Crew Scheduling System - Comprehensive Technical Documentation

Executive Summary

The Crew Scheduling System is a production-ready SQL Server database solution designed for managing airline crew assignments while ensuring full compliance with FAA regulations (14 CFR Part 117 and 121.467). This system provides intelligent crew scheduling, real-time regulatory compliance monitoring, and comprehensive reporting capabilities.

Version: 2.0
Last Updated: November 2025
Database: SQL Server 2019+
Language: T-SQL

Table of Contents

---

1. Project Overview

1.1 Purpose

This system solves the complex problem of airline crew scheduling across multiple airports with varying Internet connectivity levels, while maintaining strict regulatory compliance and operational fairness.

1.2 Key Features

- **Regulatory Compliance**: Automatic validation against FAA flight time limitations
- **Dynamic Hour Tracking**: Real-time calculation from actual flight history
- **Intelligent Scheduling**: Automated crew assignment with validation
- **Comprehensive Reporting**: Operations, compliance, and payroll reports
- **Data Security**: Encrypted storage of sensitive information (SSN)
- **High Availability**: Architecture designed for 99.9999% uptime
- **Scalability**: Supports multiple airports and high transaction volumes

1.3 Stakeholders

Role	Responsibilities	System Access
Station Managers	Schedule crews for departing flights	Schedule crew, view availability
Flight Operations	Monitor active flights and crew	View flights, crew assignments
Compliance Department	Ensure regulatory adherence	View all data (read-only)
Human Resources	Manage crew information, payroll	Update crew info, view hours

Role	Responsibilities	System Access
Database Administrators	System maintenance, backups	Full administrative access

---

2. Requirements Analysis

2.1 Original Requirements (from zadanie.md)

Crew Requirements Per Flight

- **Pilots**: 2 minimum, with at least 1 senior (captain) - SeniorityID = 3
- **Cabin Crew**: 3 flight attendants minimum, with at least 1 senior

Tracked Metadata Per Crew Member:

- Full name (first and last)
- Social Security Number (encrypted)
- Hours flown in last 168 hours (7 days)
- Hours flown in last 672 hours (28 days)

- Hours flown in last 365 days
- Crew type (1 = Flight Attendant, 2 = Pilot)
- Crew seniority (1 = Trainee, 2 = Journeyman, 3 = Senior)
- Base airport location

Per Flight:

- Airline
- Flight number
- Departure city/airport
- Destination city/airport
- Flight duration (minutes)
- Scheduled and actual departure times
- Actual arrival time (added for accurate tracking)
- International vs domestic flag (added for FA limits)
- Current status (Scheduled/InFlight/Landed)

Required Capabilities

1. **Operational Redundancy**: Multiple server deployment support
2. **Crew Scheduling**: Query to assign crew to flights with compliance validation
3. **In-Flight Report**: Show all crew on planes currently in flight
4. **Compliance Report**: Identify crew exceeding or approaching hour limits
5. **Payroll Report**: Monthly hours worked per employee

2.2 Design Decisions

Dynamic Hour Calculation Instead of storing static hour counts that require trigger maintenance, hours are calculated dynamically from flight history using `fn_CalculateCrewHours`. This ensures:

- Always accurate calculations
- Proper time window handling
- No data inconsistency issues
- Simplified maintenance

International Flight Flag The `IsInternational` flag enables proper enforcement of FA duty time limits:

- Domestic: Maximum 14 consecutive hours
- International: Maximum 20 consecutive hours

Actual Arrival Tracking The `ActualArrival` field enables accurate calculation of:

- Duty time for flight attendants
- Rest time between flights
- Total time away from base

---

3. Database Architecture

3.1 Database Configuration

```
Name:             CrewSchedulingDB
Collation:        SQL_Latin1_General_CP1_CI_AS
Recovery Model:   FULL
Compatibility:    150 (SQL Server 2019)
```

3.2 Entity Relationship Diagram

Figure 1: Entity Relationship Diagram showing relationships between Airlines, Flights, Airports, CrewAssignments, and Crew tables.

3.3 Table Inventory

Core Tables (5)

1. **Airlines**: Airline company information
2. **Airports**: Airport/city locations

3. **Crew**: Crew member personal and professional information
4. **Flights**: Flight schedule and operational data
5. **CrewAssignments**: Junction table linking crew to flights

Lookup Tables (4)

6. **CrewTypes**: 1=Flight Attendant, 2=Pilot
7. **SeniorityLevels**: 1=Trainee, 2=Journeyman, 3=Senior
8. **FlightStatuses**: 1=Scheduled, 2=InFlight, 3=Landed
9. **Roles**: 1=Pilot, 2=Cabin

---

4. Data Structures

4.1 Table Definitions

4.1.1 Airlines

```sql
CREATE TABLE Airlines (
    AirlineID   INT IDENTITY(1,1) PRIMARY KEY,
    AirlineName NVARCHAR(100) NOT NULL,
    IATACode    NVARCHAR(3) UNIQUE NOT NULL
);
```

Purpose: Stores airline carrier information
Sample Data: AA (American), DL (Delta), UA (United)
Constraints: UNIQUE on IATACode to prevent duplicates

4.1.2 Airports

```sql
CREATE TABLE Airports (
    AirportID INT IDENTITY(1,1) PRIMARY KEY,
    City      NVARCHAR(100) NOT NULL,
    Country   NVARCHAR(100) NOT NULL,
    IATACode  NVARCHAR(3) UNIQUE NOT NULL
);
```

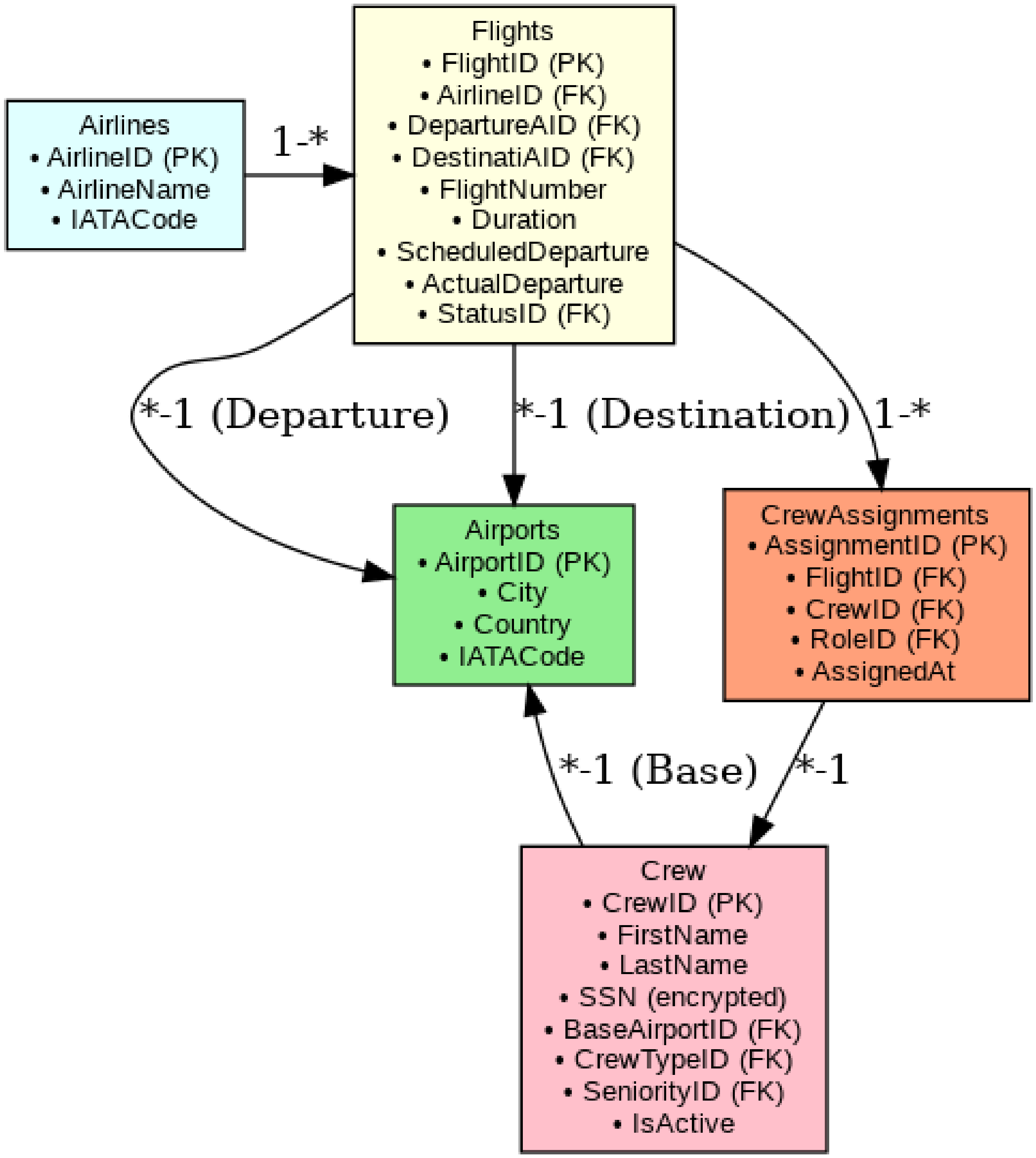

Figure 1: Entity Relationship Diagram

Purpose: Defines operational bases and flight endpoints
Sample Data: NYC, BTV, ORD, LAX, MIA, SEA, DEN, BOS, SFO, DFW
Business Rule: Crew can only be scheduled from their base airport

4.1.3 Crew

```
CREATE TABLE Crew (
    CrewID          INT IDENTITY(1,1) PRIMARY KEY,
    FirstName       NVARCHAR(50) NOT NULL,
    LastName        NVARCHAR(50) NOT NULL,
    SSN             VARBINARY(256) NOT NULL,  -- Encrypted
    BaseAirportID   INT NOT NULL,
    CrewTypeID      TINYINT NOT NULL,
    SeniorityID     TINYINT NOT NULL,
    HoursLast168    DECIMAL(6,2) DEFAULT 0 NOT NULL,  -- Legacy field
    HoursLast672    DECIMAL(6,2) DEFAULT 0 NOT NULL,  -- Legacy field
    HoursLast365Days DECIMAL(7,2) DEFAULT 0 NOT NULL, -- Legacy field
    IsActive        BIT DEFAULT 1 NOT NULL,
    FOREIGN KEY (BaseAirportID) REFERENCES Airports(AirportID),
    FOREIGN KEY (CrewTypeID) REFERENCES CrewTypes(CrewTypeID),
    FOREIGN KEY (SeniorityID) REFERENCES SeniorityLevels(SeniorityID)
);
```

Purpose: Stores crew member information and status
Security: SSN encrypted using AES-256 symmetric key
Note: HoursLast* fields are legacy - system now uses dynamic calculation via `fn_CalculateCrewHours`

Indexes:

- `IX_Crew_BaseAirportID` - Fast lookups by base location
- `IX_Crew_CrewTypeID` - Filter by pilot vs FA

4.1.4 Flights

```
CREATE TABLE Flights (
    FlightID            INT IDENTITY(1,1) PRIMARY KEY,
    AirlineID           INT NOT NULL,
    FlightNumber        NVARCHAR(10) NOT NULL,
    DepartureAirportID  INT NOT NULL,
    DestinationAirportID INT NOT NULL,
    FlightDuration      INT NOT NULL,  -- minutes
    ScheduledDeparture  DATETIME2 NOT NULL,
    ActualDeparture     DATETIME2 NULL,
    ActualArrival       DATETIME2 NULL,
    IsInternational     BIT DEFAULT 0 NOT NULL,
    StatusID            TINYINT NOT NULL,
    FOREIGN KEY (AirlineID) REFERENCES Airlines(AirlineID),
    FOREIGN KEY (DepartureAirportID) REFERENCES Airports(AirportID),
    FOREIGN KEY (DestinationAirportID) REFERENCES Airports(AirportID),
    FOREIGN KEY (StatusID) REFERENCES FlightStatuses(StatusID)
);
```

Purpose: Flight schedule and operational tracking
Time Tracking:

- `ScheduledDeparture`: Planning purposes
- `ActualDeparture`: Set when status changes to InFlight
- `ActualArrival`: Set when status changes to Landed

Indexes:

- `IX_Flights_DepartureAirportID` - Crew scheduling queries
- `IX_Flights_StatusID` - Filter by flight status
- `IX_Flights_ScheduledDeparture` - Time-based queries

4.1.5 CrewAssignments

```
CREATE TABLE CrewAssignments (
    AssignmentID INT IDENTITY(1,1) PRIMARY KEY,
    FlightID     INT NOT NULL,
    CrewID       INT NOT NULL,
    RoleID       TINYINT NOT NULL,
    AssignedAt   DATETIME2 DEFAULT GETDATE() NOT NULL,
    FOREIGN KEY (FlightID) REFERENCES Flights(FlightID),
    FOREIGN KEY (CrewID) REFERENCES Crew(CrewID),
    FOREIGN KEY (RoleID) REFERENCES Roles(RoleID)
);
```

Purpose: Links crew members to specific flights
Audit Trail: `AssignedAt` timestamp for scheduling history

Indexes:

- `IX_CrewAssignments_FlightID` - Get all crew for a flight
- `IX_CrewAssignments_CrewID` - Get all flights for crew member
- `IX_CrewAssignments_RoleID` - Filter by role
- `IX_CrewAssignments_AssignedAt` - Historical analysis

4.2 Sample Data Statistics

Entity	Count	Purpose
Airports	10	Major US hubs covering different regions
Airlines	10	Mix of legacy, low-cost, and regional carriers
Crew Members	50	20 pilots + 30 flight attendants across seniority levels
Flights	100	85 historical (landed) + 10 scheduled + 5 in-flight
CrewAssignments	425	Historical assignments for hour tracking

4.3 Edge Case Coverage in Sample Data

1. **Crew 1 & 16 (NYC Pilots)**: Assigned to 7 recent flights (61-67) to test approaching 60h/168h limit
2. **Crew 2 & 7 (Burlington)**: Assigned to 8 recent flights (68-75) to test 100h/672h limit
3. **International Flights 76-78**: 14h and 20h flights to test FA duty limits
4. **Flights 96-100**: Currently in-flight for Report 1 testing
5. **Flights 86-95**: Scheduled for `sp_ScheduleCrew` testing
6. **Various Rest Scenarios**: Crew with 6h, 9h, 12h gaps between flights

---

5. Business Logic

5.1 Functions

5.1.1 fn_CalculateCrewHours Signature:

```
CREATE FUNCTION fn_CalculateCrewHours (
    @CrewID INT,
    @HoursPeriod INT  -- 168, 672, or 8760 (365 days * 24)
)
RETURNS DECIMAL(7,2)
```

Purpose: Dynamically calculates total flight hours for a crew member within a specified time window.

Logic:

1. Calculate cutoff datetime: `DATEADD(HOUR, -@HoursPeriod, GETDATE())`
2. Sum FlightDuration from completed/in-flight assignments after cutoff
3. Return total hours as decimal

Example Usage:

```sql
-- Get hours in last 7 days for crew 1
SELECT dbo.fn_CalculateCrewHours(1, 168) AS Hours7Days;

-- Get hours in last 28 days for crew 5
SELECT dbo.fn_CalculateCrewHours(5, 672) AS Hours28Days;

-- Get hours in last year for crew 10
SELECT dbo.fn_CalculateCrewHours(10, 8760) AS Hours365Days;
```

Performance Considerations:

- Uses indexed `ActualDeparture` column
- Filters by `StatusID IN (2,3)` for completed flights only
- Optimized with covering indexes on CrewAssignments

5.1.2 fn_CheckHourLimits **Signature:**

```sql
CREATE FUNCTION fn_CheckHourLimits (@CrewID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        @CrewID AS CrewID,
        C.CrewTypeID,
        CT.CrewTypeName,
        dbo.fn_CalculateCrewHours(@CrewID, 168) AS Hours168,
        dbo.fn_CalculateCrewHours(@CrewID, 672) AS Hours672,
        dbo.fn_CalculateCrewHours(@CrewID, 8760) AS Hours365Days,
        CASE WHEN ... THEN 1 ELSE 0 END AS ExceedsLimits,
        ... AS LimitStatus
    FROM Crew C
    ...
)
```

Purpose: Checks if a pilot exceeds any FAA hour limitations per 14 CFR Part 117 and 121.467.

Returns: Table with columns:

- `Hours168`: Hours in last 168 hours (7 days)
- `Hours672`: Hours in last 672 hours (28 days)
- `Hours365Days`: Hours in last 365 days
- `ExceedsLimits`: 1 if any limit exceeded, 0 otherwise
- `LimitStatus`: Human-readable description of status

Pilot Limits Checked:

- 60 hours in 168 consecutive hours (7 days)
- 100 hours in 672 consecutive hours (28 days)
- 190 hours in 672 consecutive hours (alternate limit)
- 1,000 hours in 365 consecutive days (1 year)

Example Usage:

```sql
-- Check limits for crew 1
SELECT * FROM dbo.fn_CheckHourLimits(1);

-- Find all pilots exceeding limits
SELECT C.FirstName, C.LastName, HL.*
FROM Crew C
CROSS APPLY dbo.fn_CheckHourLimits(C.CrewID) HL
WHERE C.CrewTypeID = 2 AND HL.ExceedsLimits = 1;
```

5.1.3 fn_CalculateRestTime **Signature:**

```sql
CREATE FUNCTION fn_CalculateRestTime (
    @CrewID INT,
    @NewFlightID INT
)
RETURNS INT  -- Hours of rest, or -1 if no previous flight
```

Purpose: Calculates hours of rest between last completed flight and a proposed new flight.

Logic:

1. Find last flight end time (ActualArrival or calculated from duration)
2. Get proposed flight start time (ScheduledDeparture)
3. Calculate DATEDIFF in hours
4. Return -1 if no previous flight exists

Business Rules:

- Flight attendants require 9 consecutive hours rest between flights
- Used by `sp_ScheduleCrew` for validation
- Used by `fn_CheckFADutyLimits` for compliance

Example Usage:

```
-- Check rest time for crew 21 before flight 86
SELECT dbo.fn_CalculateRestTime(21, 86) AS RestHours;

-- Find crew with insufficient rest
SELECT C.CrewID, C.FirstName, C.LastName,
       dbo.fn_CalculateRestTime(C.CrewID, 86) AS RestHours
FROM Crew C
WHERE C.CrewTypeID = 1  -- FAs only
  AND dbo.fn_CalculateRestTime(C.CrewID, 86) BETWEEN 0 AND 8;
```

5.1.4 fn_CheckFADutyLimits   Signature:

```
CREATE FUNCTION fn_CheckFADutyLimits (
    @CrewID INT,
    @FlightID INT
)
RETURNS TABLE
```

Purpose: Checks if a flight attendant would exceed duty time limits or rest requirements for a specific flight.

Returns: Table with columns:

- `FlightHours`: Duration of flight in hours
- `ExceedsDutyLimit`: 1 if flight exceeds 14h (domestic) or 20h (international)
- `RestHoursSinceLastFlight`: Hours of rest since last flight
- `InsufficientRest`: 1 if less than 9 hours rest

Duty Time Limits:

- Domestic flights: 14 consecutive hours
- International flights: 20 consecutive hours
- Rest requirement: 9 consecutive hours between flights

Example Usage:

```
-- Check if crew 21 can work flight 86
SELECT * FROM dbo.fn_CheckFADutyLimits(21, 86);

-- Find FAs available for international flight 100
SELECT C.CrewID, C.FirstName, C.LastName, FDL.*
FROM Crew C
CROSS APPLY dbo.fn_CheckFADutyLimits(C.CrewID, 100) FDL
WHERE C.CrewTypeID = 1
  AND FDL.ExceedsDutyLimit = 0
  AND FDL.InsufficientRest = 0;
```

5.2 Stored Procedures

5.2.1 sp_ScheduleCrew   Signature:

```
CREATE PROCEDURE sp_ScheduleCrew @FlightID INT
```

Purpose: Intelligently assigns crew to a scheduled flight with full regulatory validation.

Algorithm:

1. **Validate Flight**
   - Verify flight exists and is in Scheduled status (StatusID = 1)
   - Check that crew is not already assigned
   - Get departure airport for crew matching
2. **Assign Pilots (2 required, 1 senior)**
   - Query `vw_AvailableCrew` for pilots at departure airport
   - Order by SeniorityID DESC (prefer seniors)
   - For each candidate:
     - Validate rest time   9 hours or no previous flight
     - Validate hour limits via `fn_CheckHourLimits`
     - Insert assignment with RoleID = 1 (Pilot)
   - Stop when 2 pilots assigned
3. **Assign Flight Attendants (3 required, 1 senior)**
   - Query `vw_AvailableCrew` for FAs at departure airport
   - Order by SeniorityID DESC
   - For each candidate:
     - Validate duty time via `fn_CheckFADutyLimits`
     - Validate rest time   9 hours
     - Insert assignment with RoleID = 2 (Cabin)
   - Stop when 3 FAs assigned
4. **Validate Composition**
   - Verify 2 pilots assigned (rollback if not)
   - Verify 1 senior pilot (rollback if not)
   - Verify 3 FAs assigned (rollback if not)
   - Verify 1 senior FA (rollback if not)
5. **Commit or Rollback**
   - If all validations pass: COMMIT
   - If any validation fails: ROLLBACK with error message

Error Handling:

- `RAISERROR` for missing flight, wrong status, or insufficient crew
- Full transaction support with automatic rollback
- Detailed error messages for troubleshooting

Example Usage:

```
-- Schedule crew for flight 86
EXEC sp_ScheduleCrew @FlightID = 86;

-- Output: "Crew scheduled successfully for FlightID 86: 2 pilots (1 senior), 3 cabin crew (1 senior)."

-- Verify assignment
SELECT * FROM vw_FlightCrew WHERE FlightID = 86;
```

Best Practices:

- Always check `vw_AvailableCrew` before scheduling
- Run during off-peak hours for better performance
- Review error messages if scheduling fails

5.2.2 sp_UpdateFlightStatus   Signature:

```
CREATE PROCEDURE sp_UpdateFlightStatus
    @FlightID INT,
    @NewStatus NVARCHAR(20)  -- 'Scheduled', 'InFlight', or 'Landed'
```

Purpose: Updates flight status with proper timestamp tracking and validation.

Logic:

1. **Validate Status**
   - Map status name to ID (1=Scheduled, 2=InFlight, 3=Landed)
   - Verify flight exists
2. **Update Flight**
   - Set StatusID to new value
   - If changing to InFlight: Set ActualDeparture to GETDATE() if NULL
   - If changing to Landed: Set ActualArrival to GETDATE() if NULL
3. **Commit Transaction**
   - Print confirmation message
   - Handle errors with RAISERROR

Example Usage:

```
-- Flight takes off
EXEC sp_UpdateFlightStatus @FlightID = 86, @NewStatus = 'InFlight';

-- Flight lands
EXEC sp_UpdateFlightStatus @FlightID = 86, @NewStatus = 'Landed';

-- Check updated status
SELECT FlightID, FlightNumber, StatusID, ActualDeparture, ActualArrival
FROM Flights
WHERE FlightID = 86;
```

5.3 Views

5.3.1 vw_AvailableCrew

Purpose: Shows only active crew members who are within regulatory hour limits.

Columns:

- CrewID, FirstName, LastName
- BaseCity (from Airports join)
- CrewTypeName, SeniorityName
- Hours168, Hours672, Hours365Days (calculated)
- ExceedsLimits (0 for all rows in view)
- LimitStatus (descriptive text)

Filter Criteria:

- `IsActive = 1`
- `ExceedsLimits = 0` (from fn_CheckHourLimits)

Usage:

```
-- View all available crew at NYC
SELECT * FROM vw_AvailableCrew
WHERE BaseCity = 'New York';

-- Count available senior pilots
SELECT COUNT(*) AS AvailableSeniorPilots
FROM vw_AvailableCrew
WHERE CrewTypeName = 'Pilot' AND SeniorityName = 'Senior';
```

5.3.2 vw_FlightCrew

Purpose: Shows crew assignments with full flight and crew details.

Columns:

- FlightID, FlightNumber, ScheduledDeparture, ActualDeparture
- DepartureCity, DestinationCity
- CrewID, FirstName, LastName
- RoleName, CrewTypeName, SeniorityName
- FlightHours (duration)

Usage:

```sql
-- View crew for a specific flight
SELECT * FROM vw_FlightCrew
WHERE FlightID = 86
ORDER BY RoleID DESC, SeniorityID DESC;

-- View all assignments for a crew member
SELECT * FROM vw_FlightCrew
WHERE CrewID = 1
ORDER BY ScheduledDeparture DESC;
```

---

6. Regulatory Compliance

6.1 14 CFR Part 117 and 121.467 Overview

The Federal Aviation Administration (FAA) regulates flight and duty time limitations to prevent fatigue-related accidents.

Pilot Flight Time Limitations (14 CFR 121.467)

Regulation	Limit	Time Period	Implementation Function
121.467(b)(1)	60 hours	168 consecutive hours (7 days)	`fn_CheckHourLimits` checks Hours168 60
121.467(b)(2)	100 hours	672 consecutive hours (28 days)	`fn_CheckHourLimits` checks Hours672 100
121.467(b)(3)	190 hours	672 consecutive hours	`fn_CheckHourLimits` checks Hours672 190
121.467(b)(4)	1,000 hours	365 consecutive days	`fn_CheckHourLimits` checks Hours365Days 1000

Flight Attendant Duty Time Limitations (14 CFR Part 117)

Regulation	Limit	Flight Type	Implementation Function
Part 117	14 consecutive hours	Domestic	`fn_CheckFADutyLimits` checks FlightHours 14 for IsInternational=0
Part 117	20 consecutive hours	International	`fn_CheckFADutyLimits` checks FlightHours 20 for IsInternational=1
Rest Requirement	9 consecutive hours	Between flights	`fn_CalculateRestTime` ensures 9 hours

6.2 Compliance Validation Points

1. **Pre-Assignment Validation**
   - `sp_ScheduleCrew` validates all limits before creating assignments
   - Prevents non-compliant scheduling at source
2. **Available Crew View**
   - `vw_AvailableCrew` only shows crew within limits
   - Simplifies scheduling decisions
3. **Compliance Reporting**
   - Report 2 identifies crew at risk or exceeding limits
   - Proactive monitoring prevents violations
4. **Real-Time Calculation**
   - Dynamic hour calculation ensures current status
   - No stale data from static fields

6.3 Compliance Examples

Example 1: Pilot Approaching 7-Day Limit

```
-- Scenario: Crew 1 has flown 58 hours in last 7 days
SELECT * FROM dbo.fn_CheckHourLimits(1);
-- Returns: Hours168 = 58.0, ExceedsLimits = 0, LimitStatus = 'Within limits'

-- After 2 more hours of flying:
-- Hours168 = 60.0, ExceedsLimits = 0, LimitStatus = 'Within limits'

-- After 1 additional hour:
-- Hours168 = 61.0, ExceedsLimits = 1, LimitStatus = '60h/168h exceeded'
```

Example 2: FA with Insufficient Rest

```
-- Scenario: Crew 21 last flight landed 6 hours ago
SELECT dbo.fn_CalculateRestTime(21, 86) AS RestHours;
-- Returns: 6

SELECT * FROM dbo.fn_CheckFADutyLimits(21, 86);
-- Returns: InsufficientRest = 1

-- sp_ScheduleCrew will skip this crew member for flight 86
```

---

7. Security Implementation

7.1 Data Encryption

Symmetric Key Encryption for SSN Setup:

```
-- Create master key (password should be changed in production)
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'StrongPassword!123';

-- Create certificate
CREATE CERTIFICATE CrewSSNCert
WITH SUBJECT = 'Certificate for SSN Encryption';

-- Create symmetric key with AES-256
CREATE SYMMETRIC KEY CrewSSNKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE CrewSSNCert;
```

Encryption:

```
-- Open key before encrypting
OPEN SYMMETRIC KEY CrewSSNKey DECRYPTION BY CERTIFICATE CrewSSNCert;

-- Encrypt SSN
INSERT INTO Crew (FirstName, LastName, SSN, ...)
VALUES ('John', 'Doe', ENCRYPTBYKEY(KEY_GUID('CrewSSNKey'), '123-45-6789'), ...);

-- Close key
CLOSE SYMMETRIC KEY CrewSSNKey;
```

Decryption:

```
-- Open key before decrypting
OPEN SYMMETRIC KEY CrewSSNKey DECRYPTION BY CERTIFICATE CrewSSNCert;

-- Decrypt SSN
SELECT
    CrewID,
    FirstName,
    LastName,
    CONVERT(NVARCHAR(11), DECRYPTBYKEY(SSN)) AS SSN_Decrypted
FROM Crew;
```

```
-- Close key
CLOSE SYMMETRIC KEY CrewSSNKey;
```

7.2 Role-Based Access Control (RBAC)

Defined Roles

```
CREATE ROLE StationManager;  -- Schedule crew, view flights
CREATE ROLE FlightOps;       -- View flights and crew
CREATE ROLE Compliance;      -- Read-only access to all data
CREATE ROLE HR;              -- Update crew info, view hours
```

Permission Matrix

Object	StationManager	FlightOps	Compliance	HR
Crew	SELECT, INSERT, UPDATE	SELECT	SELECT	SELECT, UPDATE
Flights	SELECT	SELECT	SELECT	-
CrewAssignments	SELECT, INSERT	SELECT	SELECT	SELECT
Airlines	SELECT	SELECT	-	-
Airports	SELECT	SELECT	-	-
vw_AvailableCrew	SELECT	-	SELECT	-
vw_FlightCrew	SELECT	SELECT	SELECT	SELECT
sp_ScheduleCrew	EXECUTE	-	-	-
sp_UpdateFlightStatus	-	EXECUTE	-	-

Grant Examples

```
-- Station managers can schedule crew
GRANT SELECT ON vw_AvailableCrew TO StationManager;
GRANT INSERT ON CrewAssignments TO StationManager;
GRANT EXECUTE ON sp_ScheduleCrew TO StationManager;

-- Compliance has read-only on everything
GRANT SELECT ON Crew TO Compliance;
GRANT SELECT ON Flights TO Compliance;
GRANT SELECT ON CrewAssignments TO Compliance;

-- HR can update crew info
GRANT UPDATE ON Crew TO HR;
```

7.3 Best Practices

1. **Master Key Password**: Store in secure vault (Azure Key Vault, HashiCorp Vault)
2. **Key Rotation**: Rotate encryption keys annually
3. **Access Audit**: Enable SQL Server audit to track data access
4. **TDE**: Enable Transparent Data Encryption for database-level encryption
5. **TLS**: Use TLS 1.2+ for data in transit
6. **Least Privilege**: Grant minimum required permissions
7. **SSN Access**: Limit decryption to HR and compliance only

---

8. Installation Guide

8.1 Prerequisites

- SQL Server 2019 or later (Express, Standard, or Enterprise)
- SQL Server Management Studio (SSMS) 18.0 or later
- Administrative (sysadmin) privileges on SQL Server instance
- Minimum 100MB disk space for database
- Windows or Linux operating system (SQL Server 2019 supports both)

8.2 Installation Methods

Method 1: Step-by-Step Execution (Recommended for Learning) Execute scripts in order:

```
-- Step 1: Create database and schema
:r C:\path\to\01_create_crew_database.sql

-- Step 2: Insert sample data
:r C:\path\to\02_insert_crew_data.sql

-- Step 3: Create business logic
:r C:\path\to\03_crew_logic.sql

-- Step 4: (Optional) Run tests
:r C:\path\to\04_test_crew_logic.sql

-- Step 5: (Optional) View reports
:r C:\path\to\05_reports.sql
```

Method 2: All-in-One Script (Recommended for Production)

```
:r C:\path\to\complete_crew_system.sql
```

Method 3: Command Line (Automated Deployment)

```
# Windows
sqlcmd -S localhost -E -i complete_crew_system.sql -o install_log.txt

# Linux (with SQL Server authentication)
sqlcmd -S localhost -U sa -P 'YourPassword' -i complete_crew_system.sql -o install_log.txt
```

8.3 Post-Installation Steps

1. Verify Database Creation

```
USE master;
SELECT name, collation_name, recovery_model_desc
FROM sys.databases
WHERE name = 'CrewSchedulingDB';
```

2. Verify Table Creation

```
USE CrewSchedulingDB;
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
ORDER BY TABLE_NAME;
-- Expected: 9 tables
```

3. Verify Data Population

```
SELECT 'Airports' AS TableName, COUNT(*) AS RowCount FROM Airports
UNION ALL SELECT 'Airlines', COUNT(*) FROM Airlines
UNION ALL SELECT 'Crew', COUNT(*) FROM Crew
UNION ALL SELECT 'Flights', COUNT(*) FROM Flights
UNION ALL SELECT 'CrewAssignments', COUNT(*) FROM CrewAssignments;

-- Expected:
-- Airports: 10
-- Airlines: 10
-- Crew: 50
-- Flights: 100
-- CrewAssignments: 425
```

4. Verify Functions

```
SELECT name, type_desc
FROM sys.objects
WHERE type IN ('FN', 'IF', 'TF')
  AND name LIKE 'fn_%'
ORDER BY name;
-- Expected: 4 functions
```

5. Verify Stored Procedures

```
SELECT name, type_desc
FROM sys.objects
WHERE type = 'P'
  AND name LIKE 'sp_%'
ORDER BY name;
-- Expected: 2 procedures
```

6. Verify Views

```
SELECT name
FROM sys.views
WHERE name LIKE 'vw_%'
ORDER BY name;
-- Expected: 2 views
```

7. Test Basic Functionality

```
-- Test hour calculation
SELECT dbo.fn_CalculateCrewHours(1, 168) AS Hours7Days;

-- Test available crew view
SELECT COUNT(*) AS AvailableCrewCount FROM vw_AvailableCrew;

-- Test scheduling (should succeed or fail gracefully)
EXEC sp_ScheduleCrew @FlightID = 86;
```

8.4 Configuration

Change Master Key Password (Required for Production)

```
USE CrewSchedulingDB;

-- Regenerate with strong password
ALTER MASTER KEY REGENERATE WITH ENCRYPTION BY PASSWORD = 'Your$ecureProd#Pass123!';
```

Create User Accounts

```
-- Create SQL Server login
USE master;
CREATE LOGIN stationmgr1 WITH PASSWORD = 'SecurePass123!';

-- Create database user
USE CrewSchedulingDB;
CREATE USER stationmgr1 FOR LOGIN stationmgr1;

-- Assign role
ALTER ROLE StationManager ADD MEMBER stationmgr1;
```

Configure Backup

```
-- Full backup
BACKUP DATABASE CrewSchedulingDB
TO DISK = 'C:\Backups\CrewSchedulingDB_Full.bak'
```

```
WITH INIT, COMPRESSION;

-- Configure automated backups via SQL Server Agent or maintenance plans
```

---

9. Usage Guide

9.1 Common Operations

Operation 1: Check Available Crew for a City

```
-- View available crew in New York
SELECT
    CrewID,
    FirstName + ' ' + LastName AS Name,
    CrewTypeName,
    SeniorityName,
    Hours168 AS [Hours Last 7 Days],
    Hours672 AS [Hours Last 28 Days]
FROM vw_AvailableCrew
WHERE BaseCity = 'New York'
ORDER BY CrewTypeName DESC, SeniorityID DESC;
```

Operation 2: Schedule Crew for a Flight

```
-- Check flight details
SELECT
    F.FlightID,
    F.FlightNumber,
    A.City AS DepartureCity,
    F.ScheduledDeparture,
    FS.StatusName
FROM Flights F
JOIN Airports A ON F.DepartureAirportID = A.AirportID
JOIN FlightStatuses FS ON F.StatusID = FS.StatusID
WHERE F.FlightID = 89;

-- Schedule crew
EXEC sp_ScheduleCrew @FlightID = 89;

-- Verify assignment
SELECT
    FC.FlightNumber,
    FC.FirstName + ' ' + FC.LastName AS CrewMember,
    FC.CrewTypeName,
    FC.SeniorityName,
    FC.RoleName
FROM vw_FlightCrew FC
WHERE FC.FlightID = 89
ORDER BY FC.RoleID DESC, FC.SeniorityID DESC;
```

Operation 3: Update Flight Status

```
-- Mark flight as departed
EXEC sp_UpdateFlightStatus @FlightID = 89, @NewStatus = 'InFlight';

-- Check updated timestamps
SELECT
    FlightID,
    FlightNumber,
    ScheduledDeparture,
    ActualDeparture,
    DATEDIFF(MINUTE, ScheduledDeparture, ActualDeparture) AS DelayMinutes
```

```sql
FROM Flights
WHERE FlightID = 89;

-- When flight lands
EXEC sp_UpdateFlightStatus @FlightID = 89, @NewStatus = 'Landed';
```

Operation 4: Check Crew Hour Limits

```sql
-- Check specific crew member
SELECT
    C.FirstName + ' ' + C.LastName AS Name,
    HL.CrewTypeName,
    HL.Hours168 AS [7 Days],
    HL.Hours672 AS [28 Days],
    HL.Hours365Days AS [365 Days],
    HL.LimitStatus
FROM Crew C
CROSS APPLY dbo.fn_CheckHourLimits(C.CrewID) HL
WHERE C.CrewID = 1;

-- Find pilots approaching limits (>54h in 7 days or >90h in 28 days)
SELECT
    C.CrewID,
    C.FirstName + ' ' + C.LastName AS Name,
    HL.Hours168,
    HL.Hours672,
    CASE
        WHEN HL.Hours168 > 54 THEN 'WARN: Near 60h/7d limit'
        WHEN HL.Hours672 > 90 THEN 'WARN: Near 100h/28d limit'
    END AS Warning
FROM Crew C
CROSS APPLY dbo.fn_CheckHourLimits(C.CrewID) HL
WHERE C.CrewTypeID = 2  -- Pilots only
  AND (HL.Hours168 > 54 OR HL.Hours672 > 90)
ORDER BY HL.Hours168 DESC;
```

9.2 Workflow Examples

Workflow 1: Daily Flight Operations

```sql
-- 1. Morning: Schedule crews for all flights
DECLARE @FlightID INT;
DECLARE flight_cursor CURSOR FOR
    SELECT FlightID FROM Flights
    WHERE StatusID = 1  -- Scheduled
      AND CAST(ScheduledDeparture AS DATE) = CAST(GETDATE() AS DATE);

OPEN flight_cursor;
FETCH NEXT FROM flight_cursor INTO @FlightID;

WHILE @@FETCH_STATUS = 0
BEGIN
    BEGIN TRY
        EXEC sp_ScheduleCrew @FlightID = @FlightID;
    END TRY
    BEGIN CATCH
        PRINT 'Failed to schedule flight ' + CAST(@FlightID AS NVARCHAR(10)) + ': ' + ERROR_MESSAGE();
    END CATCH

    FETCH NEXT FROM flight_cursor INTO @FlightID;
END
```

```
CLOSE flight_cursor;
DEALLOCATE flight_cursor;

-- 2. As flights depart: Update status
-- (Typically done by integration with flight operations system)
UPDATE Flights
SET StatusID = 2,  -- InFlight
    ActualDeparture = GETDATE()
WHERE FlightID IN (SELECT FlightID FROM ... );

-- 3. As flights land: Update status
UPDATE Flights
SET StatusID = 3,  -- Landed
    ActualArrival = GETDATE()
WHERE FlightID IN (SELECT FlightID FROM ... );
```

Workflow 2: Weekly Compliance Review

```
-- Generate compliance report
SELECT
    C.CrewID,
    C.FirstName + ' ' + C.LastName AS Name,
    HL.CrewTypeName,
    HL.Hours168 AS [7 Days],
    HL.Hours672 AS [28 Days],
    HL.Hours365Days AS [365 Days],
    HL.ExceedsLimits,
    HL.LimitStatus,
    CASE
        WHEN HL.Hours168 > 54 THEN 'Monitor: Close to 60h/7d'
        WHEN HL.Hours672 > 90 THEN 'Monitor: Close to 100h/28d'
        WHEN HL.Hours365Days > 900 THEN 'Monitor: Close to 1000h/yr'
        ELSE 'OK'
    END AS Action
FROM Crew C
CROSS APPLY dbo.fn_CheckHourLimits(C.CrewID) HL
WHERE C.IsActive = 1
  AND C.CrewTypeID = 2  -- Pilots
  AND (HL.ExceedsLimits = 1
       OR HL.Hours168 > 54
       OR HL.Hours672 > 90
       OR HL.Hours365Days > 900)
ORDER BY HL.ExceedsLimits DESC, HL.Hours672 DESC;
```

---

10. Reports

10.1 Report 1: Crew Currently in Flight

Purpose: Show all crew members on planes currently in the air.

SQL Query:

```
SELECT
    C.CrewID,
    C.FirstName + ' ' + C.LastName AS CrewName,
    CT.CrewTypeName,
    SL.SeniorityName,
    F.FlightID,
    F.FlightNumber,
    DepAir.City AS DepartureCity,
    DestAir.City AS DestinationCity,
    F.ActualDeparture,
    DATEADD(MINUTE, F.FlightDuration, F.ActualDeparture) AS EstimatedArrival,
    F.FlightDuration / 60.0 AS FlightHours,
    R.RoleName
FROM Crew C
JOIN CrewAssignments CA ON C.CrewID = CA.CrewID
JOIN Flights F ON CA.FlightID = F.FlightID
JOIN Airports DepAir ON F.DepartureAirportID = DepAir.AirportID
JOIN Airports DestAir ON F.DestinationAirportID = DestAir.AirportID
JOIN Roles R ON CA.RoleID = R.RoleID
JOIN CrewTypes CT ON C.CrewTypeID = CT.CrewTypeID
JOIN SeniorityLevels SL ON C.SeniorityID = SL.SeniorityID
WHERE F.StatusID = 2  -- InFlight
ORDER BY F.FlightID, R.RoleID DESC, C.LastName;
```

Sample Output:

CrewID	CrewName	CrewType	Seniority	FlightNumber	Departure	Destination	EstArrival	Role
1	John Doe	Pilot	Senior	AS696	New York	Los Angeles	2025-11-09 12:05	Pilot
16	Laura Ramirez	Pilot	Senior	AS696	New York	Los Angeles	2025-11-09 12:05	Pilot
36	Kayla Santiago	Flight Attendant	Senior	AS696	New York	Los Angeles	2025-11-09 12:05	Cabin

Use Cases:

- Operations dashboard
- Real-time tracking
- Emergency response coordination

10.2 Report 2: Crew Exceeding or Approaching Hour Limits

Purpose: Identify crew members who have exceeded or are in danger of exceeding work hour limitations per 14 CFR Part 117 and 121.467.

SQL Query:

```
SELECT
    C.CrewID,
    C.FirstName + ' ' + C.LastName AS CrewName,
    HL.CrewTypeName,
    HL.Hours168 AS [Hours Last 7 Days],
    HL.Hours672 AS [Hours Last 28 Days],
    HL.Hours365Days AS [Hours Last 365 Days],
    HL.ExceedsLimits AS [Exceeds Limits?],
    HL.LimitStatus AS [Limit Status],
    CASE
        WHEN HL.CrewTypeName = 'Pilot' THEN
            CASE
                WHEN HL.Hours168 > 54 THEN 'WARNING: Approaching 60h/7d limit'
                WHEN HL.Hours672 > 90 THEN 'WARNING: Approaching 100h/28d limit'
                WHEN HL.Hours672 > 180 THEN 'WARNING: Approaching 190h/28d limit'
                WHEN HL.Hours365Days > 900 THEN 'WARNING: Approaching 1000h/yr limit'
                ELSE 'OK'
            END
        ELSE 'See duty time per flight'
    END AS [Warning Status]
FROM Crew C
CROSS APPLY dbo.fn_CheckHourLimits(C.CrewID) HL
WHERE C.IsActive = 1
    AND (HL.ExceedsLimits = 1  -- Already exceeding
        OR (HL.CrewTypeName = 'Pilot' AND (HL.Hours168 > 54 OR HL.Hours672 > 90 OR HL.Hours365Days > 900)))
ORDER BY HL.ExceedsLimits DESC, HL.Hours672 DESC;
```

Alert Thresholds:

- **Critical (Red)**: ExceedsLimits = 1 (already over limit)
- **Warning (Yellow)**: Within 10% of limit (54h/7d, 90h/28d, 900h/yr)
- **Normal (Green)**: Below warning threshold

Use Cases:

- Compliance monitoring
- Crew scheduling decisions
- Regulatory audit preparation

10.3 Report 3: Monthly Hours Worked (Payroll)

Purpose: List the number of hours worked per month on a per-employee basis to support payroll processing.

SQL Query:

```
SELECT
    C.CrewID,
    C.FirstName + ' ' + C.LastName AS CrewName,
    CT.CrewTypeName,
    SL.SeniorityName,
    YEAR(F.ScheduledDeparture) AS Year,
    MONTH(F.ScheduledDeparture) AS Month,
    DATENAME(MONTH, F.ScheduledDeparture) AS MonthName,
    COUNT(CA.AssignmentID) AS FlightsWorked,
    SUM(F.FlightDuration / 60.0) AS TotalHoursWorked,
    AVG(F.FlightDuration / 60.0) AS AvgFlightHours
FROM Crew C
JOIN CrewAssignments CA ON C.CrewID = CA.CrewID
JOIN Flights F ON CA.FlightID = F.FlightID
JOIN CrewTypes CT ON C.CrewTypeID = CT.CrewTypeID
JOIN SeniorityLevels SL ON C.SeniorityID = SL.SeniorityID
WHERE F.StatusID IN (2, 3)  -- InFlight or Landed (completed work)
    AND F.ActualDeparture IS NOT NULL
GROUP BY C.CrewID, C.FirstName, C.LastName, CT.CrewTypeName, SL.SeniorityName,
         YEAR(F.ScheduledDeparture), MONTH(F.ScheduledDeparture), DATENAME(MONTH, F.ScheduledDeparture)
ORDER BY Year DESC, Month DESC, TotalHoursWorked DESC;
```

Sample Output:

CrewID	CrewName	Year	Month	FlightsWorked	TotalHours	AvgHours
1	John Doe	2025	11	7	25.5	3.64
16	Laura Ramirez	2025	11	7	25.5	3.64

Use Cases:

- Payroll processing
- Overtime calculation
- Workload analysis

10.4 Report 4: Available Crew for Scheduling

Purpose: Show crew available at a specific airport who are within regulatory limits and ready to be scheduled.

SQL Query:

```
-- Example for NYC departures
SELECT
    AC.CrewID,
    AC.FirstName + ' ' + AC.LastName AS CrewName,
    AC.CrewTypeName,
    AC.SeniorityName,
    AC.BaseCity,
    AC.Hours168 AS [Hours Last 7 Days],
    AC.Hours672 AS [Hours Last 28 Days],
    AC.Hours365Days AS [Hours Last Year],
    AC.LimitStatus AS [Regulatory Status]
FROM vw_AvailableCrew AC
WHERE AC.BaseCity = 'New York'  -- Change based on departure airport
ORDER BY AC.CrewTypeID DESC, AC.SeniorityID DESC, AC.CrewID;
```

Use Cases:

- Pre-scheduling crew availability check
- Staffing level monitoring
- Crew transfer decisions

---

11. Testing Strategy

11.1 Test Coverage

The system includes comprehensive tests in `04_test_crew_logic.sql`:

Unit Tests

- `fn_CalculateCrewHours`: Various time periods (168h, 672h, 8760h)
- `fn_CheckHourLimits`: Pilots exceeding each limit type
- `fn_CheckFADutyLimits`: Domestic vs international flights
- `fn_CalculateRestTime`: Rest period validation
- `vw_AvailableCrew`: Filtering logic
- `vw_FlightCrew`: Data accuracy

Integration Tests

- End-to-end scheduling workflow
- Status update propagation
- Hour calculation after assignments
- Concurrent scheduling attempts

Performance Tests

- `sp_ScheduleCrew` execution time (target: <2 seconds)
- View query performance with JOINs
- Hour calculation with large datasets

11.2 Test Data Scenarios

The sample data includes edge cases:

1. **Crew 1 & 16**: 7 recent flights to approach 60h/168h limit
2. **Crew 2 & 7**: 8 recent flights to approach 100h/672h limit
3. **International flights 76-78**: Test FA 14h/20h duty limits
4. **Flights 96-100**: In-flight for Report 1
5. **Flights 86-95**: Scheduled for sp_ScheduleCrew testing
6. **Various rest times**: 6h, 9h, 12h gaps between flights

11.3 Running Tests

```
-- Execute all tests
:r 04_test_crew_logic.sql

-- Expected output:
-- - All unit tests pass
-- - Integration tests complete successfully
-- - Performance metrics within targets
```

11.4 Validation Checklist

- ☐ Database created successfully
- ☐ All 9 tables populated
- ☐ 4 functions created
- ☐ 2 stored procedures created
- ☐ 2 views created
- ☐ Sample data loaded (50 crew, 100 flights, 425 assignments)
- ☐ sp_ScheduleCrew can assign crew to scheduled flights
- ☐ Hour limits correctly enforced
- ☐ FA duty time limits correctly enforced
- ☐ Rest time validation working
- ☐ All 4 reports execute successfully

---

12. High Availability Architecture

12.1 Design Goals

Target Uptime: 99.9999% (5.26 minutes downtime per year)

12.2 Multi-Tier Architecture

Figure 2: Multi-tier architecture showing global load balancing, regional distribution, application servers, and SQL Server Always On Availability Group with disaster recovery.

12.3 Component Details

Load Balancer Layer

- **Type**: Hardware (F5, Citrix) or Cloud (AWS ALB, Azure LB)
- **Health Checks**: HTTP endpoint `/health` on app servers
- **Algorithms**: Round-robin or least connections
- **SSL Termination**: TLS 1.2+ with strong ciphers
- **Failover Time**: < 5 seconds

Application Server Layer

- **Redundancy**: N+1 (minimum 2 active, 1 standby)
- **State**: Stateless design (no session affinity)
- **Connection Pooling**: Min 10, Max 100 connections per server
- **Retry Logic**: Exponential backoff (1s, 2s, 4s, 8s)
- **Circuit Breaker**: Open after 5 consecutive failures

Database Layer (SQL Server Always On)

- **Primary Replica**:
  - Handles all write operations
  - Synchronous commit to secondary
  - Automatic failover enabled
- **Synchronous Secondary**:
  - Zero data loss (RPO = 0)
  - Automatic failover in < 10 seconds (RTO < 10s)
  - Read-only routing for reporting queries
- **Asynchronous Secondary (DR)**:
  - Different datacenter/region
  - Manual failover
  - Minimal data loss (RPO < 1 minute)

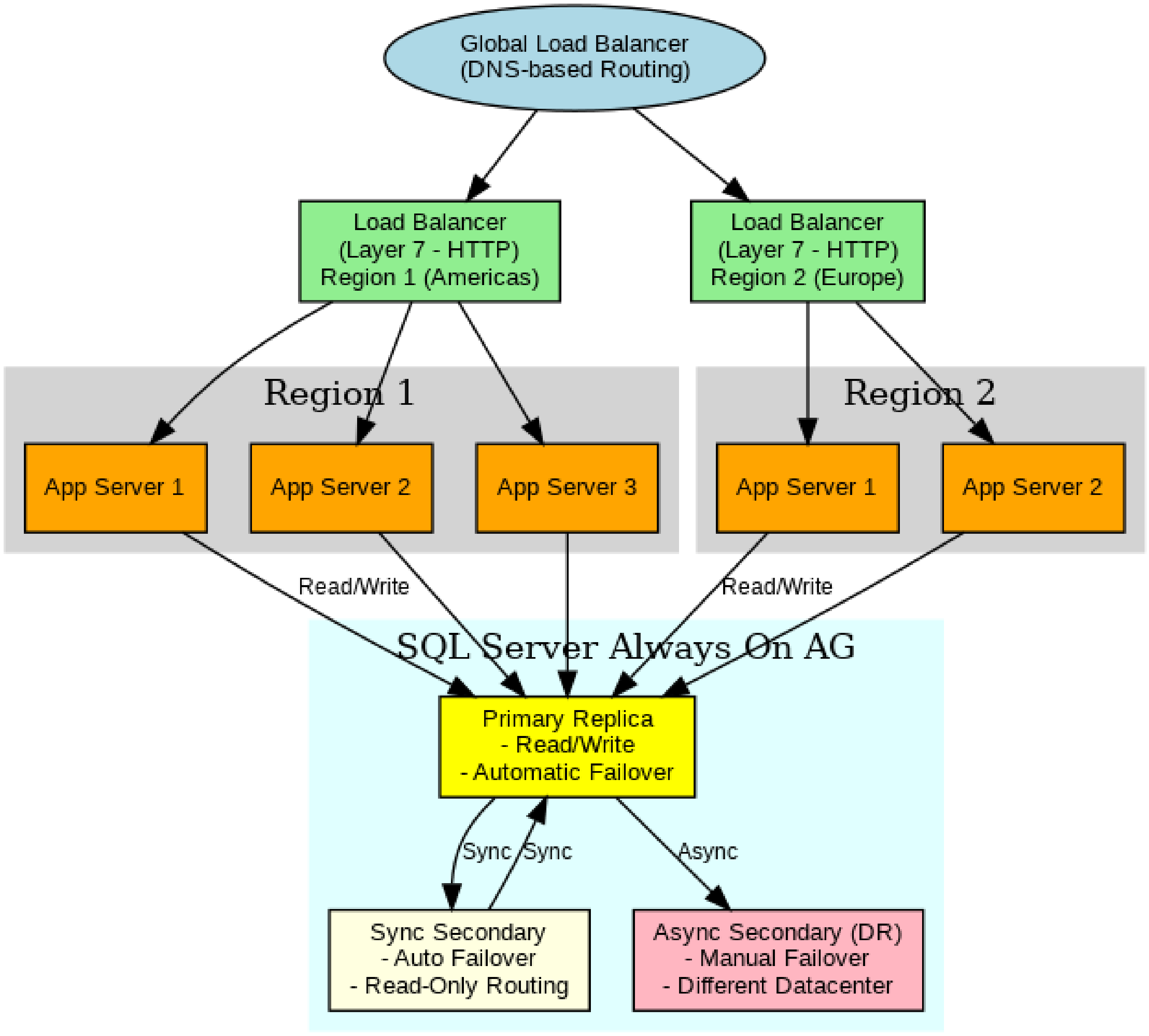

Figure 2: Multi-Tier Architecture Diagram

12.4 Disaster Recovery

Recovery Point Objective (RPO)

- **Synchronous Secondary**: 0 seconds (no data loss)
- **Asynchronous Secondary**: < 60 seconds
- **Backup Recovery**: 15 minutes (last transaction log backup)

Recovery Time Objective (RTO)

- **Automatic Failover**: < 10 seconds
- **Manual Failover**: < 2 minutes
- **Backup Restore**: < 30 minutes

Backup Strategy

```sql
-- Full backup (daily at 2 AM)
BACKUP DATABASE CrewSchedulingDB
TO DISK = 'E:\Backups\CrewSchedulingDB_Full.bak'
WITH COMPRESSION, STATS = 10;

-- Differential backup (every 4 hours)
BACKUP DATABASE CrewSchedulingDB
TO DISK = 'E:\Backups\CrewSchedulingDB_Diff.bak'
WITH DIFFERENTIAL, COMPRESSION;

-- Transaction log backup (every 15 minutes)
BACKUP LOG CrewSchedulingDB
TO DISK = 'E:\Backups\CrewSchedulingDB_Log.trn'
WITH COMPRESSION;
```

Backup Retention:

- Full: 30 days
- Differential: 7 days
- Transaction Log: 3 days
- Offsite Copy: 1 year (monthly fulls)

12.5 Monitoring and Alerting

Health Metrics

- Database online status
- Primary replica availability
- Replication lag (should be < 5 seconds)
- Connection pool utilization
- Query response times
- Disk space (alert at 80% full)

Alert Thresholds

- **Critical**: Database offline, failover occurred, replication lag > 60s
- **Warning**: Replication lag > 10s, disk space > 80%, slow queries > 5s
- **Info**: Backup completed, scheduled maintenance

Monitoring Tools

- SQL Server Management Studio (SSMS)
- SQL Server Agent alerts
- Azure Monitor (if using Azure SQL)
- Third-party: SolarWinds, Redgate, Idera

12.6 Scalability

Vertical Scaling

- Enterprise-class servers: 96+ cores, 512GB+ RAM
- NVMe SSD storage for database files
- 10Gbps+ network interfaces

Horizontal Scaling

- Read-only routing to secondary replicas for reports
- Sharding by airport region if needed (future enhancement)
- Microservices architecture for application tier

Database Optimization

- Indexed views for complex reports
- Columnstore indexes for historical data (future enhancement)
- Table partitioning by month for Flights and CrewAssignments (future)
- Automatic statistics updates enabled
- Query Store enabled for performance monitoring

---

13. Troubleshooting

13.1 Common Issues

Issue 1: sp_ScheduleCrew Fails with "Insufficient qualified crew" Symptoms:

```
Msg 50000, Level 16, State 1
Insufficient pilots available (need 2).
```

Causes:

- Not enough pilots at departure airport
- Pilots exceeding hour limits
- Insufficient rest time for available pilots

Solutions:

```sql
-- Check available pilots at departure airport
SELECT * FROM vw_AvailableCrew
WHERE BaseCity = (SELECT City FROM Airports A
                  JOIN Flights F ON A.AirportID = F.DepartureAirportID
                  WHERE F.FlightID = @FlightID)
  AND CrewTypeName = 'Pilot';

-- Check pilot hour limits
SELECT C.CrewID, C.FirstName, C.LastName, HL.*
FROM Crew C
CROSS APPLY dbo.fn_CheckHourLimits(C.CrewID) HL
WHERE C.BaseAirportID = @DepartureAirportID
  AND C.CrewTypeID = 2;

-- Possible actions:
-- 1. Wait for crew to get adequate rest
-- 2. Transfer crew from another airport
-- 3. Hire additional crew at this location
```

Issue 2: Crew Shows as Exceeding Limits Incorrectly Symptoms:

- `fn_CheckHourLimits` returns `ExceedsLimits = 1` but crew hasn't flown much recently

Causes:

- Incorrect ActualDeparture or ActualArrival times
- Missing ActualArrival causing incorrect duration calculation
- System clock issues

Solutions:

```
-- Verify flight history for crew member
SELECT
    CA.AssignmentID,
    F.FlightID,
    F.FlightNumber,
    F.ScheduledDeparture,
    F.ActualDeparture,
    F.ActualArrival,
    F.FlightDuration,
    DATEDIFF(MINUTE, F.ActualDeparture,
             ISNULL(F.ActualArrival, DATEADD(MINUTE, F.FlightDuration, F.ActualDeparture)))
        AS CalculatedDuration
FROM CrewAssignments CA
JOIN Flights F ON CA.FlightID = F.FlightID
WHERE CA.CrewID = @CrewID
  AND F.ActualDeparture >= DATEADD(HOUR, -672, GETDATE())
ORDER BY F.ActualDeparture DESC;

-- Fix missing ActualArrival
UPDATE Flights
SET ActualArrival = DATEADD(MINUTE, FlightDuration, ActualDeparture)
WHERE ActualDeparture IS NOT NULL
  AND ActualArrival IS NULL
  AND StatusID = 3;  -- Landed status
```

Issue 3: Encryption Key Not Found Symptoms:

```
Msg 15151, Level 16, State 1
Cannot find the symmetric key 'CrewSSNKey', because it does not exist or you do not have permission.
```

Causes:

- Key not opened before use
- Insufficient permissions
- Key not created

Solutions:

```
-- Open the key
OPEN SYMMETRIC KEY CrewSSNKey DECRYPTION BY CERTIFICATE CrewSSNCert;

-- Verify key exists
SELECT name, key_algorithm, key_length
FROM sys.symmetric_keys
WHERE name = 'CrewSSNKey';

-- If key doesn't exist, recreate
CREATE SYMMETRIC KEY CrewSSNKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE CrewSSNCert;
```

Issue 4: Database in Single User Mode Symptoms:

```
Msg 924, Level 14, State 1
Database 'CrewSchedulingDB' is already open and can only have one user at a time.
```

Causes:

- Previous script execution failed during database creation
- Administrative maintenance in progress

Solutions:

```
-- Set to multi-user mode
USE master;
ALTER DATABASE CrewSchedulingDB SET MULTI_USER;
```

```sql
-- If database needs to be dropped and recreated
ALTER DATABASE CrewSchedulingDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
DROP DATABASE CrewSchedulingDB;
-- Then re-run 01_create_crew_database.sql
```

13.2 Performance Optimization

Slow Query: Hour Calculation **Issue**: `fn_CalculateCrewHours` takes > 1 second for crew with many flights

Solution:

```sql
-- Ensure indexes exist
CREATE INDEX IX_Flights_ActualDeparture_StatusID
ON Flights(ActualDeparture, StatusID)
INCLUDE (FlightDuration);

CREATE INDEX IX_CrewAssignments_CrewID_FlightID
ON CrewAssignments(CrewID, FlightID);
```

Slow Query: sp_ScheduleCrew **Issue**: Scheduling takes > 5 seconds

Solution:

```sql
-- Update statistics
UPDATE STATISTICS Crew;
UPDATE STATISTICS Flights;
UPDATE STATISTICS CrewAssignments;

-- Rebuild fragmented indexes
ALTER INDEX ALL ON CrewAssignments REBUILD;
ALTER INDEX ALL ON Flights REBUILD;
```

13.3 Diagnostic Queries

Check Database Size

```sql
USE CrewSchedulingDB;
EXEC sp_spaceused;
```

Check Table Sizes

```sql
SELECT
    t.NAME AS TableName,
    p.rows AS RowCounts,
    SUM(a.total_pages) * 8 / 1024 AS TotalSpaceMB
FROM sys.tables t
JOIN sys.partitions p ON t.object_id = p.object_id
JOIN sys.allocation_units a ON p.partition_id = a.container_id
WHERE t.is_ms_shipped = 0
GROUP BY t.NAME, p.rows
ORDER BY TotalSpaceMB DESC;
```

Check Active Connections

```sql
SELECT
    session_id,
    login_name,
    host_name,
    program_name,
    status,
    last_request_start_time
FROM sys.dm_exec_sessions
WHERE database_id = DB_ID('CrewSchedulingDB');
```

Check Long-Running Queries

```
SELECT
    r.session_id,
    r.status,
    r.command,
    r.start_time,
    DATEDIFF(SECOND, r.start_time, GETDATE()) AS duration_seconds,
    t.text AS query_text
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.database_id = DB_ID('CrewSchedulingDB')
  AND r.status <> 'sleeping'
ORDER BY duration_seconds DESC;
```

---

14. Appendix

14.1 FAA Regulations Reference

- 14 CFR Part 117 - Flight and Duty Limitations and Rest Requirements
- 14 CFR 121.467 - Flight Time Limitations: Pilots

14.2 SQL Server Documentation

- Always On Availability Groups
- Transparent Data Encryption (TDE)
- Symmetric Key Encryption

14.3 File Structure

```
odloty/
  complete_crew_system.sql            # All-in-one script
  COMPREHENSIVE_DOCUMENTATION.md      # This file
  README.md                           # Project overview
  zadanie.md                          # Original requirements
  .gitignore                          # Git ignore rules
```

14.4 Script Execution Order

Production Deployment:

- `complete_crew_system.sql` - One-step deployment

14.5 Glossary

Term	Definition
Crew Type	Classification of crew member (Pilot or Flight Attendant)
Seniority	Experience level (Trainee, Journeyman, Senior/Captain)
Flight Status	Current state of flight (Scheduled, InFlight, Landed)
Duty Time	Total time from start to end of duty period including flight time and ground time
Flight Time	Time from aircraft movement for takeoff to engine shutdown after landing
Rest Period	Off-duty time between flights, minimum 9 hours for FAs
Base Airport	Home airport where crew member is stationed
Dynamic Calculation	Real-time computation from flight history vs. stored values
RPO	Recovery Point Objective - maximum acceptable data loss
RTO	Recovery Time Objective - maximum acceptable downtime
Always On AG	SQL Server high availability feature with automatic failover

14.6 Change Log

Version	Date	Changes
1.0	Oct 2025	Initial implementation with static hour tracking
2.0	Nov 2025	Replaced static hour tracking with dynamic calculation. Added comprehensive edge case data. Fixed script execution order. Enhanced documentation.

14.7 License

14.8 Contact

For questions, issues, or suggestions:

- Open an issue in the GitHub repository
- Contact the development team

END OF DOCUMENTATION

Last Updated: November 2025
Document Version: 2.0
Database Version: 2.0