Übung 3: Branch-Management und Feature-Entwicklung

Lernziele

Durch das Absolvieren dieser Übung werden Sie folgende Bereiche praktizieren:

- Erstellen und Wechseln zwischen Branches
- Verstehen der Branch-Isolation und -Unabhängigkeit
- Zusammenführen von Feature-Branches in main
- Verwalten mehrerer Features gleichzeitig
- Branch-Benennungskonventionen und bewährte Praktiken

Szenario

Sie arbeiten an einem Business Intelligence Dashboard-Projekt mit mehreren Features, die parallel entwickelt werden. Sie müssen verschiedene Feature-Branches verwalten und dabei den main-Branch für Releases stabil halten.

Setup

Initialisieren Sie das Projekt-Repository:

```
# Projektverzeichnis erstellen
mkdir BIDashboardProjekt
cd BIDashboardProjekt

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

# Standard-Branch-Namen setzen
git branch -M main
```

Teil 1: Einrichten des Main-Branches

Schritt 1: Projekt-Grundlage erstellen

Erstellen Sie die anfängliche Projektstruktur:

`README.md`:

```
# Business Intelligence Dashboard

Ein umfassendes Dashboard für Geschäftsanalysen und Berichterstattung.
```

```
## Features
- Kundenanalysen
- Verkaufsberichte
- Finanzmetriken
- Leistungs-Dashboards

## Status
- Projekt initialisiert ✓
- Kerninfrastruktur: In Bearbeitung
```

src/database/schema.sql:

```
-- Kern-Datenbankschema
CREATE DATABASE BIDashboard;
USE BIDashboard;

-- Kerntabellen
CREATE TABLE audit_log (
    id INT IDENTITY(1,1) PRIMARY KEY,
    tabellenname NVARCHAR(50),
    vorgang NVARCHAR(10),
    zeitstempel DATETIME2 DEFAULT GETDATE(),
    benutzername NVARCHAR(50)
);
```

config/datenbank_config.sql:

```
-- Datenbankkonfiguration
USE BIDashboard;

-- Standardeinstellungen setzen
SET ANSI_NULLS ON;
SET QUOTED_IDENTIFIER ON;
SET ARITHABORT ON;
```

1. Erstellen Sie die Verzeichnisstruktur: `mkdir src\database`, `mkdir config`
2. Erstellen Sie alle oben genannten Dateien
3. Stagen und committen Sie alles: "Initiale Projekteinrichtung mit Kerninfrastruktur"
4. Überprüfen Sie, dass Sie auf dem main-Branch sind: `git branch`

Teil 2: Feature-Branch-Entwicklung

Schritt 1: Kundenanalyse-Feature

1. Erstellen und wechseln Sie zu einem Feature-Branch: `git switch -c feature/kundenanalyse`
2. Überprüfen Sie, dass Sie auf dem neuen Branch sind: `git branch`

Erstellen Sie `src/database/kundentabellen.sql`:

```
-- Kundenanalyse-Tabellen
USE BIDashboard;

CREATE TABLE dim_kunde (
    kunden_key INT IDENTITY(1,1) PRIMARY KEY,
    kunden_id INT NOT NULL UNIQUE,
    vorname NVARCHAR(50),
    nachname NVARCHAR(50),
    email NVARCHAR(100),
    registrierungsdatum DATE,
    kundensegment NVARCHAR(20),
    erstellt_datum DATETIME2 DEFAULT GETDATE()
);

CREATE TABLE fact_kundenaktivitaet (
    aktivitaet_key INT IDENTITY(1,1) PRIMARY KEY,
    kunden_key INT FOREIGN KEY REFERENCES dim_kunde(kunden_key),
    aktivitaetsdatum DATE,
    seitenaufrufe INT,
    sitzungsdauer_minuten INT,
    durchgefuehrte_aktionen INT
);
```

Erstellen Sie `src/queries/kundenanalyse.sql`:

```
-- Kundenanalyse-Abfragen
USE BIDashboard;

-- Kundensegmentierungs-Analyse
SELECT
    kundensegment,
    COUNT(*) as kundenanzahl,
    AVG(DATEDIFF(day, registrierungsdatum, GETDATE())) as
durchschnittliche_tage_seit_registrierung
FROM dim_kunde
GROUP BY kundensegment;

-- Top aktive Kunden
SELECT TOP 10
    k.vorname + ' ' + k.nachname as kundenname,
    k.email,
    SUM(f.seitenaufrufe) as gesamte_seitenaufrufe,
    SUM(f.durchgefuehrte_aktionen) as gesamte_aktionen
FROM dim_kunde k
JOIN fact_kundenaktivitaet f ON k.kunden_key = f.kunden_key
GROUP BY k.kunden_key, k.vorname, k.nachname, k.email
ORDER BY gesamte_seitenaufrufe DESC;
```

3. Erstellen Sie das Queries-Verzeichnis: `mkdir src\queries`
4. Committen Sie Ihre Kundenanalyse-Arbeit: "Kundenanalyse-Tabellen und -Abfragen hinzugefügt"

Schritt 2: Verkaufsberichte-Feature (Parallele Entwicklung)

1. Wechseln Sie zurück zu main: `git switch main`
2. Erstellen Sie einen neuen Feature-Branch: `git switch -c feature/verkaufsberichte`

Erstellen Sie `src/database/verkaufstabellen.sql`:

```
-- Verkaufsberichte-Tabellen
USE BIDashboard;

CREATE TABLE dim_produkt (
    produkt_key INT IDENTITY(1,1) PRIMARY KEY,
    produkt_id INT NOT NULL UNIQUE,
    produktname NVARCHAR(100),
    kategorie NVARCHAR(50),
    unterkategorie NVARCHAR(50),
    einzelpreis DECIMAL(10,2),
    erstellt_datum DATETIME2 DEFAULT GETDATE()
);

CREATE TABLE fact_verkaeufe (
    verkauf_key INT IDENTITY(1,1) PRIMARY KEY,
    kunden_key INT, -- Wird später mit Kundentabelle verknüpft
    produkt_key INT FOREIGN KEY REFERENCES dim_produkt(produkt_key),
    verkaufsdatum DATE,
    menge INT,
    einzelpreis DECIMAL(10,2),
    rabatt_prozent DECIMAL(5,2),
    gesamtbetrag DECIMAL(10,2)
);
```

Erstellen Sie `src/queries/verkaufsberichte.sql`:

```
-- Verkaufsberichte-Abfragen
USE BIDashboard;

-- Monatliche Verkaufszusammenfassung
SELECT
    YEAR(verkaufsdatum) as verkaufsjahr,
    MONTH(verkaufsdatum) as verkaufsmonat,
    COUNT(*) as gesamttransaktionen,
    SUM(gesamtbetrag) as gesamtumsatz,
    AVG(gesamtbetrag) as durchschnittlicher_transaktionswert
FROM fact_verkaeufe
GROUP BY YEAR(verkaufsdatum), MONTH(verkaufsdatum)
ORDER BY verkaufsjahr DESC, verkaufsmonat DESC;
```

```
-- Meistverkaufte Produkte
SELECT TOP 10
    p.produktname,
    p.kategorie,
    SUM(v.menge) as verkaufte_gesamtmenge,
    SUM(v.gesamtbetrag) as gesamtumsatz
FROM dim_produkt p
JOIN fact_verkaeufe v ON p.produkt_key = v.produkt_key
GROUP BY p.produkt_key, p.produktname, p.kategorie
ORDER BY gesamtumsatz DESC;
```

3. Committen Sie die Verkaufsberichte-Arbeit: "Verkaufsberichte-Tabellen und -Abfragen hinzugefügt"

Schritt 3: Finanzmetriken-Feature

1. Wechseln Sie zurück zu main: `git switch main`
2. Erstellen Sie einen weiteren Feature-Branch: `git switch -c feature/finanzmetriken`

Erstellen Sie `src/database/finanztabellen.sql`:

```
-- Finanzmetriken-Tabellen
USE BIDashboard;

CREATE TABLE dim_konto (
    konto_key INT IDENTITY(1,1) PRIMARY KEY,
    kontocode NVARCHAR(20) NOT NULL UNIQUE,
    kontoname NVARCHAR(100),
    kontotyp NVARCHAR(20), -- Umsatz, Ausgabe, Anlage, Verbindlichkeit
    eltern_kontocode NVARCHAR(20),
    ist_aktiv BIT DEFAULT 1
);

CREATE TABLE fact_finanztransaktionen (
    transaktion_key INT IDENTITY(1,1) PRIMARY KEY,
    konto_key INT FOREIGN KEY REFERENCES dim_konto(konto_key),
    transaktionsdatum DATE,
    beschreibung NVARCHAR(200),
    soll_betrag DECIMAL(15,2),
    haben_betrag DECIMAL(15,2),
    referenznummer NVARCHAR(50)
);
```

Erstellen Sie `src/queries/finanzberichte.sql`:

```
-- Finanzberichte-Abfragen
USE BIDashboard;

-- Gewinn- und Verlustrechnung
SELECT
```

```
    k.kontotyp,
    k.kontoname,
    SUM(
        CASE
            WHEN k.kontotyp IN ('Umsatz') THEN f.haben_betrag - f.soll_betrag
            WHEN k.kontotyp IN ('Ausgabe') THEN f.soll_betrag - f.haben_betrag
            ELSE 0
        END
    ) as netto_betrag
FROM dim_konto k
JOIN fact_finanztransaktionen f ON k.konto_key = f.konto_key
WHERE k.kontotyp IN ('Umsatz', 'Ausgabe')
    AND f.transaktionsdatum >= DATEADD(month, -1, GETDATE())
GROUP BY k.kontotyp, k.kontoname
ORDER BY k.kontotyp, netto_betrag DESC;

-- Cashflow-Analyse
SELECT
    DATEPART(week, transaktionsdatum) as wochennummer,
    SUM(haben_betrag) as gesamter_zufluss,
    SUM(soll_betrag) as gesamter_abfluss,
    SUM(haben_betrag) - SUM(soll_betrag) as netto_fluss
FROM fact_finanztransaktionen
WHERE transaktionsdatum >= DATEADD(month, -1, GETDATE())
GROUP BY DATEPART(week, transaktionsdatum)
ORDER BY wochennummer;
```

3. Committen Sie die Finanzmetriken-Arbeit: "Finanzmetriken-Tabellen und -Berichte hinzugefügt"

Teil 3: Branch-Inspektion und -Vergleich

Schritt 1: Branch-Unterschiede untersuchen

1. Listen Sie alle Branches auf: `git branch`
2. Wechseln Sie zu main: `git switch main`
3. Listen Sie Dateien in src/database auf: `dir src\database`
4. Wechseln Sie zu Kundenanalyse: `git switch feature/kundenanalyse`
5. Listen Sie Dateien erneut auf: `dir src\database`
6. Bemerken Sie, wie sich der Dateiinhalt zwischen Branches ändert

Schritt 2: Branch-Verlauf ansehen

1. Sehen Sie den Commit-Verlauf des aktuellen Branches: `git log --oneline`
2. Wechseln Sie zu Verkaufsberichten: `git switch feature/verkaufsberichte`
3. Vergleichen Sie den Commit-Verlauf: `git log --oneline`
4. Wechseln Sie zu main und vergleichen Sie: `git switch main` dann `git log --oneline`

Schritt 3: Branch-Inhalte vergleichen

Vergleichen Sie vom main-Branch aus, was unterschiedlich ist:

1. `git diff main feature/kundenanalyse --name-only`
2. `git diff main feature/verkaufsberichte --name-only`
3. `git diff main feature/finanzmetriken --name-only`

Teil 4: Features zusammenführen

Schritt 1: Kundenanalyse zusammenführen

1. Stellen Sie sicher, dass Sie auf main sind: `git switch main`
2. Führen Sie das Kundenanalyse-Feature zusammen: `git merge feature/kundenanalyse`
3. Überprüfen Sie den Commit-Verlauf: `git log --oneline`
4. Überprüfen Sie, dass die neuen Dateien vorhanden sind: `dir src\database`

Schritt 2: README nach erstem Merge aktualisieren

Aktualisieren Sie `README.md`, um das abgeschlossene Feature zu reflektieren:

```
# Business Intelligence Dashboard

Ein umfassendes Dashboard für Geschäftsanalysen und Berichterstattung.

## Features
- Kundenanalysen ✓
- Verkaufsberichte: In Bearbeitung
- Finanzmetriken: In Bearbeitung
- Leistungs-Dashboards: Geplant

## Status
- Projekt initialisiert ✓
- Kerninfrastruktur ✓
- Kundenanalyse-Feature ✓
```

Committen Sie diese Aktualisierung: "README nach Kundenanalyse-Abschluss aktualisiert"

Schritt 3: Merge-Konflikt-Auflösung testen

1. Wechseln Sie zum Verkaufsberichte-Branch: `git switch feature/verkaufsberichte`
2. Ändern Sie die README.md in diesem Branch:

```
# Business Intelligence Dashboard

Ein umfassendes Dashboard für Geschäftsanalysen und Berichterstattung.

## Features
- Kundenanalysen
- Verkaufsberichte ✓
- Finanzmetriken: In Bearbeitung
- Leistungs-Dashboards: Geplant
```

```
## Status
- Projekt initialisiert ✓
- Kerninfrastruktur ✓
- Verkaufsberichte-Feature ✓
```

3. Committen Sie: "README für Verkaufsberichte-Abschluss aktualisiert"
4. Wechseln Sie zu main: `git switch main`
5. Versuchen Sie zusammenzuführen: `git merge feature/verkaufsberichte`
6. Sie sollten einen Merge-Konflikt in README.md bekommen
7. Öffnen Sie README.md in einem Texteditor und lösen Sie den Konflikt, indem Sie beide Aktualisierungen kombinieren:

```
# Business Intelligence Dashboard

Ein umfassendes Dashboard für Geschäftsanalysen und Berichterstattung.

## Features
- Kundenanalysen ✓
- Verkaufsberichte ✓
- Finanzmetriken: In Bearbeitung
- Leistungs-Dashboards: Geplant

## Status
- Projekt initialisiert ✓
- Kerninfrastruktur ✓
- Kundenanalyse-Feature ✓
- Verkaufsberichte-Feature ✓
```

8. Stagen Sie die aufgelöste Datei: `git add README.md`
9. Schließen Sie den Merge ab: `git commit -m "Verkaufsberichte-Feature mit Konfliktauflösung zusammengeführt"`

Teil 5: Branch-Aufräumen und -Verwaltung

Schritt 1: Zusammengeführte Branches aufräumen

1. Listen Sie alle Branches auf: `git branch`
2. Löschen Sie den zusammengeführten Kundenanalyse-Branch: `git branch -d feature/kundenanalyse`
3. Löschen Sie den zusammengeführten Verkaufsberichte-Branch: `git branch -d feature/verkaufsberichte`
4. Überprüfen Sie, dass die Branches gelöscht wurden: `git branch`

Schritt 2: Mit verbleibendem Feature fortfahren

1. Wechseln Sie zu Finanzmetriken: `git switch feature/finanzmetriken`
2. Überprüfen Sie, ob es auf dem neuesten Stand mit mains neuesten Änderungen ist
3. Falls nötig, führen Sie main in Ihren Feature-Branch zusammen: `git merge main`

4. Wechseln Sie zurück zu main: `git switch main`
5. Führen Sie Finanzmetriken zusammen: `git merge feature/finanzmetriken`
6. Löschen Sie den zusammengeführten Branch: `git branch -d feature/finanzmetriken`

Schritt 3: Finaler Projektzustand

Aktualisieren Sie die finale README.md:

```
# Business Intelligence Dashboard

Ein umfassendes Dashboard für Geschäftsanalysen und Berichterstattung.

## Features
- Kundenanalysen ✓
- Verkaufsberichte ✓
- Finanzmetriken ✓
- Leistungs-Dashboards: Nächste Phase

## Status
- Projekt initialisiert ✓
- Kerninfrastruktur ✓
- Alle Kernfeatures abgeschlossen ✓

## Nächste Schritte
- Leistungs-Dashboards hinzufügen
- Datenvisualisierungs-Komponenten erstellen
- Automatisierte Berichterstattung einrichten
```

Committen Sie: "Finales README-Update - alle Kernfeatures abgeschlossen"

Teil 6: Erweiterte Branch-Szenarien

Herausforderung 1: Hotfix-Branch

1. Erstellen Sie einen Hotfix-Branch: `git switch -c hotfix/schema-bug-fix`
2. "Beheben" Sie einen Bug in `config/datenbank_config.sql`, indem Sie hinzufügen:

```
-- Datenbankkonfiguration
USE BIDashboard;

-- Standardeinstellungen setzen
SET ANSI_NULLS ON;
SET QUOTED_IDENTIFIER ON;
SET ARITHABORT ON;

-- Fix: Fehlende Isolationsebenen-Einstellung hinzugefügt
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

3. Committen Sie: "Fehlende Transaktions-Isolationsebenen-Einstellung behoben"

4. Führen Sie in main zusammen: `git switch main` dann `git merge hotfix/schema-bug-fix`
5. Löschen Sie den Hotfix-Branch: `git branch -d hotfix/schema-bug-fix`

Herausforderung 2: Experimenteller Branch

1. Erstellen Sie einen experimentellen Branch: `git switch -c experiment/neue-reporting-engine`
2. Erstellen Sie `src/experimental/neue_engine.sql`:

```
-- Experimentelle neue Reporting-Engine
-- Dies ist nur ein Proof of Concept

CREATE VIEW v_vereinheitlichte_metriken AS
SELECT
    'Kunde' as metrik_typ,
    COUNT(*) as metrik_wert,
    'Aktive Kunden' as metrik_beschreibung
FROM dim_kunde
WHERE kundensegment IS NOT NULL

UNION ALL

SELECT
    'Verkäufe' as metrik_typ,
    SUM(gesamtbetrag) as metrik_wert,
    'Gesamtumsatz' as metrik_beschreibung
FROM fact_verkaeufe;
```

3. Committen Sie: "Experimentelle vereinheitlichte Metriken-Ansicht hinzugefügt"
4. Wechseln Sie zurück zu main: `git switch main`
5. Entscheiden Sie, dieses Experiment nicht zusammenzuführen (Branch für später behalten): `git branch`

Überprüfung und Review

Überprüfung Ihrer Arbeit

1. Überprüfen Sie finale Branch-Liste: `git branch`
2. Betrachten Sie vollständige Projektgeschichte: `git log --oneline --graph`
3. Listen Sie alle Dateien in Ihrem Projekt auf: `git ls-files`
4. Überprüfen Sie, dass der main-Branch alle zusammengeführten Features hat

Verständnischeck

Beantworten Sie diese Fragen:

1. Wie viele Branches haben Sie insgesamt erstellt?
2. Wie viele Branches haben Sie in main zusammengeführt?
3. Was passiert mit Dateien, wenn Sie zwischen Branches wechseln?
4. Warum ist es wichtig, zu main zu wechseln, bevor Sie neue Feature-Branches erstellen?
5. Was ist der Unterschied zwischen `git branch` und `git checkout -b`?

Erwartete Ergebnisse

Nach Abschluss dieser Übung sollten Sie:

- Verstehen, wie Branches isolierte Entwicklungsumgebungen bereitstellen
- Wissen, wie man Branches erstellt, wechselt und zusammenführt
- Erfahrung mit Merge-Konflikten und deren Auflösung haben
- Branch-Aufräumen und -Wartung verstehen
- Mit parallelen Feature-Entwicklungs-Workflows vertraut sein

Bewährte Praktiken gelernt

- Immer Feature-Branches von einem aktuellen main-Branch erstellen
- Beschreibende Branch-Namen mit Präfixen verwenden (feature/, hotfix/, experiment/)
- Feature-Branches auf einzelne Funktionalitäten fokussiert halten
- Zusammengeführte Branches löschen, um Repository sauber zu halten
- Merge-Konflikte sorgfältig durch Verstehen beider Änderungen lösen
- Branches vor Zusammenführung in main testen

Hinweis: In echten Projekten würden Sie normalerweise Pull Requests für Code-Reviews verwenden, bevor Feature-Branches in main zusammengeführt werden.