Formatierung und Einrückung

SQL-Abfragen können schnell komplex und unübersichtlich werden, insbesondere bei großen Projekten oder umfangreichen Datenabfragen. Eine gute Organisation der Abfragen sorgt für Lesbarkeit, Wartbarkeit und eine klare Struktur. Hier sind Best Practices, um SQL-Abfragen effektiv zu organisieren:

---

1. Formatierung und Einrückung

Verwende Einrückungen, um die Abfrage hierarchisch zu strukturieren. Jede SQL-Klausel sollte in einer neuen Zeile stehen.

Beispiel:

SQL

```sql
SELECT name, stadt, land
FROM kunden
WHERE land = 'Deutschland'
ORDER BY name ASC;

```

---

2. Logische Reihenfolge einhalten

Die Abfrage sollte in der Reihenfolge der SQL-Syntax geschrieben werden:

1. `SELECT` (Was auswählen?)
2. `FROM` (Woher kommen die Daten?)
3. `WHERE` (Welche Bedingungen gelten?)
4. `GROUP BY` (Wie gruppieren?)
5. `HAVING` (Welche Gruppenbedingungen?)
6. `ORDER BY` (Wie sortieren?)

---

3. Aliase sinnvoll verwenden

Verwende Kürzel (Aliase) für Tabellen und Spalten, insbesondere bei langen oder

mehrfach verwendeten Namen.

Beispiel:

```sql
SELECT k.name, k.stadt, l.code
FROM kunden AS k
JOIN laender AS l
  ON k.land_id = l.id;
```

- `AS k` und `AS l` erleichtern das Lesen und Wiederverwenden.

---

4. Kommentare einfügen

Ergänze erklärende Kommentare, um die Absicht der Abfrage klarzumachen.

Beispiel:

```sql
-- Liste aller Kunden aus Deutschland, alphabetisch sortiert
SELECT name, stadt
FROM kunden
WHERE land = 'Deutschland'
ORDER BY name ASC;
```

---

5. Logik durch Klammern verdeutlichen

Verwende Klammern in komplexen Bedingungen, um die Priorität klarzustellen.

Beispiel:

```sql
SELECT name
FROM kunden
WHERE (stadt = 'Berlin' AND land = 'Deutschland')
   OR (stadt = 'Zürich' AND land = 'Schweiz');
```

---

6. Teilabfragen und CTEs (Common Table Expressions)

Teile lange Abfragen in verständliche Blöcke auf, indem du Teilabfragen oder CTEs verwendest.

Beispiel mit einer Teilabfrage:

SQL

```sql
SELECT name
FROM kunden
WHERE id IN (
  SELECT kunden_id
  FROM bestellungen
  WHERE bestellwert > 1000
);

```

Beispiel mit einer CTE:

SQL

```sql
WITH HighValueCustomers AS (
  SELECT kunden_id
  FROM bestellungen
  WHERE bestellwert > 1000
)
SELECT name
FROM kunden
WHERE id IN (SELECT kunden_id FROM
HighValueCustomers);

```

---

7. Vermeidung von Magischen Werten

Vermeide „magische Werte“ (fest kodierte Werte) und verwende stattdessen Variablen oder Parameter, wenn möglich.

Beispiel:

Anstatt:

SQL

```sql
SELECT name FROM kunden WHERE land = 'Deutschland';

```

Lieber:

```sql
SELECT name FROM kunden WHERE land = @land;
```

---

8. Normen und Namenskonventionen

- Verwende einheitliche Groß-/Kleinschreibung (z. B. SQL-Schlüsselwörter in Großbuchstaben).
- Benenne Tabellen und Spalten konsistent (z. B. Snake Case: `kunde_id`, `bestell_wert`).

---

9. Tabellen- und Spaltenreferenzen explizit angeben

Bei Joins sollte die Tabelle immer angegeben werden, um Mehrdeutigkeiten zu vermeiden.

Beispiel:

```sql
SELECT k.name, b.bestellwert
FROM kunden AS k
JOIN bestellungen AS b
  ON k.id = b.kunden_id;
```

---

10. Verwendung von Tools für Formatierung und Versionierung

- Nutze **SQL-Formatter** wie **SQL Beautifier**, um Abfragen automatisch zu formatieren.
- Sichere Abfragen in Versionskontrollsystemen wie **Git**, um Änderungen nachzuvollziehen.

Beispiel: Gut strukturierte Abfrage

SQL

```sql
-- Umsatzanalyse: Kunden und deren Bestellungen über 1000€
WITH HighValueOrders AS (
  SELECT kunden_id, SUM(bestellwert) AS gesamtwert
  FROM bestellungen
  WHERE bestellwert > 1000
  GROUP BY kunden_id
)
SELECT k.name, o.gesamtwert
FROM kunden AS k
JOIN HighValueOrders AS o
  ON k.id = o.kunden_id
ORDER BY o.gesamtwert DESC;

```

Durch die Anwendung dieser Techniken werden Abfragen klarer, leichter zu debuggen und wartungsfreundlicher.