

Der SQL - Knigge für beliebte & berüchtigte Fehler

Ein SQL - Kommando hat folgenden Aufbau:

```
SELECT [DISTINCT | ALL] attribut [, ...] FROM tabelle [t] [, ...]  
[WHERE bedingung]  
[GROUP BY attribut [, ...] [HAVING bedingung]]  
[ORDER BY attribut [ASC | DESC] [, ...]]
```

Hinweis 1: Um die Auswahlbedingung des SELECT-Kommandos zu formulieren, kann es notwendig sein, eine Unterabfrage in die WHERE-Klausel zu integrieren. In der Regel kann auch eine (Equi-) Join-Verbindung eines SELECT-Kommandos durch eine Unterabfrage ersetzt werden:

Beispiel:

Welche Spieler haben eine Strafe erhalten?

Join:

```
SELECT DISTINCT (sp.spielernr), name FROM spieler sp, strafen st  
WHERE sp.spielernr= st.spielernr
```

Die Attribute einer Abfrage müssen eindeutig einer Relation zugeordnet werden können. Aus diesem Grund müssen die Attribute spielernr in der Join-Abfrage durch den Relationsnamen bzw. den Korrelationsvariablen der Relation ergänzt werden.

Subselect:

```
SELECT spielernr, name FROM spieler  
WHERE spielernr IN (SELECT spielernr FROM strafen)
```

Hinweis 2: In einer korrelierten Unterabfrage greift eine Bedingung der Unterabfrage auf Attribute einer übergeordneten FROM – Klausel zu.

```
SELECT spielernr, name FROM spieler  
WHERE spielernr IN (SELECT spielernr FROM strafen st  
WHERE spieler.spielernr < 100)
```

Hinweis 3: Die GROUP BY - Klausel des SELECT-Kommandos gruppiert die Auswahlliste anhand eines Attributes bzw. einer Attributgruppe.

```
SELECT auswahlliste FROM tabelle_1 GROUP BY attribut_1
```

Zu beachten ist, daß das attribut_1 in der Auswahlliste des SELECTs enthalten sein kann und alle anderen Attribute der Auswahlliste ausschließlich im Zusammenhang mit den Aggregatfunktionen MIN, MAX, SUM, AVG oder COUNT verwendet werden müssen, wie z. B.:

Annahme folgender Ausschnitt aus der Relation Spieler:

Spielernr	Geschlecht
1	w
2	m
3	w
4	m
5	w

Beispiel:

Wieviel weibliche und männliche Spieler gibt es?

```
SELECT geschlecht, count(*)
FROM spieler GROUP BY geschlecht
```

Antworttabelle:

geschlecht	Count(*)
w	3
m	2

Falsches Beispiel: Warum ist dieses SELECT nicht ausführbar?

```
SELECT geschlecht, spielernr
FROM spieler GROUP BY geschlecht
```

Formale Antwort: Weil die Antwortrelation nicht in 1. Normalform vorliegt, d. h. in der Spalte „spielernr“ und Zeile „männlich“ werden alle Spielernummern der männlichen Spieler aufgeführt. Es kann allerdings nur ein Wert enthalten bzw. angezeigt werden.

Antworttabelle:

geschlecht	spielernr
w	1, 3, 5
m	2, 4

Korrektur:

1. Alternative: Streiche „spielernr“ aus der Auswahlliste.

```
SELECT geschlecht
FROM spieler GROUP BY geschlecht
```

Antworttabelle:

geschlecht
w
m

2. Alternative: Benutze eine Aggregatfunktion für „spielernr“

```
SELECT geschlecht, MIN(spielernr)
FROM spieler GROUP BY geschlecht
```

Antworttabelle:

geschlecht	MIN(spielernr)
w	1
m	2

3. Alternative: Aufnahme in die GROUP BY – Klausel:

```
SELECT geschlecht, spielernr
FROM spieler GROUP BY geschlecht, spielernr
```

Antworttabelle:

geschlecht	Spielernr
m	2
w	3
m	4
w	5
w	1

Frage: Ist die Tabelle korrekt? Ja, denn eine Sortierung wird **ausschließlich** durch eine ORDER BY – Klausel durchgeführt, alle anderen Relationen sind unsortierte Mengen von Tupeln, d. h. die Reihenfolge der Tupel ist willkürlich gewählt!

Fazit: Die Reihenfolge der Attribute in der GROUP BY –Klausel ist für die Sortierung der Antworttabelle irrelevant. Die Sortierung wird durch das DBMS bestimmt.

Hinweis 4: Die GROUP BY - Klausel kann durch eine HAVING-Bedingung ergänzt werden. Diese sollte sich auf die temporäre Antworttabelle **nach** der Gruppierung beziehen.

Beispiel:

Liste alle Orte auf, in denen mindestens 3 Spieler wohnen.

```
SELECT ort FROM spieler GROUP BY ort HAVING COUNT(*) >= 3
```

Falsches Beispiel:

Wieviel weibliche und männliche Spieler gibt es in Düsseldorf?

```
SELECT geschlecht, count(*)
FROM spieler GROUP BY geschlecht HAVING ort = 'Düsseldorf'
```

Die Bedingung „in Düsseldorf“ bezieht sich nicht auf die Gruppierung, sondern auf die Tupelauswahl, d.h., sie gehört in die WHERE-Klausel:

```
SELECT geschlecht, count(*)
FROM spieler WHERE ort = 'Düsseldorf' GROUP BY geschlecht
```

Hinweis 5: Verwendung von Unterabfragen: Unterabfragen können innerhalb der WHERE- oder der HAVING-Klausel verwendet werden. Dabei kann ein Attribut oder Ausdruck über einen der bekannten Vergleichsoperatoren (=, <, >, >=, <=) mit einer Unterabfrage verknüpft werden. Es ist zu beachten, daß die Unterabfrage nur einen (skalaren) Wert zurückliefern darf, da ansonsten die 1. Normalform verletzt wird (siehe auch Hinweis 3). Allerdings kann der Vergleichsoperator mit Hilfe der Prädikate ALL oder ANY (bzw. SOME) modifiziert werden, so daß die Unterabfrage auch mehrere Tupel zurückliefern darf.

Beispiel:

Welche Spieler von Team 1 haben mehr Spiele gewonnen als der Teamdurchschnitt ?

```
SELECT spielernr FROM wettkämpfe WHERE teamnr = 1
AND gewonnen > ( SELECT AVG(gewonnen) FROM wettkämpfe
WHERE teamnr = 1)
```

Bei dieser Unterabfrage ist durch die Verwendung einer Aggregatfunktion sichergestellt, daß sie nur genau einen Wert zurückliefert.

Ohne die Verwendung von Vergleichsoperatoren kann eine Unterabfrage auch mit den Prädikaten IN oder EXISTS eingesetzt werden. Mit Hilfe des IN-Prädikats kann ein Attribut oder Ausdruck mit einer Unterabfrage verknüpft werden, die mehr als einen Datensatz zurückliefert. Das EXISTS-Prädikat wird ohne ein Attribut oder Ausdruck verwendet. Wenn die Antworttabelle der nachfolgenden Unterabfrage Datensätze zurückliefert, war der Test, ob Datensätze *existieren*, erfolgreich. Welche und wieviel wird nicht weiter beachtet.

Beispiele:

tabA:

spA
A
B
C
D

tabB:

spB
B
C

1. `SELECT * FROM tabA WHERE spA >= ANY (SELECT spB from tabB)`

Die Antworttabelle enthält: B, C, D

FALSCH:

```
SELECT * FROM tabA WHERE spA >= (SELECT spB from tabB)
```

Die Unterabfrage liefert mehr als einen Wert zurück, wodurch ein Fehler bei der Zuordnung zu einem Attributwert produziert wird.

2. `SELECT * FROM tabA WHERE spA >= ALL (SELECT spB from tabB)`

Die Antworttabelle enthält: C, D

3. `SELECT * FROM tabA WHERE spA IN (SELECT spB FROM tabB)`

Die Antworttabelle enthält: B, C

4. `SELECT * FROM tabA WHERE EXISTS (SELECT * FROM tabB WHERE tabA.spA = tabB.spB)`

Die Antworttabelle enthält: B, C

Hinweis 6: Es ist nicht immer wünschenswert, daß in jedem Datensatz jedes Attribut einen Wert besitzt. Bereits beim Anlegen einer Tabelle kann durch NULL oder NOT NULL definiert werden, für welche Attribute bei der Dateneingabe Werte zwingend vorhanden sein müssen (z.B. beim Primärschlüssel) und für welche nicht (z.B. eine private Faxnummer). Entsprechend kann bei der Abfrage auf Tabellendaten durch die Verwendung des NULL-Prädikats ein Test auf Null-Werte vorgenommen werden. Entscheidend ist, daß dabei nicht auf den numerischen Wert Null oder Zeichenketten der Länge Null oder Blank abgestellt wird, sondern NULL einen nicht definierten Wert repräsentiert. NULL ist insofern ein spezieller Wert, der das Nichtvorhandensein von Information dokumentiert.

Beispiel:

Welche Spieler haben (k)einen Titel?

```
SELECT spielernr FROM spieler WHERE titel IS NOT NULL
(SELECT spielernr FROM spieler WHERE titel IS NULL)
```

FALSCH:

```
SELECT spielernr FROM spieler WHERE titel <> NULL
```

Bei der Verwendung von Spaltenfunktionen ist zu beachten, daß Nullwerte ignoriert werden. Die Ausnahme ist COUNT(), bei der die Anzahl der Tupel der Antworttabelle gezählt werden, wobei einzelne Attributwerte eines Tupels auch Nullwerte enthalten können.

Hinweis 7: Die Spalten- bzw. Aggregatfunktion COUNT sollte in der allgemeinen Schreibweise COUNT(*) verwendet werden, dies impliziert die Zählung der Tupel der Antworttabelle bzw. der Gruppierung. **Ausnahme:** Die COUNT-Funktion wird mit DISTINCT verwendet:

Beispiel:

Wieviel unterschiedliche Orte gibt es?

```
SELECT COUNT(DISTINCT ort) FROM spieler
```

Leider unterstützt Access dieses (noch) nicht in der aktuellen Version. Es bleibt die Hoffnung.