Flyway Rollback Framework - Practical Examples

Migration Script Examples with Rollback Support

1. DDL Example: Create Table with Complex Structure

Forward Migration: V1.0__create_user_system.sql

sql

```sql
-- Create user table with audit fields
CREATE TABLE users (
    id BIGINT NOT NULL AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    status ENUM('ACTIVE', 'INACTIVE', 'SUSPENDED') DEFAULT 'ACTIVE',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    created_by VARCHAR(50),
    updated_by VARCHAR(50),
    version INT DEFAULT 1,
    PRIMARY KEY (id),
    UNIQUE KEY uk_username (username),
    UNIQUE KEY uk_email (email),
    INDEX idx_status (status),
    INDEX idx_created_at (created_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Create user roles table
CREATE TABLE user_roles (
    id BIGINT NOT NULL AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    role_name VARCHAR(50) NOT NULL,
    granted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    granted_by VARCHAR(50),
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    UNIQUE KEY uk_user_role (user_id, role_name)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Create audit log table
CREATE TABLE user_audit_log (
    id BIGINT NOT NULL AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    action VARCHAR(50) NOT NULL,
    old_values JSON,
    new_values JSON,
    performed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    performed_by VARCHAR(50),
    ip_address VARCHAR(45),
    user_agent TEXT,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
```

```
    INDEX idx_user_action (user_id, action),
    INDEX idx_performed_at (performed_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Rollback Script: U1.0__rollback_create_user_system.sql

sql

```sql
-- Rollback script for V1.0__create_user_system.sql
-- This script safely removes the user system tables

-- First, capture any data that needs to be preserved
CREATE TABLE IF NOT EXISTS rollback_archive_users AS
SELECT * FROM users WHERE 1=0;

INSERT INTO rollback_archive_users
SELECT * FROM users;

-- Archive related data
CREATE TABLE IF NOT EXISTS rollback_archive_user_roles AS
SELECT * FROM user_roles WHERE 1=0;

INSERT INTO rollback_archive_user_roles
SELECT * FROM user_roles;

CREATE TABLE IF NOT EXISTS rollback_archive_user_audit_log AS
SELECT * FROM user_audit_log WHERE 1=0;

INSERT INTO rollback_archive_user_audit_log
SELECT * FROM user_audit_log;

-- Drop tables in reverse order of creation (respecting foreign keys)
DROP TABLE IF EXISTS user_audit_log;
DROP TABLE IF EXISTS user_roles;
DROP TABLE IF EXISTS users;

-- Log the rollback
INSERT INTO flyway_rollback_audit (
    version,
    description,
    rollback_type,
    archived_tables,
    performed_at,
    performed_by
) VALUES (
    '1.0',
    'Rollback user system creation',
    'DDL',
    JSON_ARRAY('users', 'user_roles', 'user_audit_log'),
    NOW(),
    USER()
);
```

2. DML Example: Data Migration with Rollback

Forward Migration: V1.1__migrate_user_data.sql

sql

```sql
-- Migrate user data to new format
-- Add email verification status

-- Add new column
ALTER TABLE users ADD COLUMN email_verified BOOLEAN DEFAULT FALSE AFTER email;
ALTER TABLE users ADD COLUMN email_verification_token VARCHAR(255) AFTER email_verified;
ALTER TABLE users ADD COLUMN email_verified_at TIMESTAMP NULL AFTER email_verification_token;

-- Archive current state for rollback
CREATE TABLE IF NOT EXISTS users_snapshot_v1_1 AS
SELECT id, email, created_at FROM users;

-- Update existing users based on creation date
UPDATE users
SET email_verified = TRUE,
    email_verified_at = created_at
WHERE created_at < DATE_SUB(NOW(), INTERVAL 30 DAY);

-- Generate verification tokens for unverified users
UPDATE users
SET email_verification_token = UUID()
WHERE email_verified = FALSE;

-- Create verification tracking table
CREATE TABLE email_verifications (
    id BIGINT NOT NULL AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    token VARCHAR(255) NOT NULL,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP NOT NULL,
    verified_at TIMESTAMP NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    UNIQUE KEY uk_token (token),
    INDEX idx_expires_at (expires_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Populate verification table
INSERT INTO email_verifications (user_id, token, sent_at, expires_at)
SELECT
    id,
    email_verification_token,
    NOW(),
    DATE_ADD(NOW(), INTERVAL 7 DAY)
FROM users
```

```
WHERE email_verified = FALSE
  AND email_verification_token IS NOT NULL;
```

Rollback Script: U1.1__rollback_migrate_user_data.sql

sql

```
-- Rollback script for V1.1__migrate_user_data.sql
-- Safely revert email verification changes

-- Drop the verification tracking table
DROP TABLE IF EXISTS email_verifications;

-- Remove the added columns
ALTER TABLE users DROP COLUMN email_verified_at;
ALTER TABLE users DROP COLUMN email_verification_token;
ALTER TABLE users DROP COLUMN email_verified;

-- Clean up snapshot table
DROP TABLE IF EXISTS users_snapshot_v1_1;

-- Log the rollback
INSERT INTO flyway_rollback_audit (
    version,
    description,
    rollback_type,
    affected_rows,
    performed_at,
    performed_by
) VALUES (
    '1.1',
    'Rollback email verification migration',
    'DML',
    (SELECT COUNT(*) FROM users),
    NOW(),
    USER()
);
```

3. Complex DDL Example: Table Restructuring

Forward Migration: V1.2__restructure_user_profile.sql

sql

```sql
-- Restructure user profile data into separate table

-- Create new profile table
CREATE TABLE user_profiles (
    id BIGINT NOT NULL AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    bio TEXT,
    avatar_url VARCHAR(500),
    phone_number VARCHAR(20),
    date_of_birth DATE,
    country_code VARCHAR(2),
    timezone VARCHAR(50),
    preferences JSON,
    social_links JSON,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    UNIQUE KEY uk_user_id (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Archive existing profile data from users table
CREATE TABLE users_profile_archive_v1_2 AS
SELECT
    id,
    first_name,
    last_name,
    COALESCE(
        JSON_EXTRACT(metadata, '$.bio'),
        ''
    ) as bio,
    COALESCE(
        JSON_EXTRACT(metadata, '$.avatar_url'),
        ''
    ) as avatar_url,
    COALESCE(
        JSON_EXTRACT(metadata, '$.phone_number'),
        ''
    ) as phone_number
FROM users
WHERE metadata IS NOT NULL;

-- Migrate existing profile data
INSERT INTO user_profiles (
    user_id,
    bio,
```

```
    avatar_url,
    phone_number,
    preferences,
    created_at,
    updated_at
)
SELECT
    id,
    JSON_UNQUOTE(JSON_EXTRACT(metadata, '$.bio')),
    JSON_UNQUOTE(JSON_EXTRACT(metadata, '$.avatar_url')),
    JSON_UNQUOTE(JSON_EXTRACT(metadata, '$.phone_number')),
    JSON_EXTRACT(metadata, '$.preferences'),
    created_at,
    updated_at
FROM users
WHERE metadata IS NOT NULL;

-- Remove profile fields from users table
ALTER TABLE users DROP COLUMN first_name;
ALTER TABLE users DROP COLUMN last_name;
ALTER TABLE users DROP COLUMN metadata;
```

Rollback Script: U1.2__rollback_restructure_user_profile.sql

sql

```sql
-- Rollback script for V1.2__restructure_user_profile.sql
-- Restore profile data to users table

-- Re-add columns to users table
ALTER TABLE users
    ADD COLUMN first_name VARCHAR(100) AFTER email_verified_at,
    ADD COLUMN last_name VARCHAR(100) AFTER first_name,
    ADD COLUMN metadata JSON AFTER last_name;

-- Restore data from profile table
UPDATE users u
INNER JOIN user_profiles p ON u.id = p.user_id
SET
    u.metadata = JSON_OBJECT(
        'bio', p.bio,
        'avatar_url', p.avatar_url,
        'phone_number', p.phone_number,
        'preferences', p.preferences
    );

-- Restore first_name and last_name from archive
UPDATE users u
INNER JOIN users_profile_archive_v1_2 a ON u.id = a.id
SET
    u.first_name = a.first_name,
    u.last_name = a.last_name;

-- Drop the profile table
DROP TABLE IF EXISTS user_profiles;

-- Clean up archive table
DROP TABLE IF EXISTS users_profile_archive_v1_2;
```

4. DCL Example: Permission Changes

Forward Migration: V1.3__grant_read_permissions.sql

sql

```sql
-- Grant read permissions to reporting user

-- Create reporting user if not exists
CREATE USER IF NOT EXISTS 'reporting_user'@'%' IDENTIFIED BY 'secure_password_here';

-- Archive current permissions
CREATE TABLE IF NOT EXISTS permission_archive_v1_3 (
    user VARCHAR(100),
    host VARCHAR(100),
    privilege_type VARCHAR(100),
    is_grantable VARCHAR(3),
    archived_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Store current permissions
INSERT INTO permission_archive_v1_3 (user, host, privilege_type, is_grantable)
SELECT User, Host, Privilege_Type, Is_Grantable
FROM mysql.user
WHERE User = 'reporting_user';

-- Grant new permissions
GRANT SELECT ON mydb.users TO 'reporting_user'@'%';
GRANT SELECT ON mydb.user_roles TO 'reporting_user'@'%';
GRANT SELECT ON mydb.user_audit_log TO 'reporting_user'@'%';
GRANT SELECT ON mydb.user_profiles TO 'reporting_user'@'%';

-- Create read-only views for sensitive data
CREATE VIEW v_users_public AS
SELECT
    id,
    username,
    email,
    status,
    created_at
FROM users;

GRANT SELECT ON mydb.v_users_public TO 'reporting_user'@'%';

FLUSH PRIVILEGES;
```

Rollback Script: U1.3__rollback_grant_read_permissions.sql

sql

```sql
-- Rollback script for V1.3__grant_read_permissions.sql
-- Revoke permissions and remove reporting user

-- Revoke granted permissions
REVOKE SELECT ON mydb.users FROM 'reporting_user'@'%';
REVOKE SELECT ON mydb.user_roles FROM 'reporting_user'@'%';
REVOKE SELECT ON mydb.user_audit_log FROM 'reporting_user'@'%';
REVOKE SELECT ON mydb.user_profiles FROM 'reporting_user'@'%';
REVOKE SELECT ON mydb.v_users_public FROM 'reporting_user'@'%';

-- Drop created views
DROP VIEW IF EXISTS v_users_public;

-- Remove reporting user
DROP USER IF EXISTS 'reporting_user'@'%';

-- Clean up archive table
DROP TABLE IF EXISTS permission_archive_v1_3;

FLUSH PRIVILEGES;
```

Usage Examples

1. Basic Rollback Command

bash

```bash
# Rollback to specific version
curl -X POST http://localhost:8080/api/flyway/rollback/execute \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{
    "targetVersion": "1.1",
    "dryRun": false,
    "createSnapshot": true,
    "reason": "Feature causing performance issues"
  }'
```

2. Dry Run Before Actual Rollback

bash

```bash
# First, perform a dry run
curl -X POST http://localhost:8080/api/flyway/rollback/dry-run \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{
    "targetVersion": "1.0",
    "includeDataAnalysis": true
  }'
```

3. Emergency Rollback (Skip Approval)

bash

```bash
# Emergency rollback without approval workflow
curl -X POST http://localhost:8080/api/flyway/rollback/execute \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{
    "targetVersion": "1.1",
    "emergency": true,
    "reason": "Production outage - data corruption detected",
    "createSnapshot": true,
    "notifyOnCompletion": true
  }'
```

4. Programmatic Rollback in Spring Boot

java

```java
@Service
public class DatabaseMaintenanceService {

    @Autowired
    private FlywayRollbackManager rollbackManager;

    public void performScheduledRollback() {
        RollbackOptions options = RollbackOptions.builder()
            .dryRun(false)
            .createSnapshot(true)
            .verifyDataIntegrity(true)
            .notifyOnCompletion(true)
            .timeout(Duration.ofMinutes(30))
            .build();

        try {
            RollbackResult result = rollbackManager.rollbackToVersion("1.1", options);
            log.info("Rollback completed: {}", result);
        } catch (RollbackException e) {
            log.error("Rollback failed", e);
            // Handle failure - maybe restore from snapshot
        }
    }
}
```

5. Monitoring Rollback Progress

java

```java
@EventListener
public class RollbackProgressMonitor {

    @EventListener
    public void handleProgress(RollbackProgressEvent event) {
        log.info("Rollback progress: {} - Operation: {} - Status: {}",
            event.getContext().getRollbackId(),
            event.getOperation().getDescription(),
            event.getOperation().getStatus());

        // Send real-time updates to monitoring dashboard
        monitoringService.updateRollbackProgress(
            event.getContext().getRollbackId(),
            event.getProgress()
        );
    }

    @EventListener
    public void handleCompletion(RollbackCompletedEvent event) {
        // Send notifications
        notificationService.notifyRollbackComplete(event.getResult());

        // Update metrics
        metricsService.recordRollbackSuccess(event.getResult());
    }
}
```

Testing Rollback Scripts

1. Unit Test for Rollback Generation

java

```java
@Test
public void testDDLRollbackGeneration() {
    String createTableDDL = "CREATE TABLE test_table (id INT PRIMARY KEY, name VARCHAR(50))";

    DDLRollbackGenerator generator = new DDLRollbackGenerator(dataSource);
    String rollbackScript = generator.generateRollback(createTableDDL, new MigrationVersion("1.0"));

    assertThat(rollbackScript).contains("DROP TABLE IF EXISTS test_table");
}
```

2. Integration Test for Complete Rollback

java

```java
@SpringBootTest
@Transactional
public class RollbackIntegrationTest {

    @Autowired
    private FlywayRollbackManager rollbackManager;

    @Test
    public void testCompleteRollbackScenario() {
        // Apply migrations
        flyway.migrate();

        // Verify current version
        MigrationInfo current = flyway.info().current();
        assertThat(current.getVersion().toString()).isEqualTo("1.3");

        // Perform rollback
        RollbackResult result = rollbackManager.rollbackToVersion("1.1",
            RollbackOptions.builder().dryRun(false).build());

        // Verify rollback success
        assertThat(result.isSuccess()).isTrue();
        assertThat(result.getTargetVersion()).isEqualTo("1.1");

        // Verify database state
        MigrationInfo newCurrent = flyway.info().current();
        assertThat(newCurrent.getVersion().toString()).isEqualTo("1.1");
    }
}
```

Production Checklist

Before using this rollback framework in production:

1. **Test all rollback scripts** in a staging environment
2. **Verify data integrity** after each rollback
3. **Set up monitoring** for rollback operations
4. **Configure alerts** for failed rollbacks
5. **Document rollback procedures** for each migration
6. **Train team members** on rollback procedures
7. **Set up automated backups** before migrations
8. **Test snapshot and restore** functionality

9. **Configure appropriate permissions** for rollback operations
10. **Establish approval workflow** for non-emergency rollbacks