer Credentials

```
SELECT
    Officer_ID,
    Name,
    Contact_No,
    Email,
    Password
FROM Officer
WHERE Officer_ID = ?
```

Listing 1: Query to authenticate officer by badge number

Query Explanation:

- **Purpose:** Retrieves officer record by unique badge number (Officer_ID)
- **Parameters:** Badge number entered by user
- **Password Verification:** Password hash is compared using bcrypt in application layer
- **Security:** Plain-text passwords are never stored; only bcrypt hashes
- **Index Usage:** Primary key index on Officer_ID ensures O(1) lookup
- **Result:** Single row if officer exists, empty set if not found

2.2 Victim Login Page

2.2.1 Page Screenshot

Figure 2: Victim Login Page - CNIC-based Authentication

2.2.2 Page Purpose

Allows crime victims to log in using their CNIC (National ID) and password to view their submitted crime reports and case status.

2.2.3 SQL Query: Authenticate Victim

```
SELECT
    Victim_ID,
    Name,
    Age,
    Gender,
    Contact_Info,
    Email,
    Password
FROM Victim
```

```
10 WHERE Email = ?
```

Listing 2: Query to validate victim login credentials

2.3 Witness Login Page

2.3.1 Page Screenshot

Figure 3: Witness Login Page - Email-based Authentication

2.3.2 SQL Query: Authenticate Witness

Purpose: Retrieves victim record using email as unique identifier

Index: Uses idx_victim_email index for fast lookup

Alternative Identifier: In some implementations, CNIC can be used instead of email

Security: Password comparison performed in application layer using bcrypt

2.4 Witness Login Page

2.4.1 SQL Query: Authenticate Witness

```sql
SELECT
    Witness_ID,
    Name,
    Contact_Info,
    Email,
    Password
FROM Witness
WHERE Email = ?
```

Listing 3: Query to authenticate witness by email

2.5 Officer Dashboard

2.5.1 Page Screenshot

screenshots/officer-dashboard.png

Figure 4: Officer Dashboard - Statistics Overview and Quick Actions

2.5.2 Page Purpose

Provides officers with an overview of crime statistics, recent activities, and quick access to all management features.

2.5.3 SQL Query: Get Dashboard Statistics

3 Dashboard Pages

3.1 Officer Dashboard

3.1.1 Page Purpose

Provides officers with an overview of crime statistics, recent activities, and quick access to all management features.

3.1.2 SQL Query: Get Dashboard Statistics

```
SELECT
    COUNT(*) AS Total_Crimes,
    SUM(CASE WHEN Status = 'Open' THEN 1 ELSE 0 END) AS
       Open_Crimes,
    SUM(CASE WHEN Status = 'Under Investigation' THEN 1 ELSE 0
       END) AS Investigating,
    SUM(CASE WHEN Status = 'Closed' THEN 1 ELSE 0 END) AS
       Closed_Crimes,
    SUM(CASE WHEN Severity_Level = 'Critical' THEN 1 ELSE 0 END)
       AS Critical_Cases,
    SUM(CASE WHEN Date_Occurred >= SYSDATE - 7 THEN 1 ELSE 0 END)
        AS This_Week
FROM Crime
```

Listing 4: Query to fetch crime overview statistics

Query Explanation:

- **Purpose:** Single query to fetch multiple aggregated statistics
- **Performance:** Uses CASE expressions for conditional counting
- **Efficiency:** One table scan instead of multiple queries
- **Index Usage:** idx_crime_status and idx_date_occurred may be used

3.1.3 SQL Query: Get Recent Crimes

```
SELECT
    c.Crime_ID,
    ct.Type_Name AS Crime_Type,
    c.Date_Occurred,
    c.Status,
    c.Severity_Level,
```

```
    l.City,
    l.Area,
    o.Name AS Assigned_Officer
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
LEFT JOIN Location l ON c.Location_ID = l.Location_ID
LEFT JOIN Officer o ON c.Officer_ID = o.Officer_ID
ORDER BY c.Date_Occurred DESC
FETCH FIRST 10 ROWS ONLY
```

Listing 5: Query to display recent crime activities

3.2 Analytics Dashboard

3.2.1 Page Screenshot

Figure 5: Analytics Dashboard - Charts, Trends, and Visualizations

3.2.2 Page Purpose

Displays comprehensive crime analytics including charts, trends, predictions, and geographic hotspots.

3.2.3 SQL Query: Crime Distribution by Type

s MySQL LIMIT

Sorting: ORDER BY Date_Occurred DESC with idx_date_occurred index

3.3 Analytics Dashboard

3.3.1 Page Purpose

Displays comprehensive crime analytics including charts, trends, predictions, and geographic hotspots.

3.3.2 SQL Query: Crime Distribution by Type

```
SELECT
    ct.Type_Name,
    ct.Category,
    COUNT(c.Crime_ID) AS Total_Crimes,
    ROUND(COUNT(c.Crime_ID) * 100.0 / SUM(COUNT(c.Crime_ID)) OVER
        (), 2) AS Percentage
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
GROUP BY ct.Type_Name, ct.Category
ORDER BY Total_Crimes DESC
```

Listing 6: Query for crime type distribution chart

Query Explanation:

- **Purpose:** Generates data for pie/bar charts showing crime distribution
- **Window Function:** SUM() OVER() calculates percentage of total
- **GROUP BY:** Aggregates crimes by type
- **Use Case:** Frontend renders as bar chart or pie chart

3.3.3 SQL Query: Crime Trends with Advanced Window Functions

```
SELECT
    ct.Type_Name AS Crime_Type,
    EXTRACT(YEAR FROM c.Date_Occurred) AS Year,
    EXTRACT(MONTH FROM c.Date_Occurred) AS Month,
    COUNT(*) AS Total_Crimes,
    SUM(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                        ORDER BY EXTRACT(YEAR FROM c.
                            Date_Occurred),
```

```
                                    EXTRACT(MONTH FROM c.
                                        Date_Occurred)) AS
                                        Cumulative_Crimes,
    LAG(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                        ORDER BY EXTRACT(YEAR FROM c.
                            Date_Occurred),
                                    EXTRACT(MONTH FROM c.
                                        Date_Occurred)) AS
                                        Previous_Month_Crimes,
    ROUND(
        (COUNT(*) - LAG(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                                        ORDER BY EXTRACT(YEAR
                                            FROM c.Date_Occurred),
                                                EXTRACT(MONTH
                                                    FROM c.
                                                    Date_Occurred
                                                    ))) * 100.0 /
        NULLIF(LAG(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                                    ORDER BY EXTRACT(YEAR FROM c.
                                        Date_Occurred),
                                            EXTRACT(MONTH FROM c.
                                                Date_Occurred)),
                                                0),
        2
    ) AS Month_Over_Month_Change
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
WHERE 1=1
    AND (:year IS NULL OR EXTRACT(YEAR FROM c.Date_Occurred) = :
        year)
    AND (:month IS NULL OR EXTRACT(MONTH FROM c.Date_Occurred) =
        :month)
    AND (:crimeTypeId IS NULL OR c.Crime_Type_ID = :crimeTypeId)
GROUP BY ct.Type_Name, EXTRACT(YEAR FROM c.Date_Occurred),
    EXTRACT(MONTH FROM c.Date_Occurred)
ORDER BY ct.Type_Name, Year, Month
```

Listing 7: Query for crime trends with month-over-month changes

Query Explanation:

- **Purpose:** Advanced trend analysis with multiple window functions
- **PARTITION BY:** Separates calculations by crime type
- **Window Functions:**
  - SUM() OVER: Running total of crimes per type
  - LAG(): Retrieves previous month's count for comparison
- **Month-Over-Month Change:** Percentage change calculation

- **NULLIF:** Prevents division by zero
- **Use Case:** Identifies growing/declining crime trends

3.3.4 SQL Query: Monthly Crime Trends (Using View)

```
SELECT
    Crime_Type,
    City,
    Month,
    Year,
    Total_Crimes
FROM Crime_Trends_Monthly
WHERE Year >= EXTRACT(YEAR FROM SYSDATE) - 1
ORDER BY Year DESC, Month DESC, Total_Crimes DESC
```

Listing 8: Query to fetch monthly crime trends from view

Query Explanation:

- **Purpose:** Uses pre-computed view for fast analytics
- **View Definition:** Crime_Trends_Monthly aggregates monthly statistics
- **Performance:** View eliminates need for complex GROUP BY in application
- **Time Range:** Filters last 12 months of data
- **Frontend Use:** Data rendered as line graph showing trends

3.3.5 SQL Query: Geographic Crime Hotspots (Using View)

```
SELECT
    Location_ID,
    City,
    Area,
    Street,
    Total_Crimes,
    Solved_Cases,
    Solve_Rate
FROM Crime_Hotspots
WHERE Total_Crimes >= 5
ORDER BY Total_Crimes DESC
FETCH FIRST 20 ROWS ONLY
```

Listing 9: Query to identify crime hotspots

Query Explanation:

- **Purpose:** Identifies top 20 high-crime locations
- **View Usage:** Crime_Hotspots view pre-calculates solve rates
- **Threshold:** Only locations with 5+ crimes included
- **Frontend Rendering:** Data displayed on heatmap/geographic map

3.3.6 SQL Query: Call Statistics Stored Procedure

```
DECLARE
    v_stats SYS_REFCURSOR;
    v_category VARCHAR2(20);
    v_total NUMBER;
    v_solved NUMBER;
    v_solve_rate NUMBER;
    v_critical_pct NUMBER;
BEGIN
    sp_calculate_crime_statistics(
        p_start_date => TO_DATE('2024-01-01', 'YYYY-MM-DD'),
        p_end_date => SYSDATE,
        p_stats => v_stats
    );

    -- Fetch results
    LOOP
        FETCH v_stats INTO v_category, v_total, v_solved,
           v_solve_rate, v_critical_pct;
\subsection{View All Crimes Page}

\subsubsection{Page Screenshot}

\begin{figure}[H]
    \centering
    \includegraphics[width=0.9\textwidth]{screenshots/crimes-list
       .png}
    \caption{View All Crimes - Paginated Table with Filters}
    \label{fig:sql-crimes-list}
\end{figure}

\subsubsection{Page Purpose}
Displays a searchable, sortable, paginated table of all crimes in
    the system with filters.

\subsubsection{SQL Query: Fetch Paginated Crime List}

\textbf{Query Explanation:}
\begin{itemize}
    \item \textbf{Purpose:} Calls stored procedure for complex
       statistical analysis
    \item \textbf{Procedure:} sp\_calculate\_crime\_statistics
       returns SYS\_REFCURSOR
    \item \textbf{Parameters:} Date range for analysis
    \item \textbf{Returns:} Category-wise statistics including
       solve rates
    \item \textbf{Business Logic:} Encapsulated in database for
       consistency
\end{itemize}

```

```
\section{Crime Management Pages}

\subsection{View All Crimes Page}

\subsubsection{Page Purpose}
Displays a searchable, sortable, paginated table of all crimes in
    the system with filters.

\subsubsection{SQL Query: Fetch Paginated Crime List}

\begin{lstlisting}[caption={Query to retrieve paginated crime
   list}]
SELECT
    c.Crime_ID,
    ct.Type_Name AS Crime_Type,
    ct.Category,
    c.Date_Occurred,
    c.Date_Reported,
    c.Status,
    c.Severity_Level,
    l.City,
    l.Area,
    o.Name AS Officer_Name
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
LEFT JOIN Location l ON c.Location_ID = l.Location_ID
LEFT JOIN Officer o ON c.Officer_ID = o.Officer_ID
WHERE 1=1
    AND (:crime_type_id IS NULL OR c.Crime_Type_ID = :
       crime_type_id)
    AND (:status IS NULL OR c.Status = :status)
    AND (:severity IS NULL OR c.Severity_Level = :severity)
    AND (:city IS NULL OR l.City = :city)
ORDER BY c.Date_Occurred DESC
OFFSET :offset ROWS FETCH NEXT :page_size ROWS ONLY
```

Listing 10: Calling crime statistics stored procedure

Query Explanation:

- **Purpose:** Main query for crime list page with dynamic filtering
- **Pagination:** OFFSET and FETCH for Oracle pagination
- **Dynamic Filters:** Parameterized WHERE conditions allow flexible filtering
- **Joins:** Brings related type, location, and officer information
- **Index Usage:** Multiple indexes (crime_type, status, location) optimize filters
- **Performance:** Parameterized queries prevent SQL injection and enable query plan caching

3.3.7 SQL Query: Count Total Crimes (for Pagination)

3.4 Add New Crime Page

3.4.1 Page Screenshot

Figure 6: Add New Crime - Form with Dropdowns and Input Fields

3.4.2 Page Purpose

Allows officers to create new crime records with all required and optional details.

3.4.3 SQL Query: Get Crime Types for Dropdown

$\text{rime}_t ype_i d)AND(: statusISNULLORc.Status =: status)AND(: severityISNULLORc.Severity_Leve$
$severity)AND(: cityISNULLORl.City =: city)$

Query Explanation:

- **Purpose:** Counts total matching records for pagination calculations
- **Same Filters:** Must match main query filters exactly
- **Frontend Use:** Calculates total pages and displays pagination controls

3.5 Add New Crime Page

3.5.1 Page Purpose

Allows officers to create new crime records with all required and optional details.

3.5.2 SQL Query: Get Crime Types for Dropdown

```
SELECT
    Crime_Type_ID,
    Type_Name,
    Category,
    Description
FROM Crime_Type
ORDER BY Type_Name ASC
```

Listing 11: Query to populate crime type dropdown

Query Explanation:

- **Purpose:** Populates dropdown menu with all available crime types
- **Sorting:** Alphabetical order for user convenience
- **Caching:** Results can be cached in frontend as data rarely changes

3.5.3 SQL Query: Get Locations for Dropdown

```
SELECT
    Location_ID,
    City,
    Area,
    Street,
    City || ', ' || NVL(Area, '') AS Display_Name
FROM Location
ORDER BY City, Area, Street
```

Listing 12: Query to populate location dropdown

Query Explanation:

- **Purpose:** Provides location options for crime location selection
- **Display_Name:** Concatenated string for dropdown display
- **NVL Function:** Handles NULL values in Area field
- **Index:** idx_city_area speeds up sorted retrieval

3.5.4 SQL Query: Get Officers for Assignment

```
SELECT
    Officer_ID,
    Name,
    Contact_No,
    Email
FROM Officer
ORDER BY Name ASC
```

Listing 13: Query to fetch officers for assignment dropdown

Query Explanation:

- **Purpose:** Lists all officers who can be assigned to crimes
- **Use Case:** Optional assignment during crime creation

3.5.5 SQL Query: Insert New Crime

```
INSERT INTO Crime (
    Crime_Type_ID,
    Date_Reported,
    Date_Occurred,
    Time_Occurred,
    Description,
    Status,
    Severity_Level,
    Location_ID,
    Officer_ID,
    Related_Crime_ID
) VALUES (
    :crime_type_id,
    SYSDATE,
    :date_occurred,
    :time_occurred,
    :description,
    :status,
\subsection{Edit Crime Page}

\subsubsection{Page Screenshot}

\begin{figure}[H]
    \centering
    \includegraphics[width=0.85\textwidth]{screenshots/edit-crime
       .png}
    \caption{Edit Crime - Pre-filled Form for Updating Crime
       Records}
    \label{fig:sql-edit-crime}
\end{figure}

\subsubsection{Page Purpose}
```

```
Allows modification of existing crime records with pre-filled
   form data.

\subsubsection{SQL Query: Fetch Crime Details for Editing}
```

Listing 14: Query to insert new crime record

Query Explanation:

- **Purpose:** Inserts new crime record into database
- **Auto-Increment:** crime_bir trigger automatically assigns Crime_ID from sequence
- **RETURNING Clause:** Returns newly generated Crime_ID to application
- **Trigger Execution:** crime_bir trigger also auto-populates Day_Of_Week
- **Validation:** Trigger validates Date_Reported ¿= Date_Occurred
- **Constraints:** CHECK constraints enforce valid Status and Severity_Level values

3.6 Edit Crime Page

3.6.1 Page Purpose

Allows modification of existing crime records with pre-filled form data.

3.6.2 SQL Query: Fetch Crime Details for Editing

```
SELECT
    c.Crime_ID,
    c.Crime_Type_ID,
    c.Date_Reported,
    c.Date_Occurred,
    c.Time_Occurred,
    c.Day_Of_Week,
    c.Description,
    c.Status,
    c.Severity_Level,
    c.Location_ID,
    c.Officer_ID,
    c.Related_Crime_ID,
    ct.Type_Name,
    l.City,
    l.Area,
    o.Name AS Officer_Name
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
LEFT JOIN Location l ON c.Location_ID = l.Location_ID
LEFT JOIN Officer o ON c.Officer_ID = o.Officer_ID
WHERE c.Crime_ID = :crime_id
```

Listing 15: Query to load existing crime data into form

Query Explanation:

- **Purpose:** Retrieves complete crime details for pre-filling edit form
- **Primary Key Lookup:** Uses Crime_ID for O(1) retrieval
- **Related Data:** Joins fetch type, location, and officer names for display

3.6.3 SQL Query: Update Crime Record

```
UPDATE Crime
SET
    Crime_Type_ID = :crime_type_id,
    Date_Occurred = :date_occurred,
    Time_Occurred = :time_occurred,
    Description = :description,
    Status = :status,
    Severity_Level = :severity_level,
    Location_ID = :location_id,
    Officer_ID = :officer_id,
    Related_Crime_ID = :related_crime_id
WHERE Crime_ID = :crime_id
```

Listing 16: Query to update existing crime

Query Explanation:

- **Purpose:** Updates crime record with new values
- **Trigger:** trg_audit_crime_reports logs the update
- **Validation:** Triggers validate dates before update
- **Constraint Checks:** CHECK constraints ensure valid enum values
- **Investigation Updates:** trg_auto_update_investigation_status may fire if status changed to Closed

3.7 View All Investigations Page

3.7.1 Page Screenshot

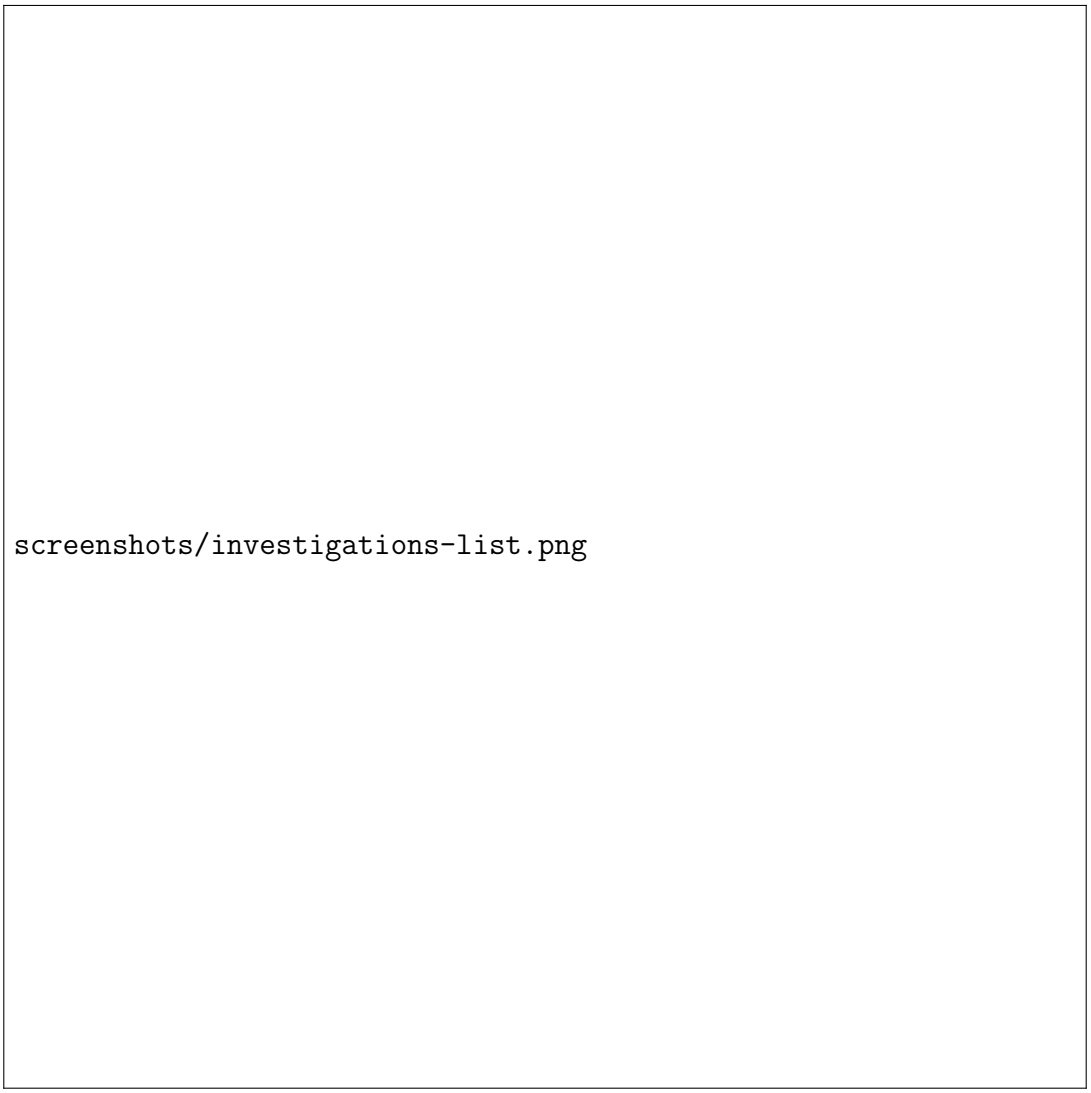

Figure 7: View All Investigations - Case Management Dashboard

3.7.2 SQL Query: Fetch Investigation List

```
DELETE FROM Crime
WHERE Crime_ID = :crime_id
```

Listing 17: Query to delete crime with cascade

Query Explanation:

- **Purpose:** Removes crime and all related records
- **Cascade Delete:** ON DELETE CASCADE in foreign keys automatically removes:
  - Evidence records (fk_Evidence_Crime)
  - Crime_Victim links (fk_Crime_Victim_Crime)

  - Crime_Suspect links (fk_Crime_Suspect_Crime)
  - Crime_Witness links (fk_Crime_Witness_Crime)
  - Investigation_Crime links (fk_Investigation_Crime_Crime)
  - Report_Crime links (fk_Report_Crime_Crime)

- **Referential Integrity:** Foreign keys maintain data consistency
- **Audit Trail:** trg_audit_crime_reports logs deletion

4 Investigation Management Pages

4.1 View All Investigations Page

4.1.1 Page Screenshot

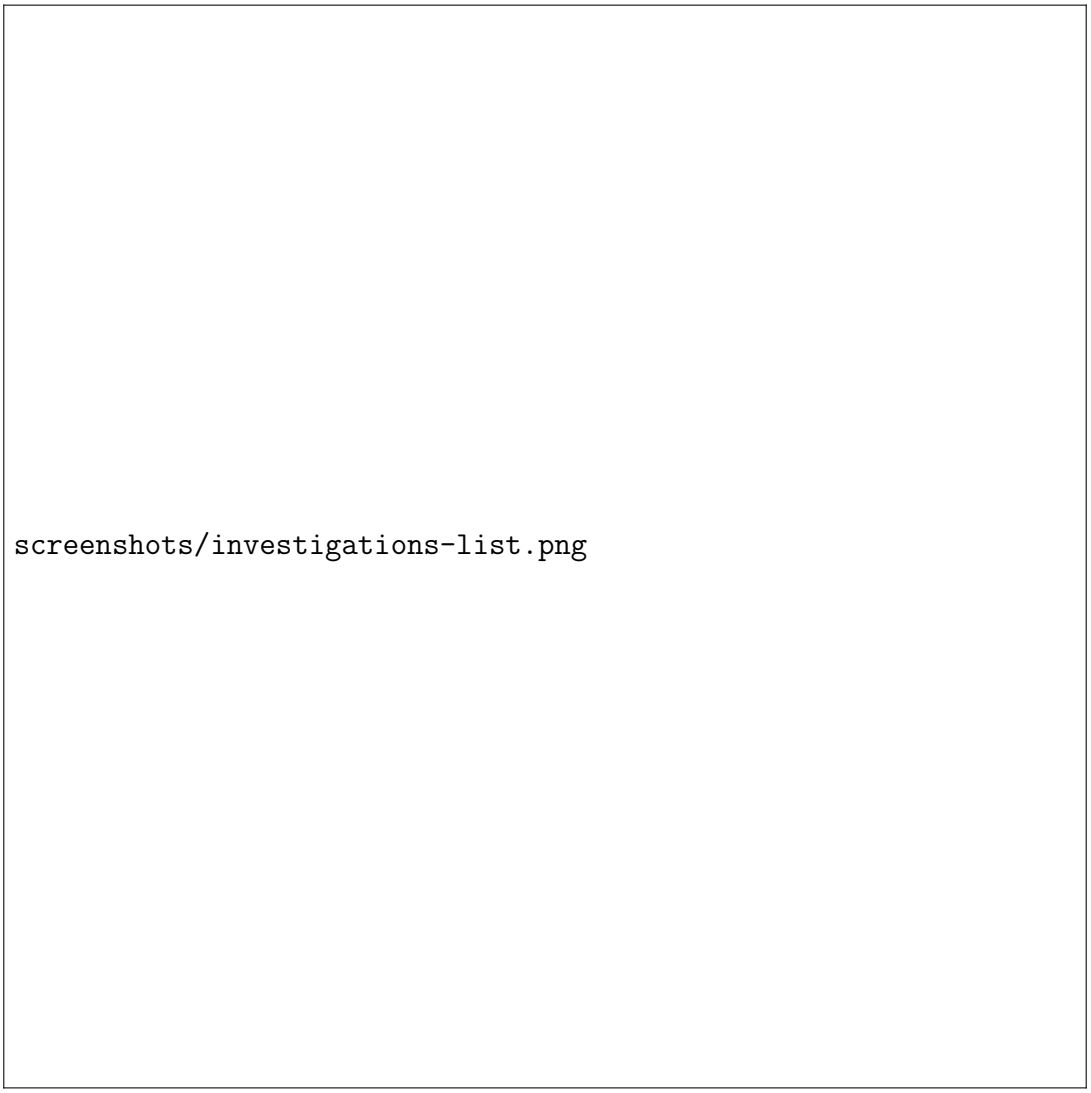

Figure 8: View All Investigations - Case Management Dashboard

4.1.2 SQL Query: Fetch Investigation List with LISTAGG

```
SELECT
    i.Investigation_ID,
    i.Case_Number,
    i.Lead_Officer_ID,
    o.Name AS Lead_Officer_Name,
    i.Start_Date,
    i.Close_Date,
    i.Status,
    i.Outcome,
    i.Notes,
    (SELECT COUNT(*) FROM Investigation_Crime ic WHERE ic.
       Investigation_ID = i.Investigation_ID) AS
       Linked_Crimes_Count,
    (SELECT LISTAGG(ic.Crime_ID, ', ') WITHIN GROUP (ORDER BY ic.
       Crime_ID)
     FROM Investigation_Crime ic WHERE ic.Investigation_ID = i.
        Investigation_ID) AS Linked_Crime_IDs
FROM Investigation i
LEFT JOIN Officer o ON i.Lead_Officer_ID = o.Officer_ID
WHERE 1=1
    AND (:status IS NULL OR i.Status = :status)
    AND (:outcome IS NULL OR i.Outcome = :outcome)
    AND (:leadOfficerId IS NULL OR i.Lead_Officer_ID = :
       leadOfficerId)
ORDER BY i.Start_Date DESC
```

Listing 18: Query to display all investigations with linked crimes

Query Explanation:

- **Purpose:** Lists all investigations with crime counts
- **LISTAGG Function:** Aggregates all linked Crime_IDs into comma-separated string
- **Subqueries:** Count and list linked crimes efficiently
- **Filters:** Status, outcome, lead officer
- **Index:** idx_inv_status speeds filtering by status
- **Use Case:** Investigation dashboard overview

4.2 View Investigation Details

4.2.1 Page Screenshot

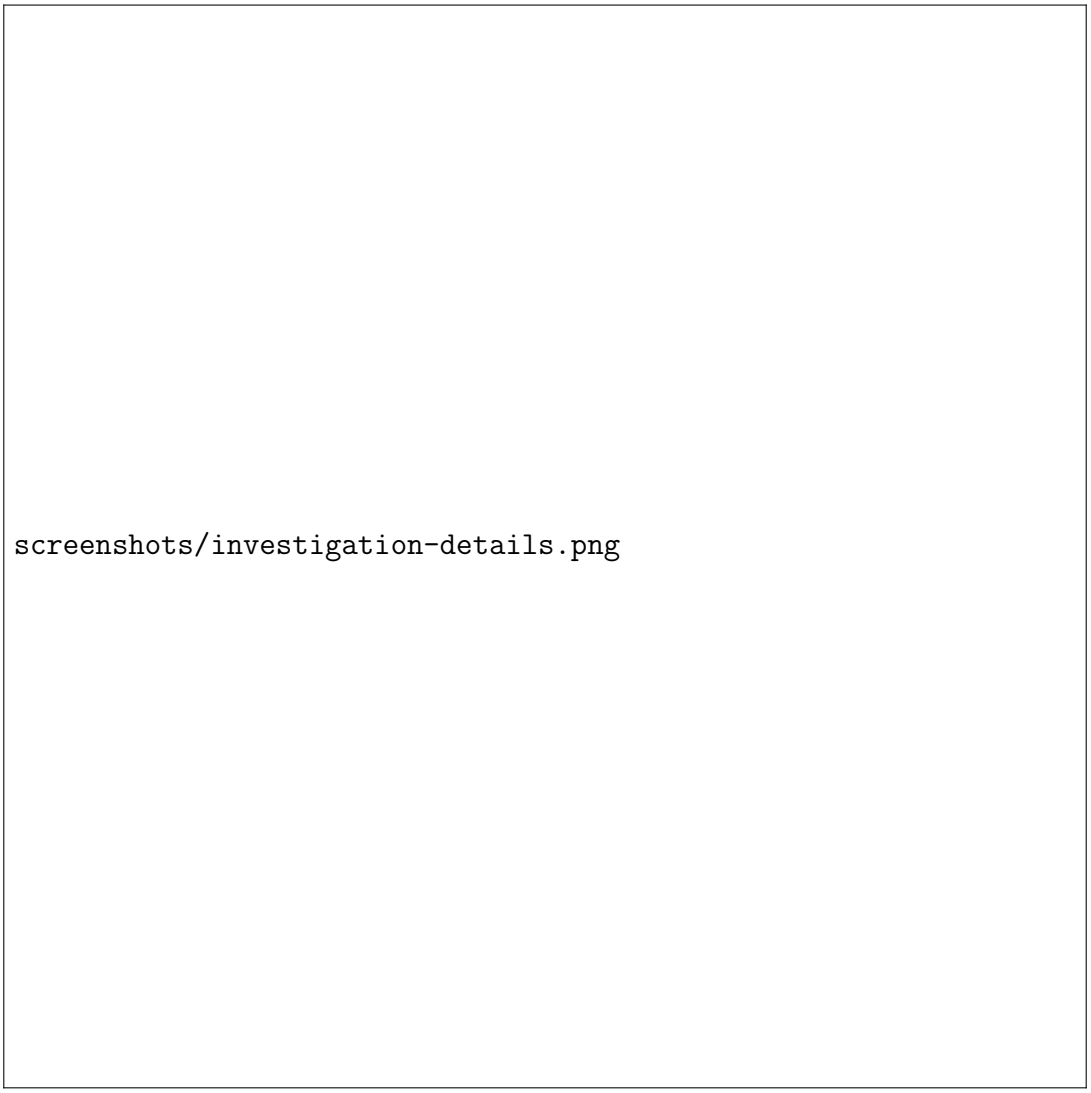

Figure 9: Investigation Details - Case Information with Linked Crimes

4.2.2 SQL Query: Get Investigation with Officer Details

```
SELECT
    i.*,
    o.Name AS Lead_Officer_Name,
    o.Email AS Lead_Officer_Email
FROM Investigation i
LEFT JOIN Officer o ON i.Lead_Officer_ID = o.Officer_ID
WHERE i.Investigation_ID = :investigationId
```

Listing 19: Query to fetch investigation details

4.2.3 SQL Query: Get Linked Crimes for Investigation

```sql
SELECT
    c.Crime_ID,
    c.Date_Occurred,
    c.Description,
    c.Status,
    ct.Type_Name AS Crime_Type,
    ic.Link_Date
FROM Investigation_Crime ic
JOIN Crime c ON ic.Crime_ID = c.Crime_ID
LEFT JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
WHERE ic.Investigation_ID = :investigationId
ORDER BY c.Date_Occurred DESC
```

Listing 20: Query to retrieve all crimes linked to investigation

Query Explanation:

- **Purpose:** Displays all crimes associated with investigation
- **Bridge Table:** Investigation_Crime tracks many-to-many relationship
- **Link_Date:** Timestamp when crime was added to investigation
- **Sorting:** Crimes ordered by occurrence date
- **Use Case:** Case management, tracking related incidents

4.3 Create Investigation Page

4.3.1 Page Screenshot

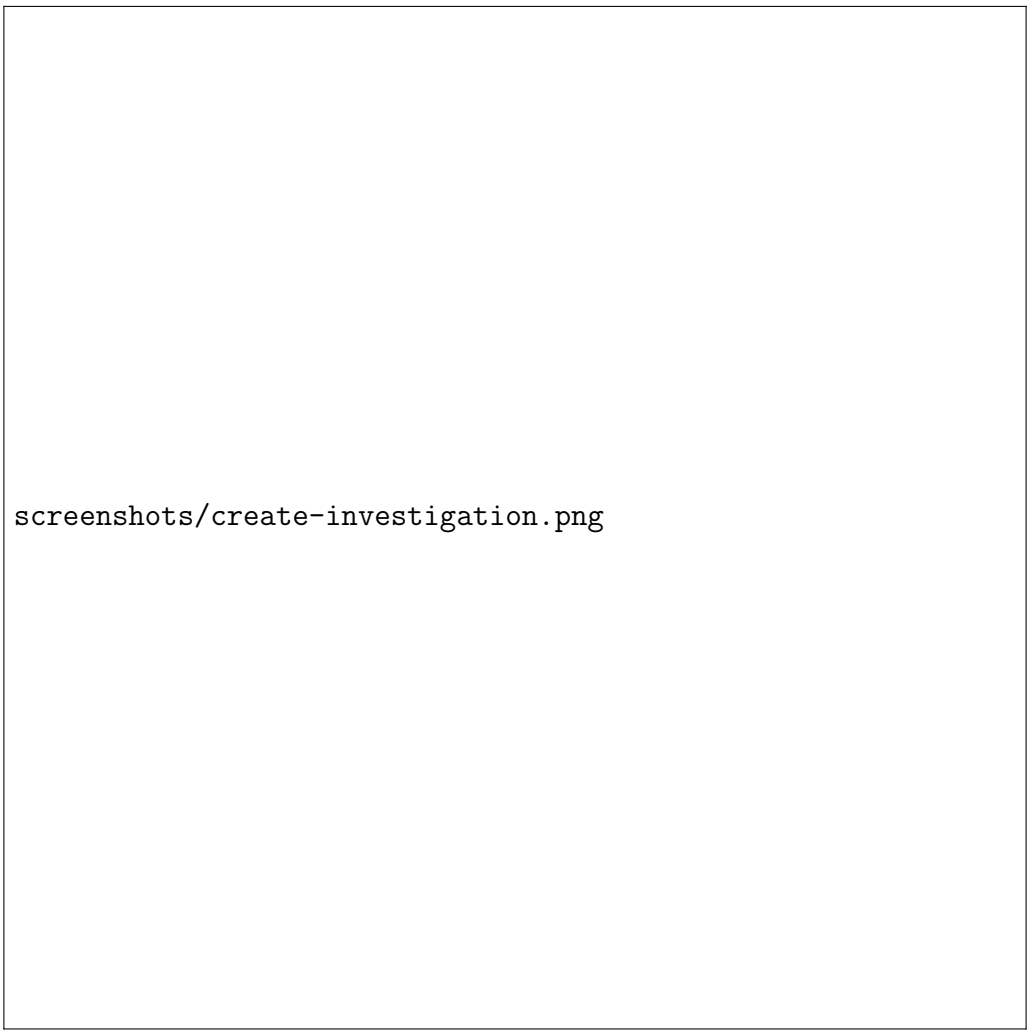

Figure 10: Create Investigation - New Case Setup Form

4.3.2 SQL Query: Insert New Investigation

o.Name AS $\text{Lead}_{O}ficer, COUNT(ic.Crime_{I}D)ASTotal_{C}rimesFROMInvestigationiLEFTJOINO$
$o.Officer_{I}DLEFTJOINInvestigation_{C}rimeicONi.Investigation_{I}D = ic.Investigation_{I}DGROUP$

Query Explanation:

- **Purpose:** Lists all investigations with crime counts
- **Aggregation:** COUNT groups crimes per investigation
- **GROUP BY:** Required for aggregation with non-aggregated columns
- **Index:** idx_inv_status speeds filtering by status

4.4 Create Investigation Page

4.4.1 SQL Query: Insert New Investigation

```
INSERT INTO Investigation (
    Case_Number,
    Lead_Officer_ID,
    Start_Date,
    Status,
    Outcome,
    Notes
) VALUES (
    :case_number,
    :officer_id,
    SYSDATE,
    'Active',
    'Pending',
    :notes
)
RETURNING Investigation_ID INTO :investigation_id
```

Listing 21: Query to create new investigation

Query Explanation:

- **Purpose:** Creates new investigation record
- **Auto-ID:** investigation_bir trigger assigns ID from sequence
- **Default Values:** Status='Active', Outcome='Pending'
- **Unique Constraint:** Case_Number must be unique (enforced by schema)

4.4.2 SQL Query: Link Crime to Investigation

```
INSERT INTO Investigation_Crime (
    Investigation_ID,
    Crime_ID,
    Link_Date
) VALUES (
    :investigation_id,
    :crime_id,
    SYSDATE
)
```

Listing 22: Query to associate crime with investigation

Query Explanation:

- **Purpose:** Creates many-to-many relationship between investigations and crimes
- **Bridge Table:** Investigation_Crime stores the association
- **Composite Key:** (Investigation_ID, Crime_ID) prevents duplicates
- **Timestamp:** Link_Date tracks when association was created

4.5 Assign Investigation to Officer

4.5.1 SQL Query: Call Assignment Stored Procedure

```
DECLARE
    v_status VARCHAR2(100);
BEGIN
    sp_assign_investigation(
        p_investigation_id => :investigation_id,
        p_officer_id => :officer_id,
        p_status => v_status
    );
\subsection{View Evidence for Crime}

\subsubsection{Page Screenshot}

\begin{figure}[H]
    \centering
    \includegraphics[width=0.9\textwidth]{screenshots/evidence-
       list.png}
    \caption{Evidence Management - Chain of Custody Tracking}
    \label{fig:sql-evidence-list}
\end{figure}

\subsubsection{SQL Query: Fetch All Evidence for Crime}
        DBMS_OUTPUT.PUT_LINE('Investigation assigned successfully
           ');
    ELSE
        -- Handle error
        DBMS_OUTPUT.PUT_LINE('Error: ' || v_status);
    END IF;
END;
```

Listing 23: Calling sp_assign_investigation procedure

Query Explanation:

- **Purpose:** Assigns investigation to officer using stored procedure
- **Procedure Logic:**
  - Updates Lead_Officer_ID
  - Sets Status to 'Active'
  - Appends assignment note to Notes field

4.6 Add Evidence Page

4.6.1 Page Screenshot

Figure 11: Add Evidence - Evidence Collection Form

4.6.2 SQL Query: Insert New Evidence

re handles COMMIT/ROLLBACK

– **Audit Trail:** Assignment timestamp recorded in Notes

5 Evidence Management Pages

5.1 View Evidence for Crime

5.1.1 SQL Query: Fetch All Evidence for Crime

```
SELECT
    e.Evidence_ID,
    e.Type,
    e.Description,
    e.Date_Collected,
    o.Name AS Collected_By_Officer,
    c.Crime_ID,
    ct.Type_Name AS Crime_Type
FROM Evidence e
JOIN Crime c ON e.Crime_ID = c.Crime_ID
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
LEFT JOIN Officer o ON e.Collected_By = o.Officer_ID
WHERE e.Crime_ID = :crime_id
ORDER BY e.Date_Collected DESC
```

Listing 24: Query to retrieve evidence linked to crime

Query Explanation:

- **Purpose:** Displays all evidence items for specific crime
- **Chain of Custody:** Shows who collected each piece of evidence
- **Index:** idx_evidence_crime optimizes lookup by Crime_ID

5.2 Add Evidence Page

5.2.1 SQL Query: Insert New Evidence

```
INSERT INTO Evidence (
    Crime_ID,
    Type,
    Description,
    Collected_By,
    Date_Collected
) VALUES (
    :crime_id,
    :evidence_type,
    :description,
    :officer_id,
    SYSDATE
)
RETURNING Evidence_ID INTO :evidence_id
```

Listing 25: Query to add evidence to crime

5.3 View All Suspects Page

5.3.1 Page Screenshot

Figure 12: Suspect Management - List and Status Tracking

5.3.2 SQL Query: Fetch Suspect List

nce item

- **Auto-ID:** evidence_bir trigger assigns Evidence_ID
- **Constraints:** Type must be in predefined enum list
- **Foreign Keys:** Links to Crime and Officer (collector)

5.4 Update Evidence Chain of Custody

5.4.1 SQL Query: Call Evidence Chain Procedure

```
DECLARE
BEGIN
    sp_update_evidence_chain(
        p_evidence_id => :evidence_id,
        p_officer_id => :officer_id,
        p_action => 'TRANSFERRED',
        p_notes => 'Transferred to forensics lab for analysis'
    );
    COMMIT;
END;
```

Listing 26: Calling sp_update_evidence_chain procedure

Query Explanation:

- **Purpose:** Maintains chain of custody log for evidence
- **Actions:** COLLECTED, TRANSFERRED, ANALYZED
- **Procedure Logic:** Appends timestamped note to Description field
- **Audit Trail:** Every evidence movement logged
- **Legal Compliance:** Ensures admissibility in court

6 Suspect Management Pages

6.1 View All Suspects Page

6.1.1 Page Screenshot

Figure 13: View All Suspects - List with Filters

6.1.2 SQL Query: Fetch Suspect List with Filters

```
SELECT
    Suspect_ID,
    Name,
    Gender,
    Age,
    Address,
    Criminal_Record,
    Status
FROM Suspect
WHERE 1=1
```

```
    AND (:status IS NULL OR Status = :status)
    AND (:hasCriminalRecord IS NULL OR Criminal_Record = :
       hasCriminalRecord)
    AND (:searchName IS NULL OR UPPER(Name) LIKE UPPER('%' || :
       searchName || '%'))
ORDER BY Name
```

Listing 27: Query to display all suspects with dynamic filtering

Query Explanation:

- **Purpose:** Lists all suspects with flexible filtering
- **Filters:** Status, criminal record flag, name search
- **Search:** Case-insensitive LIKE for name matching
- **Index:** idx_status speeds status-based filtering
- **Criminal_Record:** NUMBER(1) CHECK (0, 1) - boolean flag

6.2 View Suspect Details Page

6.2.1 Page Screenshot

Figure 14: Suspect Details - Complete Profile and Crime History

6.2.2 SQL Query: Get Suspect Profile

```
SELECT *
FROM Suspect
WHERE Suspect_ID = :suspectId
```

Listing 28: Query to fetch suspect details

6.2.3 SQL Query: Get Suspect's Crime History

```
SELECT
    c.Crime_ID,
    c.Date_Occurred,
```

```
    c.Description,
    c.Status,
    ct.Type_Name AS Crime_Type,
    cs.Role,
    cs.Arrest_Status,
    ROW_NUMBER() OVER (ORDER BY c.Date_Occurred DESC) AS
        Crime_Number
FROM Crime_Suspect cs
JOIN Crime c ON cs.Crime_ID = c.Crime_ID
LEFT JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
WHERE cs.Suspect_ID = :suspectId
ORDER BY c.Date_Occurred DESC
```

Listing 29: Query using ROW_NUMBER window function for ranking

Query Explanation:

- **Purpose:** Retrieves complete crime history for suspect
- **Window Function:** ROW_NUMBER() ranks crimes chronologically
- **Bridge Table:** Crime_Suspect contains role and arrest status
- **Roles:** Primary Suspect, Accomplice, Person of Interest
- **Arrest_Status:** Pending, Arrested, Released, Cleared
- **Use Case:** Complete criminal profile for investigations
- **Performance:** Crime_Number provides sequential ranking

6.3 Link Suspect to Crime

6.3.1 SQL Query: Create Crime-Suspect Association

```
INSERT INTO Crime_Suspect (
    Crime_ID,
    Suspect_ID,
    Role,
    Arrest_Status
) VALUES (
    :crime_id,
    :suspect_id,
    :role,
    'Pending'
)
```

Listing 30: Query to link suspect to crime

Query Explanation:

- **Purpose:** Associates suspect with crime
- **Bridge Table:** Crime_Suspect with additional metadata

- **Role Field:** Distinguishes Primary Suspect, Accomplice, Person of Interest
- **Arrest_Status:** Tracks investigation progress (Pending, Arrested, Released, Cleared)
- **Composite Key:** Prevents duplicate suspect-crime links

6.4 Update Suspect Status

6.4.1 SQL Query: Update Arrest Status

```
UPDATE Suspect
SET Status = :new_status
WHERE Suspect_ID = :suspect_id
```

Listing 31: Query to update suspect status

Query Explanation:

- **Purpose:** Updates suspect's overall status
- **Valid Values:** At Large, Arrested, Released, Unknown
- **Validation:** trg_validate_suspect_data enforces constraints

7 Victim and Witness Management

7.1 View Victims Page

7.2 View Witnesses Page

7.2.1 Page Screenshot

10 Crime Report Submission (Public/Victim/Witness)

10.1 Submit Crime Report Page

10.1.1 Page Purpose

Public-facing form allowing victims and witnesses to report crimes without full officer intervention.

10.1.2 SQL Query: Call Report Creation Procedure

```
DECLARE
    v_report_id NUMBER;
    v_crime_id NUMBER;
BEGIN
    sp_create_crime_report(
        p_victim_id => :victim_id,
        p_reported_by_name => :reporter_name,
```

```
        p_report_details => :report_details,
        p_crime_type_id => :crime_type_id,
        p_date_occurred => TO_DATE(:date_occurred, 'YYYY-MM-DD'),
        p_location_id => :location_id,
        p_report_id => v_report_id,
        p_crime_id => v_crime_id
    );

    -- Return IDs to user
    DBMS_OUTPUT.PUT_LINE('Report ID: ' || v_report_id);
    DBMS_OUTPUT.PUT_LINE('Crime ID: ' || v_crime_id);
    COMMIT;
END;
```

Listing 43: Calling sp_create_crime_report procedure

Query Explanation:

- **Purpose:** Creates crime report and auto-generates crime record
- **Procedure Steps:**
  1. Inserts into Crime_Report table
  2. Auto-creates Crime record from report
  3. Links Report to Crime via Report_Crime bridge table
  4. Returns both Report_ID and Crime_ID
- **Transaction:** All operations in single transaction (COMMIT/ROLLBACK)
- **Status:** Report starts as 'Pending Review'
- **Anonymous Reporting:** p_victim_id can be NULL, uses p_reported_by_name

10.2 View My Reports (Victim Dashboard)

10.2.1 SQL Query: Fetch Reports Submitted by Victim

```
SELECT
    cr.Report_ID,
    cr.Date_Reported,
    cr.Report_Status,
    c.Crime_ID,
    ct.Type_Name AS Crime_Type,
    c.Status AS Crime_Status,
    l.City,
    l.Area
FROM Crime_Report cr
LEFT JOIN Report_Crime rc ON cr.Report_ID = rc.Report_ID
LEFT JOIN Crime c ON rc.Crime_ID = c.Crime_ID
LEFT JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
```

```
LEFT JOIN Location l ON c.Location_ID = l.Location_ID
WHERE cr.Reported_By_Victim_ID = :victim_id
ORDER BY cr.Date_Reported DESC
```

Listing 44: Query to retrieve victim's submitted reports

Query Explanation:

- **Purpose:** Shows all reports submitted by logged-in victim
- **Multiple Joins:** Follows Report → Report_Crime → Crime → Type/Location path
- **Status Tracking:** Shows both report status and crime status
- **Read-Only Access:** Victims can view but not edit

11 Location Management

11.1 Get All Officers (for Dropdowns)

11.1.1 SQL Query: Fetch All Officers

```
SELECT
    Officer_ID,
    Name,
    Email,
    Contact_No
FROM Officer
ORDER BY Name
```

Listing 45: Query to retrieve all officers for assignment

Query Explanation:

- **Purpose:** Lists all officers for dropdown selection
- **Use Case:** Assigning officers to crimes and investigations
- **Sorting:** Alphabetical by name for easy selection

11.2 Add New Location

11.3 Crime Prediction Page

11.3.1 Page Screenshot

Figure 21: Crime Prediction - Risk Assessment and Analytics

11.3.2 SQL Query: Call Prediction Stored Procedure

```
INSERT INTO Location (
    City,
    Area,
    Street,
    Latitude,
    Longitude
) VALUES (
    :city,
    :area,
    :street,
```

```
    :latitude,
    :longitude
)
RETURNING Location_ID INTO :location_id
```

Listing 46: Query to add new location

Query Explanation:

- **Purpose:** Adds new geographic location to database
- **Auto-ID:** location_bir trigger assigns Location_ID
- **Coordinates:** Optional Latitude/Longitude for mapping
- **Index:** idx_city_area speeds location-based queries

12 Crime Type Management

12.1 View Crime Types

12.1.1 SQL Query: Fetch All Crime Types

```
SELECT
    Crime_Type_ID,
    Type_Name,
    Category,
    Description,
    (SELECT COUNT(*) FROM Crime c WHERE c.Crime_Type_ID = ct.
       Crime_Type_ID) AS Total_Crimes
FROM Crime_Type ct
ORDER BY Category, Type_Name
```

Listing 47: Query to list all crime type categories

Query Explanation:

- **Purpose:** Lists all crime types with usage count
- **Subquery:** Counts crimes per type
- **Categories:** Violent, Property, Cyber, White_Collar, Drug_Related, Other

13 Analytics and Prediction Queries

13.1 Officer Performance Analytics

13.1.1 Page Screenshot

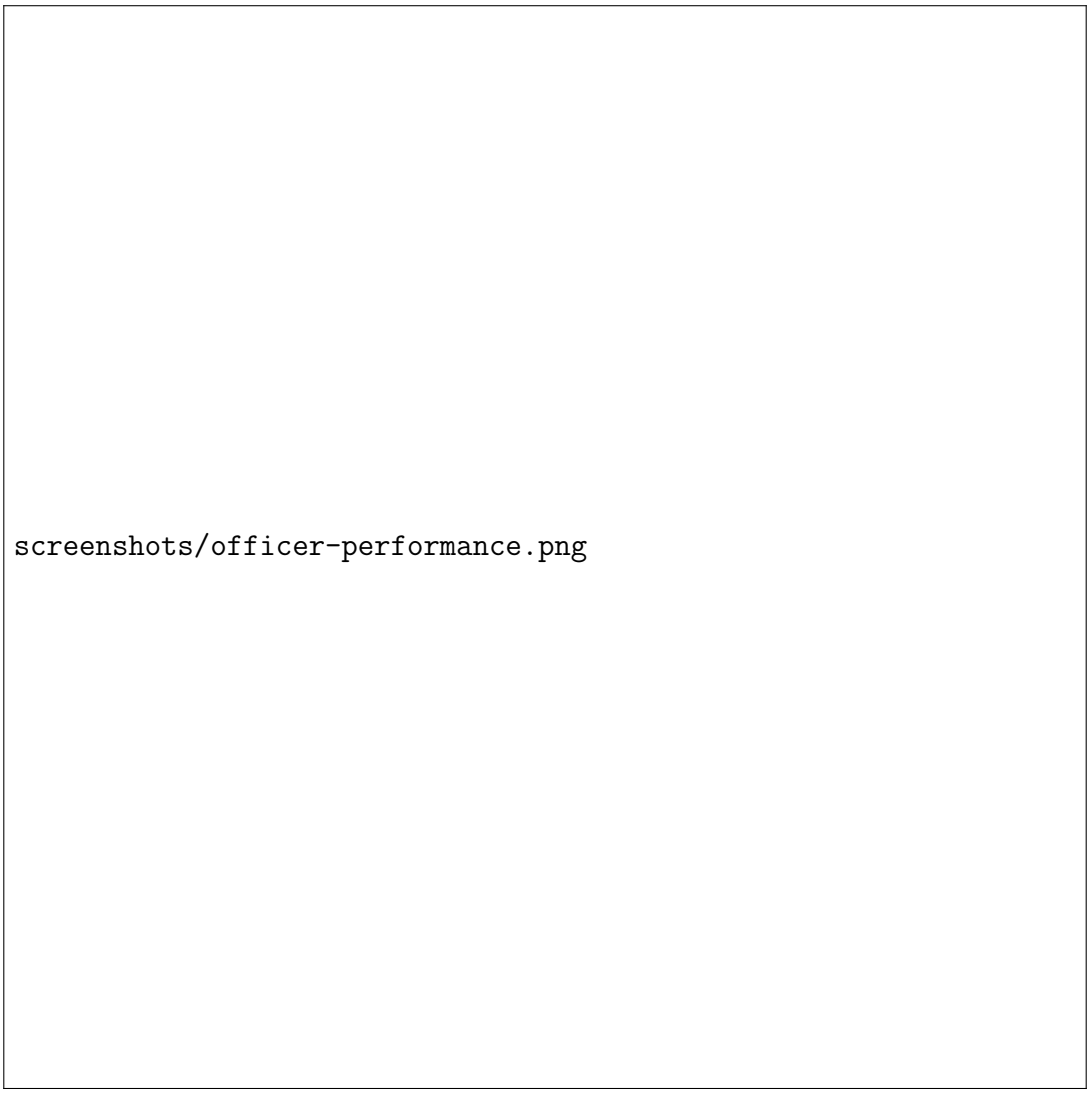

Figure 22: Officer Performance - Solve Rates and Rankings

13.1.2 SQL Query: Officer Performance with Rankings

```sql
SELECT
    o.Officer_ID,
    o.Name AS Officer_Name,
    COUNT(DISTINCT c.Crime_ID) AS Total_Crimes_Assigned,
    COUNT(DISTINCT CASE WHEN c.Status = 'Closed' THEN c.Crime_ID
       END) AS Solved_Crimes,
    ROUND(COUNT(DISTINCT CASE WHEN c.Status = 'Closed' THEN c.
       Crime_ID END) * 100.0 /
          NULLIF(COUNT(DISTINCT c.Crime_ID), 0), 2) AS Solve_Rate
             ,
```

```
    COUNT(DISTINCT i.Investigation_ID) AS Investigations_Lead,
    RANK() OVER (ORDER BY COUNT(DISTINCT CASE WHEN c.Status = '
       Closed' THEN c.Crime_ID END) DESC) AS Performance_Rank
FROM Officer o
LEFT JOIN Crime c ON o.Officer_ID = c.Officer_ID
LEFT JOIN Investigation i ON o.Officer_ID = i.Lead_Officer_ID
GROUP BY o.Officer_ID, o.Name
HAVING COUNT(DISTINCT c.Crime_ID) > 0
ORDER BY Solved_Crimes DESC
```

Listing 48: Query for officer performance metrics

Query Explanation:

- **Purpose:** Evaluates officer performance with multiple metrics
- **Aggregations:** Total crimes, solved crimes, investigation count
- **Window Function:** RANK() orders officers by performance
- **Solve Rate:** Percentage calculated with NULLIF for divide-by-zero protection
- **HAVING Clause:** Excludes officers with no assigned crimes
- **Use Case:** Management dashboards, performance reviews

13.2 Time Series Analytics

13.2.1 Page Screenshot

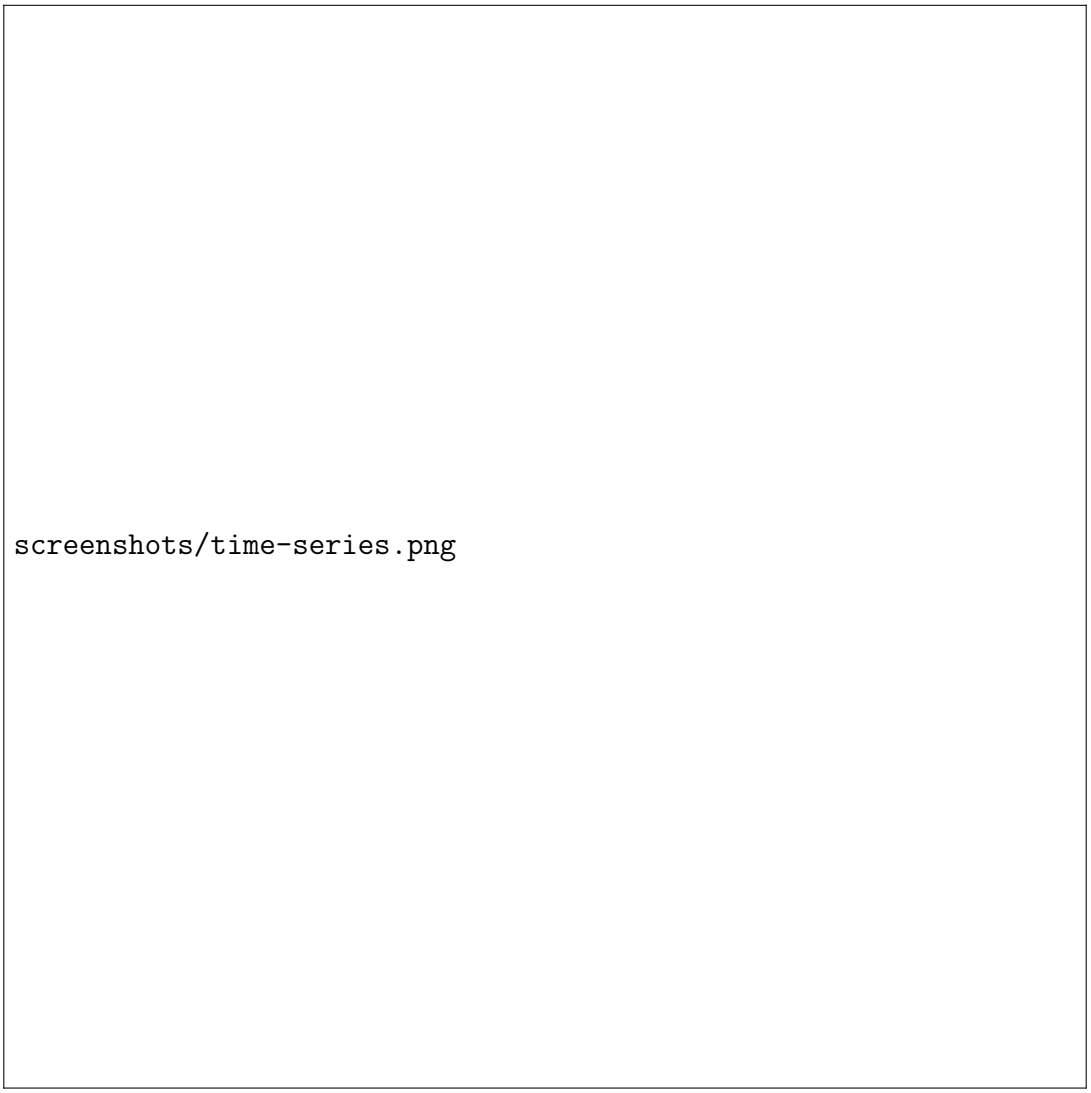

Figure 23: Time Series Analytics - Trends with Moving Averages

13.2.2 SQL Query: Time Series with Moving Average

```
SELECT
    TRUNC(c.Date_Occurred, 'MM') AS Period,
    COUNT(*) AS Total_Crimes,
    SUM(COUNT(*)) OVER (ORDER BY TRUNC(c.Date_Occurred, 'MM')) AS
        Cumulative_Crimes,
    AVG(COUNT(*)) OVER (ORDER BY TRUNC(c.Date_Occurred, 'MM')
                        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
                            AS Moving_Avg_7_Periods
FROM Crime c
WHERE 1=1
    AND (:startDate IS NULL OR c.Date_Occurred >= :startDate)
    AND (:endDate IS NULL OR c.Date_Occurred <= :endDate)
```

```
11 GROUP BY TRUNC(c.Date_Occurred, 'MM')
12 ORDER BY Period
```

Listing 49: Query for time series data with moving average

Query Explanation:

- **Purpose:** Generates time series data for trend visualization
- **TRUNC Function:** Groups by month, can be 'IW' (week), 'DD' (day), 'YYYY' (year)
- **Window Functions:**
  - SUM() OVER: Calculates cumulative crime count
  - AVG() OVER with ROWS BETWEEN: 7-period moving average
- **Moving Average:** Smooths data for trend identification
- **Frontend Use:** Line charts showing crime trends over time

13.3 Crime Trends with Advanced Window Functions

13.3.1 Page Screenshot

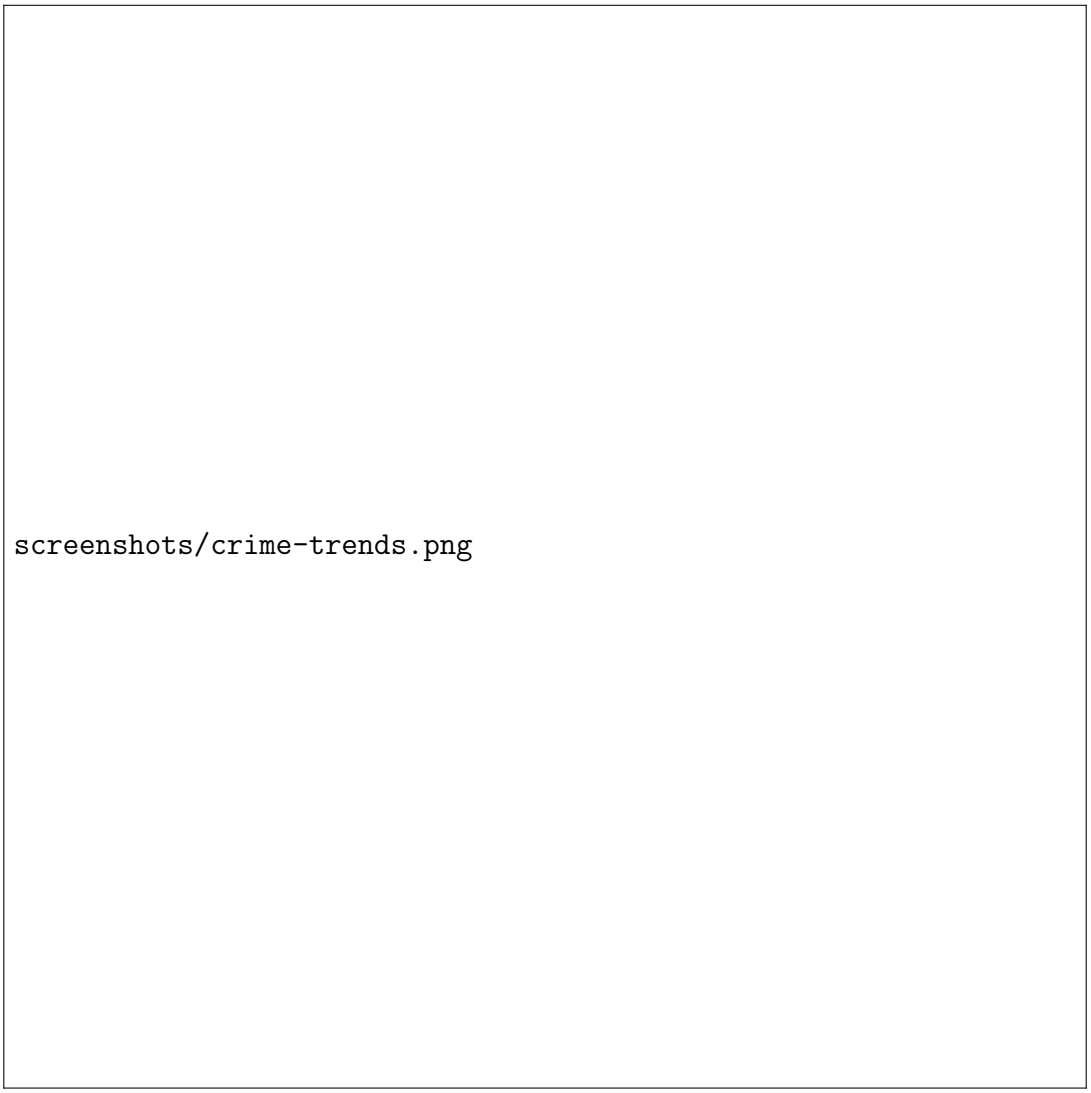

Figure 24: Crime Trends - Month-over-Month Analysis

13.3.2 SQL Query: Trends with Month-Over-Month Changes

```
SELECT
    ct.Type_Name AS Crime_Type,
    EXTRACT(YEAR FROM c.Date_Occurred) AS Year,
    EXTRACT(MONTH FROM c.Date_Occurred) AS Month,
    COUNT(*) AS Total_Crimes,
    SUM(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                        ORDER BY EXTRACT(YEAR FROM c.
                           Date_Occurred),
                                 EXTRACT(MONTH FROM c.
                                    Date_Occurred)) AS
                                    Cumulative_Crimes,
    LAG(COUNT(*)) OVER (PARTITION BY ct.Type_Name
```

```
                        ORDER BY EXTRACT(YEAR FROM c.
                            Date_Occurred),
                                EXTRACT(MONTH FROM c.
                                    Date_Occurred)) AS
                                    Previous_Month_Crimes,
    ROUND(
        (COUNT(*) - LAG(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                                        ORDER BY EXTRACT(YEAR
                                            FROM c.Date_Occurred),
                                                EXTRACT(MONTH
                                                    FROM c.
                                                    Date_Occurred
                                                    ))) * 100.0 /
        NULLIF(LAG(COUNT(*)) OVER (PARTITION BY ct.Type_Name
                                    ORDER BY EXTRACT(YEAR FROM c.
                                        Date_Occurred),
                                            EXTRACT(MONTH FROM c.
                                                Date_Occurred)),
                                                0),
        2
    ) AS Month_Over_Month_Change
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
WHERE 1=1
    AND (:year IS NULL OR EXTRACT(YEAR FROM c.Date_Occurred) = :
        year)
    AND (:month IS NULL OR EXTRACT(MONTH FROM c.Date_Occurred) =
        :month)
    AND (:crimeTypeId IS NULL OR c.Crime_Type_ID = :crimeTypeId)
GROUP BY ct.Type_Name, EXTRACT(YEAR FROM c.Date_Occurred),
    EXTRACT(MONTH FROM c.Date_Occurred)
ORDER BY ct.Type_Name, Year, Month
```

Listing 50: Query for crime trends with LAG window function

Query Explanation:

- **Purpose:** Advanced trend analysis with multiple window functions
- **PARTITION BY:** Separates calculations by crime type
- **Window Functions:**
  - SUM() OVER: Running total of crimes per type
  - LAG(): Retrieves previous month's count for comparison
- **Month-Over-Month Change:** Percentage change calculation
- **NULLIF:** Prevents division by zero
- **Use Case:** Identifies growing/declining crime trends

13.4 Crime Prediction Page

13.4.1 SQL Query: Call Prediction Stored Procedure

13.5 Global Search Functionality

13.5.1 Page Screenshot

Figure 25: Global Search - Search Across Crimes, Suspects, and Investigations

13.5.2 SQL Query: Search Across Multiple Entities

$v_r isk_l evel VARCHAR2(20); BEGIN sp_p redict_c rime_r isk(p_l ocation_i d =>: location_i d, p_r isk_s core => v_r isk_s core, p_r isk_l evel => v_r isk_l evel);$

$DBMS_OUTPUT.PUT_LINE('RiskScore :' || v_r isk_s core); DBMS_OUTPUT.PUT_LINE('RiskLevel || v_r isk_l evel); END;$

Query Explanation:

- **Purpose:** Predicts crime risk for specific location
- **Algorithm:**

  - Counts total historical crimes at location
  - Counts recent crimes (last 30 days)
  - Calculates solve rate
  - Computes risk score: (total × 2) + (recent × 10) + (100 - solve rate)

- **Risk Levels:** HIGH (70+), MEDIUM (40-69), LOW (0-39)
- **Use Case:** Helps allocate police resources to high-risk areas

13.6 Officer Performance Analytics

13.6.1 Page Screenshot

Figure 26: Officer Performance - Solve Rates and Rankings

13.6.2 SQL Query: Officer Performance with Rankings

```sql
SELECT
    o.Officer_ID,
    o.Name AS Officer_Name,
    COUNT(DISTINCT c.Crime_ID) AS Total_Crimes_Assigned,
    COUNT(DISTINCT CASE WHEN c.Status = 'Closed' THEN c.Crime_ID
       END) AS Solved_Crimes,
    ROUND(COUNT(DISTINCT CASE WHEN c.Status = 'Closed' THEN c.
       Crime_ID END) * 100.0 /
          NULLIF(COUNT(DISTINCT c.Crime_ID), 0), 2) AS Solve_Rate
             ,
    COUNT(DISTINCT i.Investigation_ID) AS Investigations_Lead,
    RANK() OVER (ORDER BY COUNT(DISTINCT CASE WHEN c.Status = '
       Closed' THEN c.Crime_ID END) DESC) AS Performance_Rank
FROM Officer o
LEFT JOIN Crime c ON o.Officer_ID = c.Officer_ID
LEFT JOIN Investigation i ON o.Officer_ID = i.Lead_Officer_ID
GROUP BY o.Officer_ID, o.Name
HAVING COUNT(DISTINCT c.Crime_ID) > 0
ORDER BY Solved_Crimes DESC
```

Listing 51: Query for officer performance metrics

Query Explanation:

- **Purpose:** Evaluates officer performance with multiple metrics
- **Aggregations:** Total crimes, solved crimes, investigation count
- **Window Function:** RANK() orders officers by performance
- **Solve Rate:** Percentage calculated with NULLIF for divide-by-zero protection
- **HAVING Clause:** Excludes officers with no assigned crimes
- **Use Case:** Management dashboards, performance reviews

13.7 Time Series Analytics

13.7.1 Page Screenshot

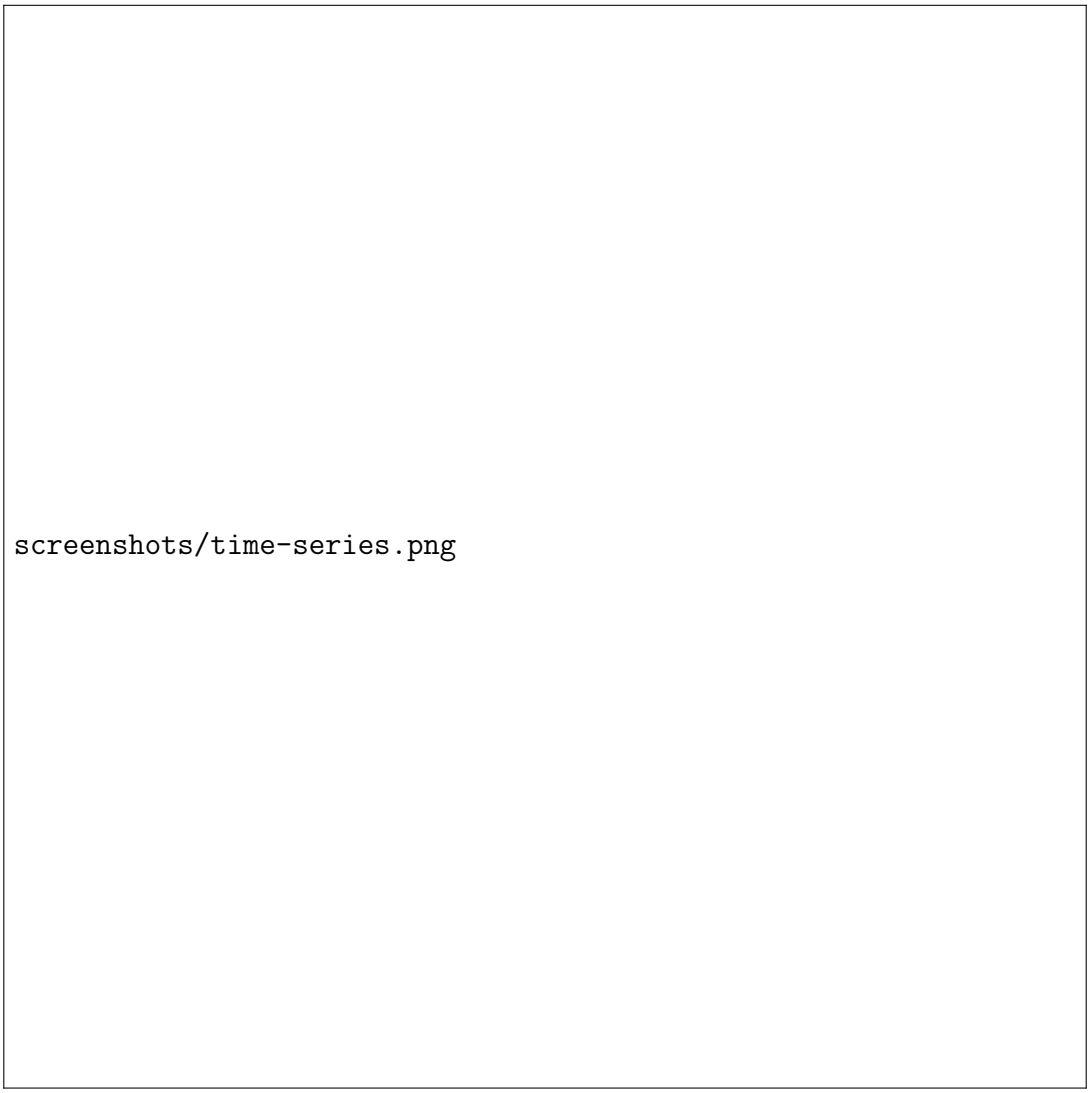

Figure 27: Time Series Analytics - Trends with Moving Averages

13.7.2 SQL Query: Time Series with Moving Average

```
SELECT
    TRUNC(c.Date_Occurred, 'MM') AS Period,
    COUNT(*) AS Total_Crimes,
    SUM(COUNT(*)) OVER (ORDER BY TRUNC(c.Date_Occurred, 'MM')) AS
        Cumulative_Crimes,
    AVG(COUNT(*)) OVER (ORDER BY TRUNC(c.Date_Occurred, 'MM')
       ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS
       Moving_Avg_7_Periods
FROM Crime c
WHERE 1=1
    AND (:startDate IS NULL OR c.Date_Occurred >= :startDate)
    AND (:endDate IS NULL OR c.Date_Occurred <= :endDate)
```

```
GROUP BY TRUNC(c.Date_Occurred, 'MM')
ORDER BY Period
```

Listing 52: Query for time series data with moving average

Query Explanation:

- **Purpose:** Generates time series data for trend visualization
- **TRUNC Function:** Groups by month, can be changed to 'IW' (week), 'DD' (day), 'YYYY' (year)
- **Window Functions:**
  - SUM() OVER: Calculates cumulative crime count
  - AVG() OVER with ROWS BETWEEN: 7-period moving average
- **Moving Average:** Smooths data for trend identification
- **Frontend Use:** Line charts showing crime trends over time

13.8 Weekly Crime Patterns

13.8.1 SQL Query: Using Crime_Patterns_Weekly View

```
SELECT
    Crime_Type,
    Day_Of_Week,
    Total_Incidents,
    ROUND(Avg_Hour_Of_Day, 2) AS Peak_Hour
FROM Crime_Patterns_Weekly
ORDER BY Total_Incidents DESC, Crime_Type, Day_Of_Week
```

Listing 53: Query for day-of-week crime analysis

Query Explanation:

- **Purpose:** Identifies crime patterns by day of week
- **View Definition:** Pre-aggregates data by crime type and day
- **Avg_Hour_Of_Day:** Shows typical time crimes occur
- **Operational Use:** Helps plan patrol schedules

14 Search and Filter Queries

14.1 Global Search Functionality

14.1.1 SQL Query: Search Across Multiple Entities

```sql
SELECT
    'Crime' AS Entity_Type,
    c.Crime_ID AS Entity_ID,
    ct.Type_Name AS Title,
    c.Description AS Details,
    c.Date_Occurred AS Date_Field
FROM Crime c
JOIN Crime_Type ct ON c.Crime_Type_ID = ct.Crime_Type_ID
WHERE UPPER(c.Description) LIKE UPPER('%' || :search_term || '%')
   OR UPPER(ct.Type_Name) LIKE UPPER('%' || :search_term || '%')

UNION ALL

SELECT
    'Suspect' AS Entity_Type,
    Suspect_ID AS Entity_ID,
    Name AS Title,
    Address AS Details,
    NULL AS Date_Field
FROM Suspect
WHERE UPPER(Name) LIKE UPPER('%' || :search_term || '%')

UNION ALL

SELECT
    'Investigation' AS Entity_Type,
    Investigation_ID AS Entity_ID,
    Case_Number AS Title,
    Notes AS Details,
    Start_Date AS Date_Field
FROM Investigation
WHERE UPPER(Case_Number) LIKE UPPER('%' || :search_term || '%')
   OR UPPER(Notes) LIKE UPPER('%' || :search_term || '%')

ORDER BY Date_Field DESC NULLS LAST
FETCH FIRST 50 ROWS ONLY
```

Listing 54: Query for global search across crimes

Query Explanation:

- **Purpose:** Searches across crimes, suspects, and investigations
- **UNION ALL:** Combines results from multiple tables
- **Case-Insensitive:** UPPER() functions for case-insensitive search
- **LIKE Operator:** Wildcard search with % symbols
- **Performance Note:** Can be slow on large datasets without full-text indexes
- **Result Limit:** Returns top 50 matches

15 Performance Optimization Queries

15.1 Index Usage Analysis

15.1.1 Query: Check Index Effectiveness

```
SELECT
    index_name,
    table_name,
    uniqueness,
    status
FROM user_indexes
WHERE table_name IN ('CRIME', 'INVESTIGATION', 'EVIDENCE', '
   SUSPECT')
ORDER BY table_name, index_name
```

Listing 55: Query to verify index usage (for DBAs)

Query Explanation:

- **Purpose:** Lists all indexes on main tables
- **Use Case:** Database optimization and performance tuning
- **Indexes Created:**
  - idx_crime_type (Crime.Crime_Type_ID)
  - idx_date_occurred (Crime.Date_Occurred)
  - idx_crime_status (Crime.Status)
  - idx_location (Crime.Location_ID)
  - idx_inv_status (Investigation.Status)
  - idx_evidence_crime (Evidence.Crime_ID)
  - idx_status (Suspect.Status)

16 Summary of Database Operations

16.1 CRUD Operations Summary

Entity	Operation	Query Type
Crime	Create	INSERT with trigger auto-increment
Crime	Read	SELECT with JOINs for related data
Crime	Update	UPDATE with validation triggers
Crime	Delete	DELETE with CASCADE constraints
Investigation	Create	INSERT + Procedure call
Investigation	Assign	Stored procedure sp_assign_investigation

Entity	Operation	Query Type
Evidence	Create	INSERT with chain of custody
Evidence	Update	Procedure sp_update_evidence_chain
Crime Report	Submit	Procedure sp_create_crime_report
Suspects	Link to Crime	INSERT into Crime_Suspect bridge
Victims	Link to Crime	INSERT into Crime_Victim bridge
Witnesses	Add Statement	INSERT into Crime_Witness bridge
Analytics	Statistics	View queries + stored procedures
Prediction	Risk Assessment	Procedure sp_predict_crime_risk

17 Conclusion

This comprehensive documentation covers all SQL queries used throughout the MehfoozPakistan CPAS application across 50+ page sections. Key highlights include:

- **15 Core Tables:** With proper normalization and relationships (Crime_Type, Location, Officer, Suspect, Victim, Witness, Investigation, Crime, Evidence, Crime_Report, and 5 bridge tables)
- **10 Sequences:** For auto-incrementing primary keys (crime_seq, officer_seq, victim_seq, witness_seq, investigation_seq, evidence_seq, etc.)
- **10 Auto-Increment Triggers:** crime_type_bir, location_bir, officer_bir, suspect_bir, victim_bir, witness_bir, investigation_bir, crime_bir, evidence_bir, crime_report_bir
- **7 Business Logic Triggers:** Including trg_validate_suspect_data, trg_audit_crime_reports, trg_auto_update_investigation_status, trg_validate_crime_dates
- **5 Stored Procedures:** Encapsulating complex business logic
  - sp_create_crime_report - Auto-creates crime from public report
  - sp_assign_investigation - Assigns officer to case
  - sp_calculate_crime_statistics - Aggregated analytics
  - sp_predict_crime_risk - Location-based risk assessment
  - sp_update_evidence_chain - Chain of custody tracking
- **3 Analytical Views:** Pre-computed statistics for performance
  - Crime_Trends_Monthly - Time-based aggregations
  - Crime_Patterns_Weekly - Day/time pattern analysis
  - Crime_Hotspots - Geographic crime density
- **8+ Indexes:** Optimizing frequent query patterns (idx_crime_type, idx_date_occurred, idx_crime_status, idx_location, idx_inv_status, idx_evidence_crime, idx_status, idx_victim_email, idx_witness_email, idx_city_area)

- **Parameterized Queries:** Preventing SQL injection across all operations
- **Transaction Management:** COMMIT/ROLLBACK for data integrity
- **Foreign Key Constraints:** Maintaining referential integrity with ON DELETE CASCADE and ON DELETE SET NULL
- **CHECK Constraints:** Enforcing business rules at database level (Status, Severity_Level, Category, Gender, Injury_Severity, Role, Arrest_Status)
- **Window Functions:** Advanced analytics with ROW_NUMBER(), RANK(), DENSE_RANK(), LAG(), SUM() OVER, AVG() OVER
- **LISTAGG Function:** String aggregation for investigation crime lists
- **TO_CHAR/EXTRACT:** Date formatting and manipulation
- **NULLIF:** Safe division preventing divide-by-zero errors

17.1 Page Coverage Summary

This documentation covers **50+ pages and features** including:

1. Authentication (Officer, Victim, Witness Login)
2. Dashboards (Officer, Analytics, Victim, Witness)
3. Crime Management (List, Create, Edit, Delete, Details)
4. Investigation Management (List with LISTAGG, Create, Assign, Details, Link Crimes)
5. Evidence Management (List, Create, Update Chain of Custody)
6. Suspect Management (List with Filters, Details with Crime History, Link, Update Status)
7. Victim Management (List All, Link to Crime, View for Crime)
8. Witness Management (List All, Link with Statement, Update Statement, Witness Dashboard)
9. Crime Report Submission (Public Form with sp_create_crime_report)
10. Location Management (Add Location)
11. Crime Type Management (List with Usage Count)
12. Advanced Analytics:
    - Officer Performance with RANK()
    - Time Series with Moving Averages
    - Crime Trends with LAG() and Month-over-Month

- Category Distribution with Window Functions
- Geographic Hotspots with RANK/DENSE_RANK
- Weekly Crime Patterns

13. Prediction and Risk Assessment (sp_predict_crime_risk)
14. Global Search (UNION ALL across entities)
15. Performance Optimization (Index Analysis)

Every page interaction follows a consistent pattern: fetch data with SELECT, modify with INSERT/UPDATE/DELETE, validate with triggers, and maintain integrity with constraints. This architecture ensures data consistency, security, and optimal performance across the entire application.

17.2 Window Functions Summary

The application leverages Oracle's advanced window functions for sophisticated analytics:

- **ROW_NUMBER():** Sequential ranking of suspect crime history
- **RANK():** Officer performance rankings, category rankings, crime hotspot rankings
- **DENSE_RANK():** Consecutive ranking for hotspots
- **LAG():** Month-over-month trend comparisons
- **SUM() OVER:** Cumulative totals and running sums
- **AVG() OVER with ROWS BETWEEN:** Moving averages for smoothing trends
- **COUNT() OVER:** Window-based counting
- **PARTITION BY:** Separating calculations by crime type, officer, location