Übung 2: Git-Bereiche und Änderungen rückgängig machen

Lernziele

Durch das Absolvieren dieser Übung werden Sie folgende Bereiche praktizieren:

- Verstehen des Arbeitsbereichs, Staging-Bereichs und Repositorys
- Verwendung des Stash für temporäre Speicherung
- Rückgängigmachen von Änderungen mit `git restore`, `git reset` und `git revert`
- Wiederherstellung verschiedener Fehler und unerwünschter Änderungen

Szenario

Sie arbeiten an einem Data-Warehouse-Projekt und müssen das Verwalten verschiedener Git-Bereiche und die Wiederherstellung von häufigen Fehlern üben. Diese Übung simuliert reale Szenarien, in denen Sie Änderungen rückgängig machen, laufende Arbeiten verwalten und verschiedene Git-Zustände handhaben müssen.

Setup

Initialisieren Sie ein neues Repository:

```
# Projektverzeichnis erstellen
mkdir DataWarehouseProjekt
cd DataWarehouseProjekt

# Repository initialisieren
git init
git config user.name "Ihr Name"
git config user.email "ihre.email@beispiel.de"
```

Teil 1: Verstehen der Git-Bereiche

Schritt 1: Erstelle anfängliche Dateien

Erstellen Sie die folgenden Dateien, um eine Grundlage zu schaffen:

`schema.sql`:

```
-- Data Warehouse Schema
CREATE SCHEMA dwh;

CREATE TABLE dwh.dim_kunde (
    kunden_key INT IDENTITY(1,1) PRIMARY KEY,
    kunden_id INT NOT NULL,
```

```
    vorname NVARCHAR(50),
    nachname NVARCHAR(50),
    email NVARCHAR(100)
);
```

etl_prozess.sql:

```
-- ETL-Prozess-Skript
-- Dimensionentabellen laden

INSERT INTO dwh.dim_kunde (kunden_id, vorname, nachname, email)
SELECT
    kunden_id,
    vorname,
    nachname,
    email
FROM staging.kunden
WHERE erstellt_datum >= DATEADD(day, -1, GETDATE());
```

1. Stagen und committen Sie diese Dateien mit der Nachricht: "Anfängliches Schema und ETL-Skripte"

Schritt 2: Arbeitsbereich vs. Staging-Bereich Übung

1. Ändern Sie `schema.sql`, indem Sie diese Tabelle am Ende hinzufügen:

```
CREATE TABLE dwh.dim_produkt (
    produkt_key INT IDENTITY(1,1) PRIMARY KEY,
    produkt_id INT NOT NULL,
    produktname NVARCHAR(100),
    kategorie NVARCHAR(50),
    preis DECIMAL(10,2)
);
```

2. Erstellen Sie eine neue Datei `config.sql`:

```
-- Konfigurationseinstellungen
USE DataWarehouse;
SET ANSI_NULLS ON;
SET QUOTED_IDENTIFIER ON;
```

3. Verwenden Sie `git status`, um beide Dateien im Arbeitsbereich zu sehen
4. Stagen Sie nur `schema.sql`: `git add schema.sql`
5. Verwenden Sie `git status` erneut - bemerken Sie eine Datei gestaged, eine unverfolgt

6. Ändern Sie `schema.sql` erneut, indem Sie einen Kommentar hinzufügen: `-- Aktualisiert: [heutiges Datum]`
7. Verwenden Sie `git status` - bemerken Sie, dass `schema.sql` sowohl in gestaged als auch in unstaged Bereichen erscheint

Teil 2: Git Restore verwenden

Schritt 1: Dateien aus dem Staging entfernen

1. Entfernen Sie `schema.sql` aus dem Staging: `git restore --staged schema.sql`
2. Überprüfen Sie mit `git status`, dass es jetzt nur im Arbeitsbereich ist
3. Stagen Sie alle Dateien: `git add .`
4. Entfernen Sie alle Dateien aus dem Staging: `git restore --staged .`

Schritt 2: Arbeitsverzeichnis-Änderungen verwerfen

1. Stellen Sie sicher, dass `schema.sql` Ihre Änderungen von früher hat
2. Verwerfen Sie Änderungen im Arbeitsverzeichnis: `git restore schema.sql`
3. Überprüfen Sie, dass die Datei zu ihrem ursprünglichen Zustand zurückgekehrt ist
4. Fügen Sie Ihre Änderungen erneut hinzu und committen Sie sie: "Produktdimensions-Tabelle hinzugefügt"

Schritt 3: Übung mit mehreren Zuständen

1. Erstellen Sie `fact_sales.sql`:

```
-- Verkaufs-Fakten-Tabelle
CREATE TABLE dwh.fact_sales (
    verkauf_key INT IDENTITY(1,1) PRIMARY KEY,
    kunden_key INT FOREIGN KEY REFERENCES dwh.dim_kunde(kunden_key),
    produkt_key INT FOREIGN KEY REFERENCES dwh.dim_produkt(produkt_key),
    verkaufsdatum DATE,
    menge INT,
    einzelpreis DECIMAL(10,2),
    gesamtbetrag DECIMAL(10,2)
);
```

2. Stagen Sie die Datei: `git add fact_sales.sql`
3. Ändern Sie die Datei, indem Sie einen Index hinzufügen:

```


-- Index für bessere Abfrageleistung
CREATE INDEX IX_fact_sales_datum ON dwh.fact_sales(verkaufsdatum);
```

4. Jetzt haben Sie: committete Version, gestagte Version und Arbeitsverzeichnis-Version
5. Verwenden Sie `git restore fact_sales.sql` - welche Version wird wiederhergestellt?
6. Stagen Sie Ihre Änderungen und committen Sie: "Verkaufs-Faktentabelle mit Index hinzugefügt"

Teil 3: Den Stash verwenden

Schritt 1: Grundlegende Stash-Operationen

1. Erstellen Sie `datenqualitaet.sql`:

```
-- Datenqualitätsprüfungen
SELECT
    'Kunden mit fehlender E-Mail' as pruefungsname,
    COUNT(*) as problemanzahl
FROM dwh.dim_kunde
WHERE email IS NULL OR email = '';

-- Weitere Prüfungen werden hier hinzugefügt...
```

2. Fügen Sie die Datei zum Staging hinzu: `git add datenqualitaet.sql`
3. Beginnen Sie mit der Änderung von `etl_prozess.sql`, indem Sie Fehlerbehandlung hinzufügen:

```
-- Fehlerbehandlung
BEGIN TRY
    -- ETL-Logik hier
END TRY
BEGIN CATCH
    PRINT 'Fehler im ETL-Prozess: ' + ERROR_MESSAGE();
    THROW;
END CATCH;
```

4. Sie merken, dass Sie den Kontext wechseln müssen - stashen Sie Ihre Arbeit: `git stash`
5. Überprüfen Sie, dass Ihr Arbeitsverzeichnis und Staging-Bereich sauber sind: `git status`
6. Sehen Sie Ihren Stash: `git stash list`

Schritt 2: Stash-Verwaltung

1. Machen Sie eine schnelle Korrektur an `schema.sql`, indem Sie einen Kommentar hinzufügen: `-- Version 1.1`
2. Committen Sie diese Änderung: "Schema-Version aktualisiert"
3. Stellen Sie Ihre gestashte Arbeit wieder her: `git stash pop`
4. Überprüfen Sie, dass sowohl gestagte als auch ungestagte Änderungen zurück sind
5. Committen Sie Ihre Arbeit: "Datenqualitätsprüfungen und ETL-Fehlerbehandlung hinzugefügt"

Schritt 3: Mehrere Stashes

1. Erstellen Sie `backup_skript.sql`:

```
-- Datenbank-Backup-Skript
BACKUP DATABASE DataWarehouse
```

```
TO DISK = 'C:\Backups\DataWarehouse.bak'
WITH FORMAT, INIT;
```

2. Stagen Sie es: `git add backup_skript.sql`
3. Stashen Sie mit einer Nachricht: `git stash push -m "Backup-Skript in Arbeit"`
4. Nehmen Sie verschiedene Änderungen an `config.sql` vor:

```
-- Konfigurationseinstellungen
USE DataWarehouse;
SET ANSI_NULLS ON;
SET QUOTED_IDENTIFIER ON;

-- Leistungseinstellungen
SET STATISTICS IO ON;
```

5. Stashen Sie dies auch: `git stash push -m "Config-Datei-Updates"`
6. Listen Sie Stashes auf: `git stash list`
7. Wenden Sie den Backup-Skript-Stash an: `git stash apply stash@{1}`
8. Räumen Sie Stashes auf: `git stash clear`

Teil 4: Git Reset Übung

Schritt 1: Soft Reset (Commit rückgängig machen, Änderungen behalten)

1. Stellen Sie sicher, dass alle Ihre Arbeiten committed sind
2. Erstellen Sie `monitoring.sql`:

```
-- Datenbank-Monitoring-Abfragen
SELECT
    name,
    groesse_mb = CAST(size * 8.0 / 1024 AS DECIMAL(10,2))
FROM sys.master_files
WHERE database_id = DB_ID('DataWarehouse');
```

3. Committen Sie diese Datei: "Datenbank-Monitoring-Skript hinzugefügt"
4. Verwenden Sie `git log --oneline`, um Ihre Commits zu sehen
5. Führen Sie einen Soft Reset durch, um den letzten Commit rückgängig zu machen: `git reset --soft HEAD~1`
6. Überprüfen Sie `git status` - die Datei sollte gestaged sein
7. Überprüfen Sie, dass die Datei noch in Ihrem Arbeitsverzeichnis existiert
8. Re-committen Sie: "Datenbank-Monitoring-Skript hinzugefügt (korrigiert)"

Schritt 2: Mixed Reset (Standard)

1. Ändern Sie `monitoring.sql`, indem Sie eine weitere Abfrage hinzufügen:

```
-- Datenbankwachstum überprüfen
SELECT
    name,
    physical_name,
    groesse_mb = CAST(size * 8.0 / 1024 AS DECIMAL(10,2))
FROM sys.database_files;
```

2. Committen Sie: "Datenbankwachstums-Monitoring hinzugefügt"
3. Reset: `git reset HEAD~1`
4. Überprüfen Sie `git status` - Änderungen sollten im Arbeitsverzeichnis, nicht gestaged sein

Schritt 3: Hard Reset (Gefahrenzone!)

1. Fügen Sie temporären Debug-Code zu `etl_prozess.sql` hinzu:

```
-- TEMPORÄRER DEBUG-CODE - VOR PRODUKTION ENTFERNEN
PRINT 'Debug: ETL-Prozess startet';
PRINT 'Debug: Kundendaten werden verarbeitet';
-- ENDE DEBUG-CODE
```

2. Stagen und committen Sie: "Debug-Code hinzugefügt (temporär)"
3. Führen Sie Hard Reset durch: `git reset --hard HEAD~1`
4. Überprüfen Sie, dass Ihr Debug-Code vollständig verschwunden ist
5. Überprüfen Sie mit `git log --oneline`, dass der Debug-Commit verschwunden ist

Teil 5: Git Revert Übung

Schritt 1: Sicheres "Rückgängigmachen" mit Revert

1. Erstellen Sie `gespeicherte_prozeduren.sql`:

```
-- Gespeicherte Prozeduren
CREATE PROCEDURE HoleKundenverkaeufe
    @kunden_id INT
AS
BEGIN
    SELECT
        k.vorname,
        k.nachname,
        SUM(f.gesamtbetrag) as gesamtverkaeufe
    FROM dwh.dim_kunde k
    JOIN dwh.fact_sales f ON k.kunden_key = f.kunden_key
    WHERE k.kunden_id = @kunden_id
    GROUP BY k.vorname, k.nachname;
END;
```

2. Committen Sie: "Kundenverkäufe gespeicherte Prozedur hinzugefügt"
3. Bemerken Sie, dass die Prozedur einen Bug hat - revertiern Sie sie: `git revert HEAD`
4. Überprüfen Sie `git log --oneline` - Sie sollten sowohl den ursprünglichen Commit als auch den Revert-Commit sehen
5. Überprüfen Sie, dass die Datei aus Ihrem Arbeitsverzeichnis verschwunden ist

Schritt 2: Mehrere Commits revertiern

1. Erstellen Sie drei kleine Commits:
    - Fügen Sie `views.sql` mit einer einfachen Ansicht hinzu: "Kundenansicht hinzugefügt"
    - Fügen Sie `functions.sql` mit einer Datumsfunktion hinzu: "Datumsdienstfunktion hinzugefügt"
    - Ändern Sie `config.sql`, um eine Einstellung hinzuzufügen: "Konfiguration aktualisiert"
2. Verwenden Sie `git log --oneline`, um alle Commits zu sehen
3. Revertiern Sie den mittleren Commit (die Funktionen): `git revert HEAD~1`
4. Beobachten Sie, wie Git diesen selektiven Revert behandelt

Teil 6: Wiederherstellungsszenarien

Szenario 1: Versehentlich falsche Dateien gestaged

1. Erstellen Sie sowohl `temp_analyse.sql` (eine temporäre Datei) als auch `permanente_analyse.sql` (wichtige Datei)
2. Versehentlich beide stagen: `git add .`
3. Entfernen Sie nur die temp-Datei aus dem Staging: `git restore --staged temp_analyse.sql`
4. Committen Sie nur die permanente Datei

Szenario 2: Vermischte Änderungen

1. Nehmen Sie Änderungen sowohl an `schema.sql` als auch an `etl_prozess.sql` vor
2. Stagen Sie alles: `git add .`
3. Bemerken Sie, dass Sie jede Datei separat committen möchten
4. Unstagen Sie alles: `git restore --staged .`
5. Stagen und committen Sie jede Datei einzeln

Szenario 3: Zu früh committed

1. Erstellen Sie `unvollstaendiges_feature.sql` mit nur:

```
-- Neues Feature - in Arbeit
CREATE TABLE dwh.dim_zeit (
    datum_key INT PRIMARY KEY
    -- Weitere Spalten werden hinzugefügt
);
```

2. Committen Sie: "Zeitdimension hinzugefügt (unvollständig)"

3. Fügen Sie die fehlenden Spalten hinzu:

```
-- Neues Feature - in Arbeit
CREATE TABLE dwh.dim_zeit (
    datum_key INT PRIMARY KEY,
    vollstaendiges_datum DATE,
    jahr INT,
    monat INT,
    tag INT,
    tagesname NVARCHAR(10)
);
```

4. Verwenden Sie `git add` und `git commit --amend`, um den vorherigen Commit zu korrigieren
5. Überprüfen Sie mit `git log`, dass Sie immer noch nur einen Commit für dieses Feature haben

Überprüfung und Review

Überprüfen Sie Ihr Verständnis

1. Verwenden Sie `git log --oneline`, um Ihren Commit-Verlauf zu überprüfen
2. Verwenden Sie `git status`, um sicherzustellen, dass das Arbeitsverzeichnis sauber ist
3. Zählen Sie, wie oft Sie jeden Befehl verwendet haben:
    - `git restore`
    - `git restore --staged`
    - `git reset`
    - `git revert`
    - `git stash`

Herausforderungsfragen

1. Was ist der Unterschied zwischen `git reset --soft`, `git reset --mixed` und `git reset --hard`?
2. Wann würden Sie `git revert` anstelle von `git reset` verwenden?
3. Wie unterscheidet sich der Stash vom Erstellen eines Commits?
4. Was passiert mit nicht committeten Änderungen, wenn Sie `git stash` verwenden?

Erwartete Ergebnisse

Nach Abschluss dieser Übung sollten Sie:

- Den Unterschied zwischen Arbeitsbereich, Staging-Bereich und Repository verstehen
- Wissen, wie man Änderungen in verschiedenen Phasen rückgängig macht
- Mit der Verwendung des Stash für temporäre Speicherung vertraut sein
- Verstehen, wann man restore vs reset vs revert verwendet
- Übung in der Wiederherstellung von häufigen Git-Fehlern haben

Warnung: Seien Sie sehr vorsichtig mit `git reset --hard` und ähnlichen destruktiven Befehlen in echten Projekten. Stellen Sie immer sicher, dass Sie Backups haben oder dass Ihre Arbeit sicher anderswo committed ist!