

Dokumentation Sakila DB

Aufgabe 2.1.2 – Aufgabe 2.1.4

2.1.2 Spezielle Integritätsbedingungen

Das Problem

In unserem Datenmodell wollten wir Kunden verschiedenen Diensten zuordnen: Subscription, Package oder Spot Watching (einzelner Film). Jedes dieser Angebote hat eine eigene Tabelle – aber in der `srv_customer_allocation`-Tabelle sollte trotzdem nur ein gemeinsames Fremdschlüsselfeld (`srv_reference_id`) verwendet werden. Das Problem: In SQL kann ein Fremdschlüssel nicht gleichzeitig auf mehrere Tabellen zeigen.

Der Ansatz

Wir wollten eine polymorphe Beziehung abbilden – also eine Beziehung, bei der ein Feld (hier `srv_reference_id`) auf unterschiedliche Tabellen zeigen kann, je nachdem, um welchen Typ (`service_type_id`) es sich handelt. So etwas kennt man aus objektorientierter Programmierung, ist in relationalen Datenbanken aber nicht direkt vorgesehen.

Die Lösung

Statt auf klassische Fremdschlüssel zu setzen, haben wir einen Trigger mit einer Validierungsfunktion erstellt. Diese Funktion prüft beim Einfügen oder Aktualisieren eines Datensatzes, ob `srv_reference_id` in der *richtigen* Tabelle existiert – abhängig vom Wert von `service_type_id`:

1 → muss in subscription existieren

2 → muss in package existieren

3 → muss in content_stream existieren

Wenn das nicht zutrifft, wird ein Fehler geworfen. Damit erreichen wir logische Konsistenz, ohne dass SQL polymorphe Fremdschlüssel nativ unterstützen muss.

2.1.3 Weitere Integritätsbedingungen

1. Keine Mehrfachvergabe derselben Lizenzrechte für den gleichen Zeitraum

Problem:

Ein Inhalt darf nicht zur gleichen Zeit in zwei verschiedenen Lizenzverträgen (package, subscription, etc.) parallel aktiv sein. Wenn eine Subscription nach dem Kauf eines packages erfolgt, dann sollte ebenfalls eine Teilerstattung folgen, da der Inhalt über die subscription gedeckt ist.

Warum kein SQL-Constraint möglich:

Das benötigt eine Überschneidungsprüfung von Zeiträumen (z. B. start_date, end_date) auf gleichem Inhalt, was nur über Trigger oder Queries mit Zeitlogik geht.

2. Eindeutige und lückenlose Folgenummerierung in Serien

Problem:

Innerhalb einer Serie (und deren Seasons) sollen season_number und episode_number eindeutig und sinnvoll fortlaufend sein.

Warum kein SQL-Constraint möglich:

Ein UNIQUE-Constraint kann Mehrfachverwendung verhindern, aber keine logische Reihenfolge oder Lückenfreiheit garantieren. Diese muss durch zusätzliche Logik geprüft werden.

3. Content muss einem gültigen Franchise-Typ entsprechen

Problem:

Ein Inhalt in einem Franchise muss z. B. zu dessen Genre oder Medium passen (z. B. keine Podcast-Episode in einem Film-Franchise).

Warum kein SQL-Constraint möglich:

Hier ist eine semantische Validierung nötig, die von anderen Tabellen (z. B. franchise, genre, content_type) abhängig ist – das erfordert mehrschrittige Prüfungen.

2.1.4 Zusätzliche Indizes

1. Kategorisierung bei Inhalte-Zuordnung („Dramedy“)

Die Abfrage zum Einfügen der neuen Kategorie „Dramedy“ basiert auf mehrfachen JOINS und Filterungen über die Tabellen content_category und category. Damit diese effizienter ausgeführt werden, sollten folgende Indizes noch angelegt werden:

category(name)

→ Für schnellere Auswahl der Kategorie-ID per Name (WHERE name = 'Drama')

content_category(content_id, category_id)

→ Optimiert die mehrfach verwendeten JOINS

Optional: content_category(category_id)

→ Falls nach einzelnen Kategorien gefiltert wird

2. Lieblingskategorie je Kunde (View v_cust_fav_content_cat)

Diese View arbeitet mit mehreren Tabellen und aggregiert die Inhalte pro Kunde. Dabei werden unter anderem cust_watch_act, content_stream, content_category und category verwendet. Um hier die Performance zu verbessern, müssen folgende Indizes ergänzt werden:

cust_watch_act(customer_id, content_id)

→ Optimiert sowohl JOINS als auch die spätere Aggregation

content_category(content_id, category_id)

→ Unterstützt JOINS bei der Kategoriezählung

3. Abrechnungen nach Kunde und Datum (View v_billing_cust_date)

Für die Summierung der Abrechnungspositionen pro Kunde und Datum ist ein effizienter Zugriff auf Rechnungs- und Positionstabellen nötig. Daher sollten auch hier folgende Indizes noch erstellt werden:

billing_item(billing_id)

→ Optimiert den JOIN zwischen Kopf- und Positionsdaten

billing_head(customer_id)

→ Unterstützt die Gruppierung nach Kunde

Optional: billing_head(billing_date)

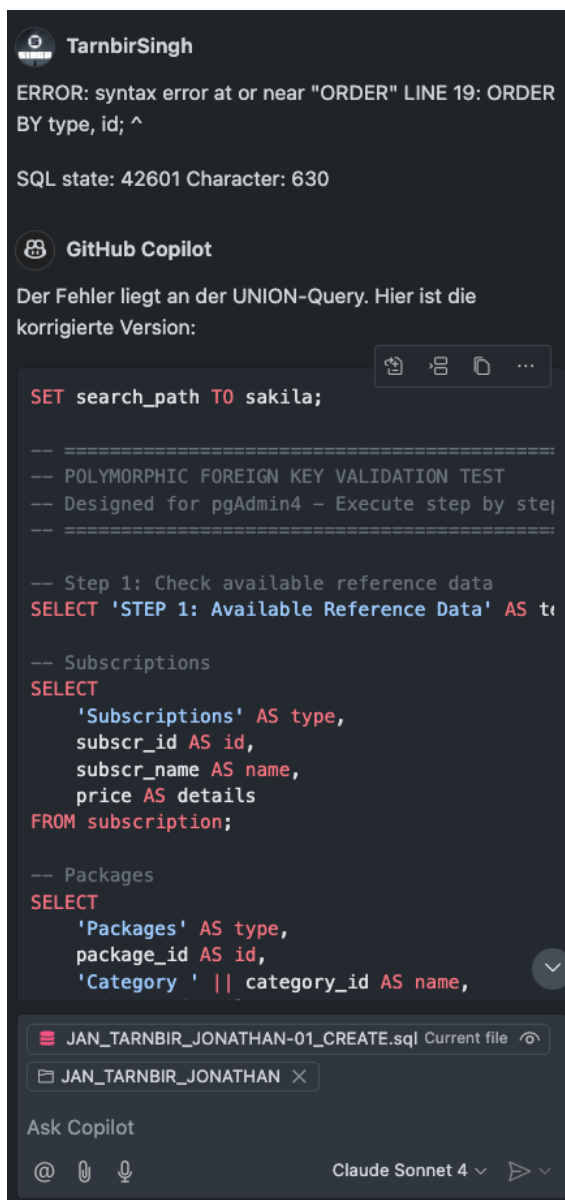
→ Erleichtert zeitbasierte Auswertungen

KI Nutzung:

Während des Projekts wurde KI vor allem unterstützend eingesetzt. Sie half bei der Suche nach Fehlern in SQL-Statements und lieferte sinnvolle Hinweise zur Behebung von Problemen. Außerdem war sie nützlich, um aus dem ER-Diagramm erste SQL-Statements grob zu skizzieren, was den Einstieg in die Umsetzung erleichterte. Auch bei der Überprüfung auf inhaltliche Konsistenz der Statements und Datenbanklogik war sie sehr hilfreich. Die KI diente dabei als praktische Ergänzung zur eigenen Arbeit, besonders, wenn man mal nicht weiter wusste.

Genutzt haben wir unter anderem GPT-4 oder Claude Sonnet 4 über den Browser oder über den GitHub Copilot.

Beispiel:



The screenshot shows a code editor interface with a dark theme. At the top, a user profile for 'TarnbirSingh' is visible. Below it, an error message states: 'ERROR: syntax error at or near "ORDER" LINE 19: ORDER BY type, id; ^'. The SQL state is '42601' and the character position is '630'. Below the error, the 'GitHub Copilot' logo is shown, followed by the text: 'Der Fehler liegt an der UNION-Query. Hier ist die korrigierte Version:'. A code block contains the corrected SQL query. The query starts with 'SET search_path TO sakila;' and includes several comments. The main query is a SELECT statement with multiple parts. The first part is 'SELECT 'STEP 1: Available Reference Data' AS type, ...'. The second part is 'Subscriptions' with a SELECT statement: 'SELECT 'Subscriptions' AS type, subscr_id AS id, subscr_name AS name, price AS details FROM subscription;'. The third part is 'Packages' with a SELECT statement: 'SELECT 'Packages' AS type, package_id AS id, 'Category ' || category_id AS name, ...'. The bottom of the editor shows a file explorer with 'JAN_TARNBIR_JONATHAN-01_CREATE.sql' as the current file, and a search bar with 'JAN_TARNBIR_JONATHAN'. At the very bottom, there is a 'Ask Copilot' button and a status bar showing 'Claude Sonnet 4'.

```
SET search_path TO sakila;

-- =====
-- POLYMORPHIC FOREIGN KEY VALIDATION TEST
-- Designed for pgAdmin4 - Execute step by step
-- =====

-- Step 1: Check available reference data
SELECT 'STEP 1: Available Reference Data' AS type, ...

-- Subscriptions
SELECT
    'Subscriptions' AS type,
    subscr_id AS id,
    subscr_name AS name,
    price AS details
FROM subscription;

-- Packages
SELECT
    'Packages' AS type,
    package_id AS id,
    'Category ' || category_id AS name,
```



TarnbirSingh

why am i getting error because of srv_reference_is: SQL state: 23503

JAN_TARNBIR_JONATHAN

JAN_TARNBIR_JONATHAN-03_MIGRATE.sql:6-15

JAN_TARNBIR_JONATHAN-03_MIGRATE.sql:6-15



GitHub Copilot

> Used 1 reference

> Searched JAN_TARNBIR_JONATHAN for
"srv_reference_id", 5 results

Ah, jetzt sehe ich das Problem! 🔍

Sie haben in Ihrer `01_CREATE.sql` noch die **alten FK-Constraints**, die wir entfernen sollten:

```
CONSTRAINT fk_sca_content_stream
  FOREIGN KEY (srv_reference_id) REFERENCES coi
CONSTRAINT fk_sca_subscription
  FOREIGN KEY (srv_reference_id) REFERENCES sul
CONSTRAINT fk_sca_package
  FOREIGN KEY (srv_reference_id) REFERENCES par
```

SQL State 23503 = Foreign Key Constraint Violation!

Problem:

