DATENBANKEN & SQL

– Übersicht gewinnen und behalten / Komplexere Abfragen –

Kurs am RRZK der Universität zu Köln

Rüdiger Voigt, M.A.

22.07. – 26.07.2019

Übersicht

Übersicht gewinnen

TOP

Warnung

Oft ist ein **SELECT** TOP() Statement eleganter als der Einsatz von Limit. TOP(5) gibt etwa die ersten 5 Zeilen aus und verhält sich ähnlich wie LIMIT. TOP(3)PERCENT gibt die ersten 3 Prozent zurück und erspart die Berechnung. Die Option WITH TIES zeigt mehr Datensätze als angefordert an, wenn weitere Einträge identische Werte hinsichtlich des ORDER BY Kriteriums aufweisen wie der eigentlich Letzte.

ABER TOP() ist nicht Teil des ANSI Standards und wird weder von MySQL noch von MariaDB unterstützt. Im MS SQLServer steht der Befehl zur Verfügung.

ORDER BY RAND() - zufällig Zeilen auswählen

Manchmal ist es sinnvoll eine Stichprobe zu ziehen. Zum Beispiel um einen ersten Eindruck von den Daten zu gewinnen, oder um ein Programm mit einem Teildatensatz zu testen.
Eine Stichprobe erhält man leicht indem man die Ausgabe zufällig sortiert und die Zahl der Datensätze einschränkt, also:

SELECT ∗ **FROM table ORDER BY** RAND() **LIMIT** 10;

ORDER BY RAND() - Probleme

HINWEIS

Die Stichprobe mittels ORDER BY RAND() hat zwei Nachteile:

1. "[...] this technique should never be used on a big table. In fact, MariaDB will read all rows in the table, will generate a random value for each of them, will order them, and finally will apply the LIMIT clause." (MariaDB Docs)
2. Diese Methode ist rein zufällig. Sie gewährleistet nicht, dass bestimmte Fallgruppen im richtigen Verhältnis enthalten sind. (Das könnte man mit mehreren Teilabfragen und UNION gewährleisten.)

Wiederholung: Einträge zählen

Der SELECT Befehl erweiter um COUNT erlaubt es das Resultset durchzuzählen.

Die Gesamtzahl der Zeilen erhalten Sie mit:
SELECT COUNT(∗) **FROM** ‘exampleTable‘

Wenn Sie eine WHERE-Bedingung benutzen können Sie Teilgruppen auszählen:
SELECT COUNT(∗) **FROM** ‘exampleTable‘ **WHERE** ‘type‘ = 2

DISTINCT

Oft ist es sinnvoll zu wissen, welche verschiedenen Werte in einer Spalte codiert wurden. Mittels
SELECT DISTINCT(‘exampleColumn‘) **FROM** ‘exampleTable‘;
erhalten Sie eine Auflistung aller in der Spalte vorkommenden Werte.

Wenn Sie die Zahl der verschiedenen Werte innerhalb der Spalte interessiert, dann hilft Ihnen
SELECT COUNT(**DISTINCT**(‘exampleColumn‘))
FROM ‘exampleTable‘;
weiter.

GROUP BY

JOIN

Wofür JOIN?

Joins sind zentral für relationale Datenbanken und sind Vorraussetzung für Deduplizierung. Mittels eines JOIN verbinden Sie eine Tabelle mit Spalten *aus einer oder aus mehreren anderen Tabellen*. Dafür nutzen Sie ein eindeutiges Identifizierungsmerkmal.

Ein JOIN ist in der Regel deutlich performanter als ein Sub-SELECT und diesem Konstrukt entsprechend vorzuziehen!

Verschiedene Arten von JOIN

- LEFT JOIN
- RIGHT JOIN
- INNER JOIN
- FULL JOIN

Der LEFT JOIN wird sehr häufig genutzt. Deshalb behandeln wir diesen hier exemplarisch und es ist Ihnen überlassen sich die weiteren Varianten anzueignen.

LEFT JOIN: Beispiel

```
SELECT
FROM thisVeryTable t
WHERE someCondition IS NOT NULL
LEFT JOIN thatOtherTable o
```

Exkurs / FYI: JOIN in R

UNION

Was macht der UNION Befehl?

UNION verhält sich ähnlich wie JOIN. Beide Befehle verbinden Daten aus verschiedenen Tabellen zu einem einzigen Resultset.

Der entscheidene Unterschied:
Bei einem JOIN wird das Resultset um weitere Columns vergrößert, während bei UNION neue Rows hinzu kommen.

Vorraussetzung für die Anwendung von UNION

Um die von mehreren Abfragen zurückgegebenen Zeilen miteinander zu verbinden, muss die Zahl der Spalten und die Reihenfolge der Datentypen zwingend übereinstimmen!

Wenn die Reihenfolge der Felder in den Tabellen unterschiedlich ist, sortiert man in der SELECT-Query die Reihenfolge der Spalten um. Sollte ein Feld gar nicht vorkommen hilft das Konstrukt NULL AS colName um eine mit NULL gefüllte Spalte zu simulieren.

Beispiel für eine UNION

SELECT a, b, c **FROM** someT **WHERE** someC = 1
UNION
SELECT NULL AS a, d, e **FROM** otherT **WHERE** otherC = 0;

Beachten Sie: nur am Ende finden Sie ein Semikolon!

UNION versus UNION ALL

Der Befehl UNION prüft, ob im Ergebnis Zeilen doppelt vorkommen und entfernt dann alle bis auf eine.

Wenn das nicht gewünscht ist, oder Sie sicher sind, dass sich keine Dubletten ergeben, können Sie UNION ALL verwenden. Dieser Befehl führt diese Prüfung nicht durch und kann dadurch erheblich schneller sein.

Methoden um UNION zu vermeiden

Manchmal sind die Ergebnisse von UNION nicht optimal. Zm Beispiel erlauben Ihnen neuere Versionen des Microsoft SQL Server innerhalb eines VIEW nicht das Ergebnis einer Query zu sortieren, die mit UNION erstellt wurde.
Wenn Sie Teilabfragen auf die selbe Tabelle miteinander verbinden möchten, erweitern Sie einfach die WHERE Bedingung indem Sie zum Beispiel mit OR ein weiteres Identifizierungsmerkmal hinzufügen.

WHERE . . . IN()

Eines der hilfreichsten Konstrukte ist WHERE . . . IN(). Im Prinzip eine ganz normale WHERE Bedingung, außer das innerhalb der runden Klammern hinter IN() von SQL eine Liste erwartet wird.

Einerseits ist das eine andere Schreibweise für
WHERE cond1 **OR** cond2 **OR** cond3
aber vor allem kann hier ein Sub-SELECT verwenden können, welches eine Liste zurückgibt. Also zum Beispiel:

```
SELECT name FROM students s
WHERE s.id IN (SELECT studentID FROM ueberfaelligeBuecher);
```

Subqueries

Definition

SQL erlaubt es Ihnen Befehle zu verschachteln. Sie können innerhalb eines Befehls weitere Befehle absetzen um Werte für eine Column zu ermitteln. In der Regel verschachteln Sie SELECT-Queries, so dass meist von Sub-SELECTs gesprochen wird.
Subqueries sind dann sinnvoll, wenn Sie keinen JOIN verwenden können, zum Beispiel weil Sie eine Aggregatfunktion auf eine Tabelle anwenden wollen um das Ergebnis der Subquery zu berechnen.

Übersicht behalten: Dokumentation

Sobald komplexer Code mehrfach ausgeführt, oder auch nur aufgehoben werden soll, lohnt es sich zu dokumentieren.

Für wen dokumentieren Sie Ihren SQL-Code?

Häufige Antworten:

1. für mich selbst in drei Wochen
2. für Dritte, die den Code verstehen müssen
3. für die Person, welche den Code in fünf Jahren aktualisieren muss

lesbarer Code I

Guter Code spart erheblich Aufwand bei der Dokumentation!

3 Tricks:

1. **Sprechende Bezeichnungen**
   table01 und var001 schweigen. Wenn eine Konvention festgelegt wurde, erkennt man t_events als Tabelle, die wahrscheinlich Ereignisse auflistet. p_int_eventSeverity verrät, dass es ein Paramter vom Typ int ist und wahrscheinlich in irgendeiner Form den Schweregrad eines Ereignis misst.
2. **Zeilenumbrüche**
   In SQL geht der Befehl immer bis zum Semikolon. Es schadet nicht logische Einheiten mit einem Zeilenumbruch zu beenden. Der Code wird aber besser lesbar.

lesbarer Code II

3. **Benennung mittels AS**
   Abhängig vom Kontext kann man Werte und Tabellen mittels AS bennen um ihren Zweck zu verdeutlichen.

Kommentarzeichen

MariaDB erlaubt drei Methoden Kommentare mit Code zu mischen:

1. Nach einem # wird *Alles bis zum Zeilenende* als Kommentar verstanden.
2. Alles nach - - *und einem Leerzeichen* wird ebenfalls bis zum Ende der Zeile als Kommentar interpretiert.
3. Ein Kommentar über mehrere Zeilen beginnt mit /* und endet mit */

Der Microsoft SQL Server kennt die beiden letzten Methoden, aber nicht die Erste.

Kommentare

- Wiederholen Sie im Kommentar nicht was man im Code sieht – außer bei komplexen Code.
- Beantworten Sie eher die Frage “Warum?”.

Im Vergleich

Unverständlich:

SELECT id, var001, var002 **FROM** table1 **WHERE** p > @t;

Äquivalent und leicht zu verstehen:

```
SELECT id,
p_int_eventType, --ID entspricht Nummer im Codebook
p_txt_desc AS eventDescription,
FROM t_eventLog
WHERE eventSeverity > @userDefinedTreshold;
```

@varname ist die Schreibweise von Variablen in T-SQL

Externe Code-Dokumentation

Erfahrungswerte:

- Consultants ändern gelegentlich Code – ohne zu fragen, oder zu dokumentieren.
- Das ein oder andere DBMS – wie etwa Microsoft SQL Server – schreibt Code von Funktionen und Views direkt nach dem Abspeichern um.

Versionsverwaltung

Praktikable Lösung: Code in einer externen Datei mit Datum im Dateinamen speichern.

Bessere Lösung: Verwenden Sie ein Versionskontrollsystem wie git (mit zentralem Repository) oder subversion.

Codebook

Wenn man Daten wissenschaftlich verwendet, müssen Sie als *Replication Data* verfügbar sein. Hier beschreibt man nicht unbedingt die Struktur der Datenbank selber, denn unter Umständen exportiert man einfach eine Ansicht.

Beschreiben Sie welche Werte in welcher Spalte gültig sind und was sie bedeuten.

Weiter mit dem Foliensatz “Datenbanken erstellen”.

Den kompletten Foliensatz finden Sie in aktueller Version unter:
`https://www.ruediger-voigt.eu/kurs-datenbanken-und-sql.html`