



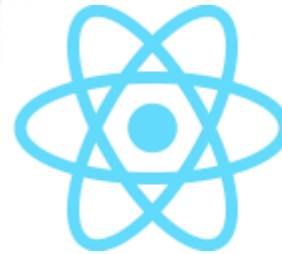
Datenbankdesign für Entwickler: Tabellen sind doch keine JSON-Container

Thorsten Kansy



Meine Person- Thorsten Kansy

Freier Consultant, Software Architekt,
Entwickler, Trainer & Fachautor



Azure Cosmos DB



Mein Service- Ihr Benefit

- Individuelle Inhouse Trainings
- (Online on-demand) Projektbegleitung
- Beratung
 - Problemanalyse und Lösungen
 - Technologieentscheidungen



Agenda

- Aber, die Datenbank läuft doch!
- Das Missverständnis: .NET und SQL Server!
- Anti-Pattern
- Sieben Regeln für Entwickler

A large, ancient, woven basket or boat, partially submerged in shallow water, with a stone wall and trees in the background. The basket is made of dark, woven material and has a curved, bowl-like shape. It is tilted, with its rim facing upwards. The water is clear and shallow, reflecting the sky and the surrounding environment. In the background, there is a stone wall made of large, irregular stones, and several trees with green foliage. The sky is blue with some white clouds. The overall scene suggests an archaeological site or a natural setting with historical significance.

Aber, die Datenbank läuft doch!

Leitfragen

Ist die Datenbank nur ein Ablageort für
Objekte?

Oder ist sie ein aktiver Teil der
Anwendungsarchitektur?

Typische Aussage, wer kenn sie nicht?

„Die Anwendung funktioniert seit Jahren.“

„Das Datenmodell können wir später aufräumen.“

„EF Core/ ORM kümmert sich schon darum.“

Die üblichen Verdächtigen

- Unpassende Datentypen in Spalten
- Datentypen
 - `varchar(max)`, `nvarchar(max)`, `decimal(x, 0)`
 - Kein einziges `bit`
 - `sql_variant`
- Zuviel (Geschäfts-)Logik in T-SQL
- (Geschäfts-)daten (überwiegende) als JSON/ XML

Ergo

Schlechtes Datenbankdesign fällt häufig erst auf, wenn Datenmenge, Benutzeranzahl und Änderungsdruck steigen.



Das Missverständnis: .NET und SQL Server!

Leitfrage

Soll sich die Datenbank nach der App richten
oder umgekehrt?

Verschiedene Sichtweisen

- Entwickler denken häufig in
 - Objekten
 - Klassenhierarchien
 - Collections
 - Serialisierung
 - flexiblen Datenstrukturen
- Der SQL Server arbeitet mit
 - Relationen
 - Mengen
 - Datentypen
 - Statistiken
 - Constraints
 - Zugriffspfade

ORM ≠ Magie

- Ein ORM beseitigt diesen Unterschied nicht
- ORM übersetzt Code in SQL, entwirft aber nicht automatisch ein gutes Datenmodell
- Datenbankdesign ist keine Aufgabe, die man vollständig an Migrationen oder Konventionen delegieren sollte

Ergo

Das Objektmodell und das relationale Modell
dürfen unterschiedlich aussehen

Anti-Pattern





„Tabellen als JSON-Container“

Warum JSON so attraktiv ist?

- Ein Objekt kann direkt serialisiert werden
- Keine Schemaänderung notwendig
- Neue Eigenschaften können sofort gespeichert werden
- Unterschiedliche Objekttypen passen in dieselbe Tabelle
- Das Datenmodell scheint „flexibel“ und „zukunftsicher“

Eigentlich werden Designentscheidungen nur verschoben

Probleme

- Die Datenbank kennt die fachliche Struktur nicht
- Pflichtfelder können nicht zuverlässig erzwungen werden
- Datentypen sind nur noch Konvention
- Indizes müssen speziell geplant werden
- Statistiken und Kardinalitätsschätzungen werden schwieriger
- Änderungen an der JSON-Struktur erzeugen faktisch neue Schema-Versionen – nur ohne sichtbares Datenbankschema
- Fehler werden aus der Datenbank in den Anwendungscode verschoben
- Mehrere Anwendungen können dasselbe JSON unterschiedlich interpretieren

Wann JSON sinnvoll sein kann

- Seltene, optionale Zusatzattribute
- Eigenschaften, nach denen nicht regelmäßig (nie?) gesucht oder gruppiert wird
- Importierte Rohdaten
- Nachrichten- oder Ereignis-Payloads
- Auditinformationen
- Übergangs- und Integrationsszenarien

Gute Kompromisslösung

- Häufig verwendete fachliche Eigenschaften als typisierte Spalten
- Beziehungen relational modellieren
- Optionale Zusatzinformationen in einer JSON-Spalte
- JSON nicht als Ersatz für das komplette Datenmodell verwenden



„Strings passen immer“

Nicht immer String, aber wer kennt das nicht

- Datum als (N) VARCHAR, INT, ...
- INT als DECIMAL (x, 0)
- CHAR (1), INT, BIT, ...
- Geldbetrag als FLOAT
- Status als freier Text
- IDs als NVARCHAR (100)
- Telefonnummer, Postleitzahl und Betrag...
- Vorsorglich überall NVARCHAR (MAX)

Probleme

- Ungültige Werte können gespeichert werden.
- Sortierungen liefern fachlich falsche Ergebnisse.
- Berechnungen benötigen Konvertierungen.
- Implizite Konvertierungen können Indexzugriffe verhindern.
- Speicherbedarf und Indexgröße steigen.
- Vergleiche hängen eventuell von Collation und Formatierung ab.
- Jede Anwendung muss dieselben Konvertierungsregeln implementieren.

A photograph of a narrow, colorful street in a hillside neighborhood. The street is paved with stone steps, and the buildings on either side are painted in various bright colors like blue, orange, and pink. Two people are walking up the steps, and a blue metal railing is visible in the foreground. The text "„Constraints sind nur lästig“" is overlaid on the image in white font on a dark blue background.

„Constraints sind nur lästig“

Probleme

- Ungültige/ fehlende Daten
- Doppelte/ verwaiste Datensätze
- Inkonsistente Beziehungen
- Verteilte Geschäftsregeln
- Umgehbare Validierung
- Schlechtere Datenqualität
- Aufwendige Datenbereinigung
- Schwierigere Fehlersuche
- Weniger Wissen für den Optimizer

Entschuldigungen

- „Das prüft bereits die API.“
- „EF Core stellt sicher, dass die Daten korrekt sind.“
- „Fremdschlüssel kosten Performance.“
- „Die Daten werden nur von unserer Anwendung geschrieben.“
- „Für Tests brauchen wir flexible Daten.“

Warum die Prüfung in API nicht genügt

- Datenimporte
- Administrationsskripte
- Hintergrunddienste
- mehrere Anwendungsversionen
- direkte Datenkorrekturen
- Fehler und Race Conditions
- zukünftige Anwendungen, die heute noch nicht bekannt sind



„EAV- Das universelle Datenmodell“

EAV Entity-attribute-value

EAV speichert Eigenschaften einer Entität nicht als feste Spalten, sondern zeilenweise als Attribut-Wert-Paare.

EntityID	AttributeID	Value
1001	Name	Müller AG
1001	AnzahlMitarbeiter	250
1001	Jahresumsatz	12500000.50
1001	IsAktiv	True
1002	Name	Meyer GmbH
1002	AnzahlMitarbeiter	10

EAV eignet sich für flexible, selten einheitliche Attribute

Probleme

- Fachliche Bedeutung steht nicht mehr im Schema
- Datentypen werden aufgeweicht
- Abfragen benötigen viele Joins oder Pivot-Operationen
- Constraints sind kaum möglich
- Reporting ist schwierig
- Refactoring wird zum Risiko oder ist quasi nicht möglich
- Fehler erscheinen erst zur Laufzeit
- Die angebliche Flexibilität erzeugt Komplexität an jeder anderen Stelle

Ähnliche Ansätze

- Eine Tabelle für alle Arten von Entitäten
 - Generische Namen
 - Objects, Properties, Data, Values
- Tabelle(n) mit generischen Spalten
 - TextValue1 ... TextValueN
 - NumberValue1... NumberValueN



„Die App erledigt die Datenbankarbeit“

Typische Beispiele

- Alle Datensätze laden und anschließend im Code filtern
- Aggregationen im Anwendungsspeicher durchführen
- Schleifen mit einer Abfrage je Datensatz
- Beziehungen nachträglich im Anwendungscode zusammensetzen
- Sortierung und Paging nach dem Materialisieren
- komplette Entitäten laden, obwohl nur drei Spalten benötigt werden
- JSON deserialisieren, um anschließend nach einem Feld zu filtern

Aber warum ist das problematisch?

- unnötige Datenübertragung
- zusätzliche Roundtrips
- hoher Speicherverbrauch
- längere Sperr- und Verarbeitungszeiten
- schlechte Skalierung
- die Datenbank kann ihre Optimierungsfähigkeiten nicht nutzen

Speziell: .NET-/EF-Core-Bezug

- `IQueryable` und `IEnumerable` unterscheiden
- Projektionen gezielt einsetzen
- nur benötigte Spalten laden
- Filter, Join, Aggregation und Paging möglichst in SQL ausführen
- generiertes SQL kontrollieren (immer!)
- nicht jede Performancefrage vorschnell mit `AsNoTracking()` beantworten
- Datenbankdesign und Abfragedesign gemeinsam betrachten

The image features the Christ the Redeemer statue in Rio de Janeiro, Brazil, set against a backdrop of a bright blue sky filled with soft, white clouds. The statue is shown from the waist up, with its arms outstretched in a gesture of blessing. It is positioned centrally in the upper half of the frame. A dark blue horizontal band spans the width of the image, partially obscuring the statue's torso and serving as a background for the title text.

„Sieben Regeln für Entwickler“

Besser früher als später

1. Daten mit dem fachlich korrekten Datentyp speichern
2. Wichtige fachliche Regeln durch Constraints absichern
3. Beziehungen als Beziehungen modellieren
4. JSON gezielt einsetzen, nicht als Standardlösung
5. Abfragen von den erwarteten Zugriffsmustern her denken
6. Das von EF Core erzeugte SQL prüfen
7. Datenbankdesign als Teil der Softwarearchitektur behandeln

Fragen? Jetzt oder später!



Kontakt

 **E-Mail**
tkansy@dotnetconsulting.eu

 **Telefon**
[+49 \(0\) 6187 / 2009090](tel:+49(0)61872009090)

 **Microsoft Teams**
[Meet now](#)

 **LinkedIn**
[Link me](#)

 **XING**
[Xing me](#)

 **X (Twitter)**
[@tkansy](#)



Wir bitten um dein Feedback

Jetzt bewerten!



entwickler.de

www.dotnetconsulting.eu

SQL Server meets .NET (Core)- professionally!



Ich berate, coache und trainiere im Bereich Entwicklung von .NET (Core) Anwendungen mit Microsoft SQL Server- mit Allem, was dazu gehört- und was man vielleicht weglassen sollte.

