

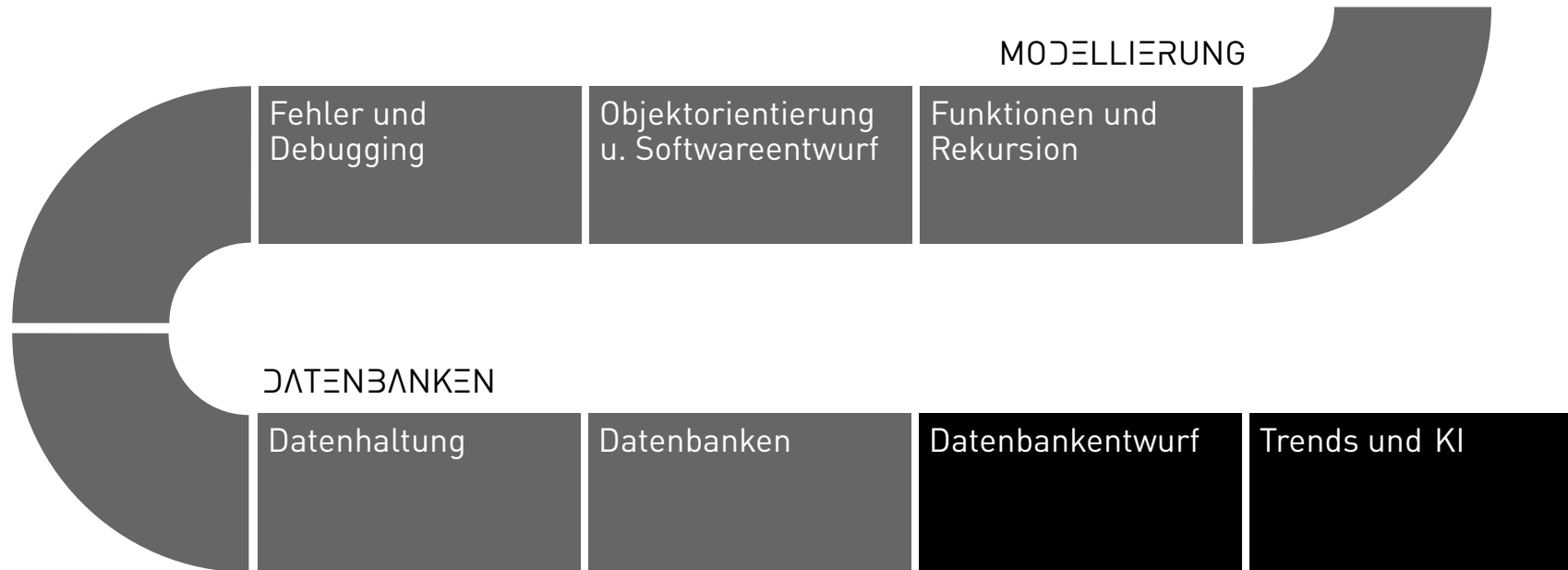
programmierung und datenbanken

Joern Ploennigs

SQL Select

MIDJOURNEY: DEEP DIVE IN A WIMMELBILD

ABLAUF



WIE ERHÄLT MAN DATEN AUS DEN TABELLEN?

- Die Tabellen werden nicht wie Dateien sequentiell ausgelesen, sondern wir führen *Anfrageoperationen* aus
- Diese berechnen aus einer Menge an Tabellen eine neue *Ergebnistabelle*
- Diese Operationen sind dabei ähnlich wie bekannte Rechenarten kombinierbar
- Wir nutzen also ein System aus Berechnungen über Tabellen: eine *Relationenalgebra*

WIE FÜHREN WIR DIESE IN DER PRAXIS AUS?

- Historisch haben sich hier eigene *Anfragesprachen* entwickelt, anstatt nur Zusatz-Funktionen und Bibliotheken in anderen Programmiersprachen
- Die populärste dieser Sprachen ist die *Structured Query Language (SQL)*
- Hat eine Syntax die speziell auf die hier erlaubten Operationen ausgelegt ist
- Damit in der Anwendung deutlich fokussierter als Python

SQL (STRUCTURED QUERY LANGUAGE)

- 1975 SEQUEL = Structured English Query Language
- *seit 1986* SQL ANSI-Standard, *seit 1987* ISO-Standard
- 1992 SQL 2 (SQL-92) neuer ISO-Standard
- 1999 SQL 3 (SQL:1999) neuer ISO-Standard
- 2016 SQL 7 (SQL:2011) neuer ISO-Standard
- 2019 SQL 9 (SQL:2019) stable release
- *SQL:2016* ISO/IEC 9075-1:2016 ist der derzeitige ISO-Standard

PROJEKTION

- Gegeben sei die Relation Gemeinden {GemeindeID, Name, Einwohner}
- *Projektion* bedeutet das Auswählen bestimmter Spalten durch explizite Auflistung
- Beispiel: GemeindeID, Name

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
2	Graal-Müritz	4.278
3	Sanitz	5.831

PROJEKTION - IN SQL

Syntax:

```
SELECT ... FROM ...
```

- *SELECT* – Selektiert die benötigten Spalten
(mit * werden alle Spalten gewählt)
- *FROM* – Legt die Relationen fest, aus denen
selektiert wird

```
SELECT GemeindeID, Name  
FROM Gemeinden
```

Ergebnis:

GemeindeID	Name
1	Dummerstorf
2	Graal-Müritz
3	Sanitz

SELEKTION

- *Selektion* bedeutet das Auswählen bestimmter Zeilen anhand einer Bedingung
- Für die Bedingungen werden wie gehabt Vergleichsoperatoren genutzt
- Bedingung: Einwohner > 5000

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
2	Graal-Müritz	4.278
3	Sanitz	5.831

SELEKTION - IN SQL

Syntax:

```
SELECT ... FROM ... WHERE ...
```

- *WHERE* – Filtert in den ausgewählten Tabellen nach bestimmten Bedingungen

Beispiel:

```
SELECT * FROM Gemeinden  
WHERE Einwohner > 5000
```

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
3	Sanitz	5.831

NATÜRLICHER VERBUND

Bei einem *JOIN* kombinieren wir die Einträge aus beiden Tabellen da wo immer Fremdschlüssel und Primärschlüssel übereinstimmen

Gemeinden Tabelle

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
2	Graal-Müritz	4.278
3	Sanitz	5.831

Primärschlüssel: GemeindeID

Bauwerke Tabelle

BauwerksID	Bauwerkstyp	GemeindeID
5000	Tankstelle	2
5001	Hotel	1
5002	Kirche	2

Fremdschlüssel: GemeindeID

NATÜRLICHER VERBUND - ERGEBNIS

Verknüpfung von Tabellen durch übereinstimmende Spalten:

Gemeinden Tabelle

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
2	Graal-Müritz	4.278
3	Sanitz	5.831

Bauwerke Tabelle

BauwerksID	Bauwerkstyp	GemeindeID
5000	Tankstelle	2
5001	Hotel	1
5002	Kirche	2

Primärschlüssel: GemeindeID

Fremdschlüssel: GemeindeID

Ergebnis

BauwerksID	Bauwerkstyp	GemeindeID	Name	Einwohner
5000	Tankstelle	2	Graal-Müritz	4.278
5001	Hotel	1	Dummerstorf	7.329
5002	Kirche	2	Graal-Müritz	4.278

Wichtig: Tupel, deren Attributwert in der jeweiligen Spalte der anderen Relation nicht auftaucht, werden auch in der Ergebnistabelle nicht auftauchen.

Im Beispiel: Sanitz taucht im Ergebnis nicht auf, weil keines der vorhandenen Bauwerke hier gebaut ist.

NATÜRLICHER VERBUND - IN SQL

Syntax:

```
SELECT ... FROM ... NATURAL JOIN
```

- *NATURAL JOIN* – Verbindet zwei Tabellen über den natürlichen Verbund

Beispiel:

```
SELECT * FROM Bauwerke  
NATURAL JOIN Gemeinden
```

WEITERE VERBUNDFORMEN IN SQL

Kreuzprodukt:

```
SELECT ... FROM Table1, Table2
```

- Separiert man statt mit NATURAL JOIN mit einem Komma, wird ein *Kreuzprodukt* gebildet – alle Zeilen werden mit allen Zeilen verbunden!

Sinnvoll um mehrere Tabellen nach bestimmten Bedingungen zu kombinieren:

```
SELECT ... FROM Table1, Table2  
WHERE Table1.Foreign = Table2.Primary
```

Kann damit auch den Natürlichen Verbund erzeugen

UMBENENNUNG

- Manchmal haben die gleichen Attributwerte in verschiedenen Tabellen verschiedene Bedeutungen, besonders beim Erzeugen von Verbunden
- Gegeben Relation Bauwerke {BauwerksID, Bauwerkstyp, Name}
- Verbund: Bauwerke × Gemeinden
- Spalten: BauwerksID, Bauwerkstyp, Name
- *Umbenennung*: Name → Gemeindename

BauwerksID	Bauwerkstyp	Gemeindename
5000	Tankstelle	Graal-Müritz
5001	Hotel	Dummerstorf
5002	Kirche	Graal-Müritz

UMBENENNUNG - IN SQL

Mit **AS** Keyword:

```
SELECT tbl1.attr1 AS Identifikationsnummer  
FROM Table1 AS tbl1
```

Das Beispiel von vorhin:

```
SELECT BauwerksID, Bauwerkstyp, Name AS Gemeindename  
FROM Bauwerke  
NATURAL JOIN Gemeinden
```

Sowohl im **SELECT** als auch im **FROM** möglich durch die Nutzung des **AS** Keywords

AGGREGATFUNKTIONEN

- Fassen Spaltenwerte nach einer bestimmten Vorgehensweise zusammen
- Bilden dabei standardmäßig eine 1×1 Tabelle als Ergebnis
- Soll mehr als nur ein Wert erscheinen, muss nach einer bestimmten Bedingung aggregiert werden
- Gibt es eine Bedingung, repräsentieren die Zeilen verschiedene Fälle
- *Häufige Aggregatfunktionen:*
Summenbildungen oder Zählungen

GemeindeID	Name	Einwohner
1	Dummerstorf	7.329
2	Graal-Müritz	4.278
3	Sanitz	5.831

Summe: 17.438

AGGREGATFUNKTIONEN - IN SQL

COUNT-Funktion zählt die Anzahl der Zeilen:

```
SELECT COUNT (GemeindeID) FROM  
Gemeinden
```

```
*COUNT: 3*
```

SUM-Funktion summiert eine Spalte numerisch auf:

```
SELECT SUM (Einwohner) FROM Gemeinden
```

```
*SUM: 17.438*
```

BEDINGTE AGGREGATION - IN SQL

- Aggregiert man nach einem bestimmten Attribut, dann werden nur jeweils jene Tupel zusammengefasst (summiert, gezählt, gemittelt,...) die den gleichen Attributwert haben
- Hierfür ist in SQL der **GROUP BY** Befehl zuständig

Beispiel: Eine Tabelle Mitarbeiter enthält Honorarzahlen an Mitarbeiter, pro Zahlung ist dabei eine Zeile angelegt. Man will nun das gemittelte Honorar für jeden Mitarbeiter ausgeben lassen.
Gegeben die Relation Mitarbeiter{MitarbeiterID, Name, Honorar}

```
SELECT MitarbeiterID, Name, AVG(Honorar)
FROM Mitarbeiter
GROUP BY MitarbeiterID
```

WEITERE OPERATIONEN: SORTIERUNG & ANZAHL

Sortierung:

```
SELECT * FROM Gemeinden  
ORDER BY Einwohner DESC
```

- `ORDER BY` mit `ASC` (aufsteigend) oder `DESC` (absteigend)

Begrenzung der Ergebnismenge:

```
SELECT * FROM Gemeinden  
ORDER BY Einwohner DESC  
LIMIT 2
```

- `LIMIT` schneidet die Ergebnistabelle nach einer gewissen Anzahl an Elementen ab
- Hier: Die zwei Gemeinden mit der höchsten Einwohnerzahl

KOMBINATION VON BEDINGUNGEN

Bedingungen können wie in Python mit **AND** und **OR** kombiniert werden:

```
SELECT * FROM Bauwerke
NATURAL JOIN Gemeinden
WHERE Einwohner > 5000 AND
Bauwerkstyp = "Hotel"
```

Ergebnis:

BauwerksID	Bauwerkstyp	GemeindeID	Name	Einw
5001	Hotel	1	Dummerstorf	7.329

AUSBLICK: SQL - MEHR ALS EINE ANFRAGESPRACHE

Bisher: Nur ein einzelner Aspekt von SQL (`SELECT` ...)

SQL bietet weit mehr Funktionalitäten:

- *DML (Data Manipulation Language):*
 - Daten aus Tabellen anfragen und modifizieren (`SELECT, UPDATE, DELETE` ...)
- *DDL (Data Definition Language):*
 - Definieren und Verwalten von Schemas und Tabellen (`CREATE, ALTER, DROP` ...)
- *DCL (Data Control Language):*
 - Nutzerzugriffsrechte (`GRANT, REVOKE` ...)
 - Kontrollieren von Transaktionen (`COMMIT, ROLLBACK` ...)

fragen?

