Practical Example - Complete Flow with Code

🎬 Real Scenario: E-commerce Database Evolution

Let's follow a realistic example of an e-commerce application database that evolves and needs rollback.

Timeline & Database Evolution

```
Version 1: Basic Users
├── users table
└── Simple registration

Version 2: Add Orders
├── users table
├── orders table
└── Order management

Version 3: Add Products (Has Bug!)
├── users table
├── orders table
├── products table ← Bug here!
└── Product catalog

Need to rollback to V2!
```

📝 Step-by-Step Code Flow

Day 1: Initial Deployment (V1)

V1__create_users.sql:

```sql
CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    name VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (email, name) VALUES
('admin@shop.com', 'Admin User');
```

What happens in the framework:

```
App Start
   ↓
FlywayMigrationStrategy.migrate()
   ↓
1. Create snapshot: "pre_migration_20240109_090000"
2. Execute V1__create_users.sql
3. Update flyway_schema_history:
  | version | description  | success |
  |---------|--------------|---------|
  | 1       | create users| true    |
4. App Ready ✅
```

Day 2: Add Orders Feature (V2)

V2__create_orders.sql:

sql

```sql
CREATE TABLE orders (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    total_amount DECIMAL(10,2),
    status VARCHAR(20) DEFAULT 'PENDING',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Rollback script: U2__rollback_orders.sql
CREATE TABLE IF NOT EXISTS archive_orders AS SELECT * FROM orders;
DROP TABLE orders;
```

Framework execution:

java

```java
// In FlywayRollbackManager.java
public void onMigration() {
    // 1. Snapshot before V2
    snapshotManager.createSnapshot("pre_v2_20240110_090000");

    // 2. Run migration
    flyway.migrate(); // Executes V2__create_orders.sql

    // 3. Update history
    // | version | description   | success |
    // |---------|---------------|---------|
    // | 1       | create users | true    |
    // | 2       | create orders| true    |
}
```

Day 3: Add Products Feature (V3 - Has Issues!)

V3__create_products.sql:

sql

```sql
CREATE TABLE products (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10,2),
    stock INT DEFAULT 0
    -- Missing comma here! Syntax error!
    category VARCHAR(50)
);
```

What happens:

```
Migration Starts
    ↓
1. Create snapshot: "pre_v3_20240111_090000" ✅
    ↓
2. Execute V3__create_products.sql
    ↓
3. ERROR: SQL Syntax Error ❌
    ↓
4. Check auto-rollback setting
    ↓
5. Auto-rollback disabled → Stop
```

Day 3: Manual Rollback Needed!

API Call to rollback:

bash

```
curl -X POST http://localhost:8080/api/flyway/rollback/execute \
  -H "Content-Type: application/json" \
  -d '{
    "targetVersion": "2",
    "dryRun": false,
    "reason": "V3 migration failed - syntax error"
  }'
```

Inside the Framework - Complete Flow:

java

```java
@PostMapping("/execute")
public ResponseEntity<RollbackResult> executeRollback(@RequestBody RollbackRequest request) {
    // Step 1: Entry point
    return rollbackManager.rollbackToVersion(request.getTargetVersion());
}

// In FlywayRollbackManager.java
public RollbackResult rollbackToVersion(String targetVersion) {
    String rollbackId = "rollback_" + UUID.randomUUID();

    // Step 2: Validation
    validateTargetVersion(targetVersion); // "2" is valid ✅

    // Step 3: Safety Checks
    SafetyCheckResult safety = performSafetyChecks();
    // - Active connections: 2 (OK) ✅
    // - Disk space: 500MB free (OK) ✅
    // - Replication lag: N/A ✅

    // Step 4: Create Rollback Snapshot
    String snapshotId = snapshotManager.createSnapshot("rollback_to_v2");
    // Saves current state including partial V3 changes

    // Step 5: Generate Rollback Plan
    RollbackPlan plan = generateRollbackPlan("3", "2");
    // Plan: Just need to clean up failed V3 (no successful V3 to undo)

    // Step 6: Execute Rollback
    executePlan(plan);

    // Step 7: Update Schema History
    updateFlywayHistory(targetVersion);

    return new RollbackResult(true, rollbackId, "2", snapshotId, null);
}
```

Detailed Execution Steps:

ROLLBACK EXECUTION DETAILS:
==========

1. Current State Analysis:
   - Schema History shows: V1 ✅, V2 ✅, V3 ❌
   - Tables exist: users, orders
   - Partial V3 table: None (failed before creation)

2. Rollback Plan Generated:
   - No U3 script needed (V3 never succeeded)
   - Clean failed migration entry
   - Verify V2 state

3. SQL Executed:
   ```sql
   -- Remove failed migration record
   DELETE FROM flyway_schema_history WHERE version = '3';

   -- Verify database state
   SELECT COUNT(*) FROM users;    -- Returns: 1
   SELECT COUNT(*) FROM orders;   -- Returns: 0

4. Update Audit Log:

```sql
INSERT INTO flyway_rollback_audit
(rollback_id, version, status, performed_at, reason)
VALUES
('rollback_abc123', '2', 'SUCCESS', NOW(), 'V3 migration failed - syntax error');
```

5. Final State:

- Database at V2 ✅
- All tables intact ✅
- Ready for fixed V3 ✅
```

Day 4: Fix and Redeploy V3

V3__create_products_fixed.sql:
```sql
CREATE TABLE products (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10,2),
    stock INT DEFAULT 0,  -- Fixed: Added comma
    category VARCHAR(50)
);
```

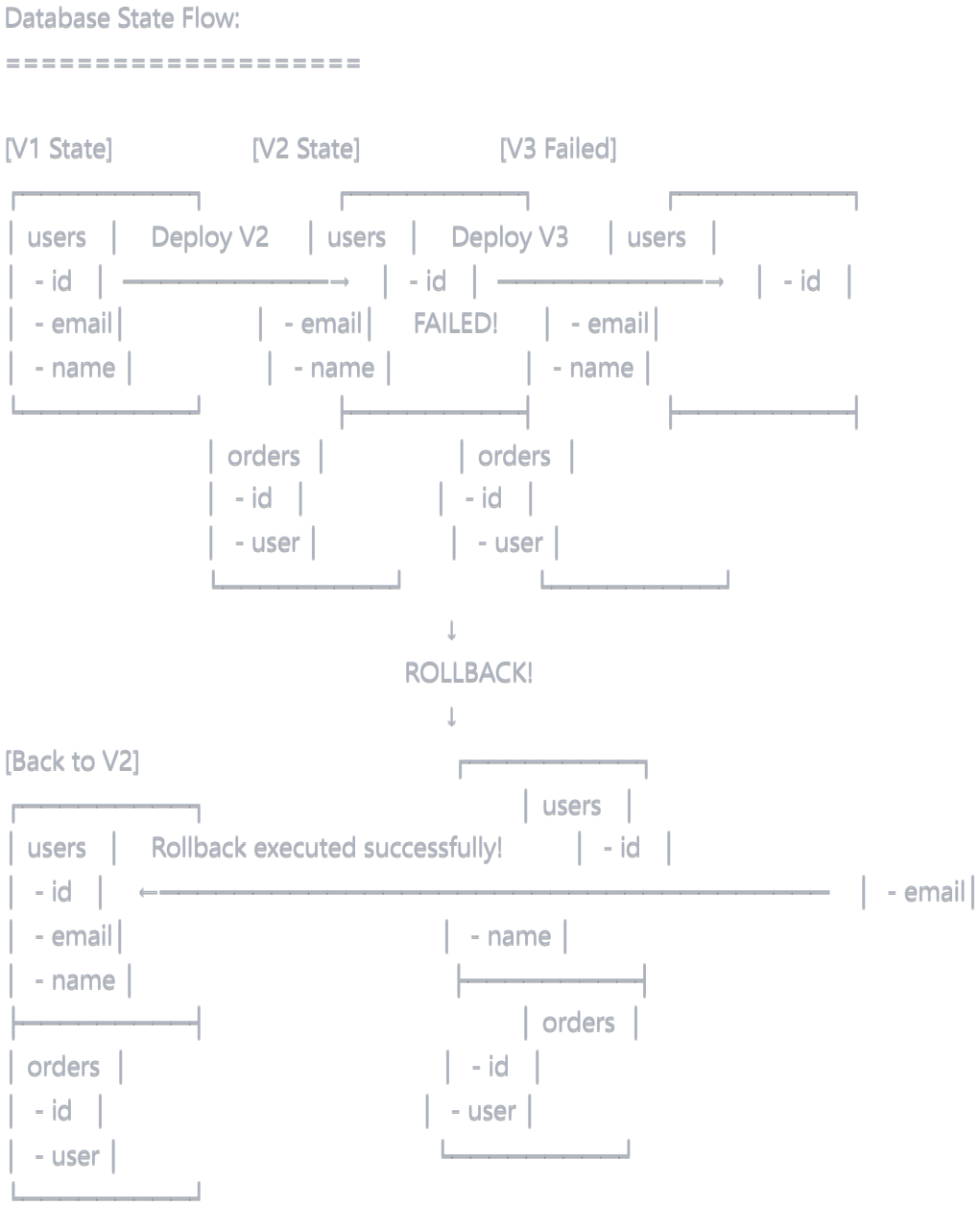
Now it works!

Migration Success Path:
1. Snapshot: "pre_v3_fixed_20240112_090000"
2. Execute V3 (fixed) ✅
3. Update history:

version	description	success
1	create users	true
2	create orders	true
3	create products	true

4. App running with all features! 🎉

📊 Visual State Transitions



🔍 What's Happening Behind the Scenes

During Normal Migration:

java

```java
// FlywayMigrationStrategy intercepts
@Bean
public FlywayMigrationStrategy flywayMigrationStrategy(FlywayRollbackManager manager) {
    return flyway -> {
        // 1. Pre-migration snapshot
        log.info("Creating pre-migration snapshot...");
        String snapshotId = manager.createPreMigrationSnapshot();

        try {
            // 2. Let Flyway do its thing
            log.info("Executing Flyway migration...");
            flyway.migrate();
            log.info("Migration successful!");

        } catch (Exception e) {
            log.error("Migration failed!", e);

            // 3. Auto-rollback check
            if (manager.isAutoRollbackEnabled()) {
                log.warn("Attempting automatic rollback...");
                manager.handleMigrationFailure(e);
            }
            throw e;
        }
    };
}
```

During Rollback Request:

java

```java
// Complete request flow
Client Request
    ↓
RollbackController
    ↓
FlywayRollbackManager
    ↓
[Validate → Safety → Snapshot → Plan → Execute → Verify → Audit]
    ↓
Response to Client
```

💾 File System After Our Example

```
~/flyway-snapshots/
├── pre_migration_20240109_090000/     (Before V1)
├── pre_v2_20240110_090000/            (Before V2)
├── pre_v3_20240111_090000/            (Before failed V3)
├── rollback_to_v2_20240111_091500/    (During rollback)
└── pre_v3_fixed_20240112_090000/      (Before fixed V3)

project/src/main/resources/db/
├── migration/
│   ├── V1__create_users.sql
│   ├── V2__create_orders.sql
│   └── V3__create_products_fixed.sql
└── rollback/
    ├── U1__rollback_users.sql
    ├── U2__rollback_orders.sql
    └── U3__rollback_products.sql
```

🎯 Key Learning Points

1. **Snapshots are taken BEFORE each operation** - Safety first!
2. **Failed migrations don't need rollback scripts** - They never completed
3. **Successful migrations can be undone** - Using U scripts
4. **Everything is logged** - Complete audit trail
5. **The framework is smart** - It knows what to do in each situation

This is how the framework protects your database while allowing you to evolve it safely!