Vortrag 5 Weiterführende Themen

Datenbanken

Lukas Wais

Codersbay

Version: 19. April 2023

Inhaltsverzeichnis

Was sind Subqueries?

Es sind verschachtelte Abfragen, welche komplexere Queries ermöglichen. Im folgenden Beispiel werden wir die *countries* Tabelle verwenden.

Subselects

Beispiel

Angenommen wir möchten alle Länder, die eine Fläche größer als 5.000.000 km^2 besitzten abfragen.

Subselects

Beispiel

Angenommen wir möchten alle Länder, die eine Fläche größer als 5.000.000 km^2 besitzten abfragen. Als Erstes benötigen, wir den Primärschlüssel aller, Länder auf welche diese Eigenschaft zutrifft.

```
select country_id
from countries
where area > 5000000;
```

Als Ergebnis bekommen wir folgende IDs:

$$12, 15, 31, 38, 42, 182, 224$$

Subselects

Beispiel

Um mehr Informationen von country zu erhalten, müssen wir jetzt eine neue Abfrage mit diesen IDs schreiben.

```
select
    name,
    area
from countries
where country_id in (12,15,31,38,42,182,224)
order by
    area,
    name;
```

Nachteil

- Sollten sich die Flächen, ändern, so muss auch die zweite Abfrage geändert werden.
- Man benötigt zwei Queries und die zweite muss sogar manuell basierend auf der Ersten verändert werden.

Lösung mit Subquerie

Wir verwenden jetzt ineinander verschachtelte Abfragen.

```
select
    name,
    area
from
    countries
where country_id in (
    select
        country_id
    from
       countries
    where
        area > 5000000
)
order by
    area,
    name;
```

Outerquery vs. Subquery

```
select country_id
from countries
where area > 5000000
```

Ist die Subquery und der äußere Teil der gesamten Abfrage das Outerquery.

Scalar Queries

Es gibt auch noch skalare Abfragen, diesen liefern nur einen Wert zurück. Um herauszufinden, welches Land das größte ist, kann mit `max(area)` ein skalarer Wert abgefragt werden.

```
select *
from countries
where area = (
    select max(area)
    from countries
);
```

Row Subqueries

Diese geben eine ganze Reihe zurück, im folgenden Beispiel verwenden wir zusätzlich die *country_id* Tabelle. Die Abfrage auf der nächsten Folie gibt countries aus, deren Bevölkerung und BIP im Durchschnitt größer ist als die durchschnittliche Bevölkerung und der BIP aller Länder im Jahr 2018

```sql
select
    name
from
    country_stats
inner join countries
    using (country_id)
where
    year = 2018 and
    (population, gdp) > (
        select
            avg(population),
            avg(gdp)
        from country_stats
        where year = 2018)
order by
    name;
```

Views

Eine View ist ein Subset von einer oder mehrerer Tabellen. Man kann sie sich als virtuelle Tabelle vorstellen. Sie werden als named queries im DBMS gespeichert und können verwendet werden, um häufig verwendete, komplexe Abfragen zu speichern. Im folgenden Beispiel kommen die *countries*, *regions* und *continents* Tabelle zum Einsatz.

Diese Abfrage möchten wir wiederverwenden

```
select
    c.name country,
    r.name region,
    t.name continent,
    area
from countries c
inner join regions r
    using (region_id)
inner join continents t
    using (continent_id)
order by country;
```

Lösung

```
create view country_details
as
select
    c.name country,
    r.name region,
    t.name continent,
    area
from countries c
inner join regions r
    using (region_id)
inner join continents t
    using (continent_id)
order by country;
```

Verwendung von Views

Neu hinzugekommen ist nur der Teil:

```
create view country_details as
```

Um die View zu verwenden genügt:

```
select *
from country_details;
```

Vorteile von Views

- **Einfachheit:** Kapselung von komplizierten Abfragen.
- **Konsistenz:** Businesslogik und Formeln werden verbunden.
- **Sicherheit:** Feinere Berchtigungen sind möglich; nicht alle Benutzer können sensible Daten in der Tabelle einsehen.

Definition Trigger

Ein Trigger wird ausgelöst bei veränderungen in einer Tabelle. Ähnlich Eventlistener in Programmiersprachen.
Folgende Arten unterstützt MariaDB:

- before insert
- after insert
- before update
- after update
- before delete
- after delete

Erstellung eines Triggers

```
create trigger trigger_name
{before | after} {insert | update | delete }
on table_name for each row
trigger_body;
```

Logging für country_reports

```
create trigger before_country_reports_update
    before update on country_reports
    for each row
    insert into population_logs(
        country_id,
        year,
        old_population,
        new_population
    )
    values(
        old.country_id,
        old.year,
        old.population,
        new.population
    );
```

population_logs Tabelle

```
create table population_logs(
    log_id int auto_increment,
    country_id int not null,
    year int not null,
    old_population int not null,
    new_population int not null,
    updated_at timestamp default current_timestamp,
    primary key(log_id)
);
```

Was sind Stored Routines?

Es sind SQL Statements, die im DBMS gespeichert und dann aufgerufen werden. Das reduziert die Anzahl an queries auf der Softwareseite und bietet auch Performance Vorteile, weil die Queries direkt in der Datenbank ausgeführt werden (sortieren, average, ...). Sollten mehrere Applikationen auf dieselbe DB zugreifen und die gleichen Abfragen durchführen, können diese wiederverwendet werden.

Definition Stored Procedures

Es sind Stored Routines, die Eingabe, Ausgabe und Einausgabeparameter haben können. Außerdem werden sie mit CALL aufgerufen.

Syntax Stored Procedure

```
CREATE
    [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PROCEDURE [IF NOT EXISTS] sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statement
```

Aufruf von Stored Procedures

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

Aufruf Stored Procedure mit OUT Parameter

```
CALL get_max_salary(@S);
SELECT @S;
```

SP Beispiel mit Inputs; verwendet von Banken

```
CREATE PROCEDURE
  Withdraw                                /* Routine name */
  (parameter_amount DECIMAL(6,2),         /* Parameter list */
  parameter_teller_id INTEGER,
  parameter_customer_id INTEGER)
  MODIFIES SQL DATA                       /* Data access clause */
  BEGIN                             /* Routine body */
    UPDATE Customers
        SET balance = balance - parameter_amount
        WHERE customer_id = parameter_customer_id;
    UPDATE Tellers
        SET cash_on_hand = cash_on_hand + parameter_amount
        WHERE teller_id = parameter_teller_id;
    INSERT INTO Transactions VALUES (
        parameter_customer_id,
        parameter_teller_id,
        parameter_amount);
  END;
```

Definition Stored Functions

Sie sind ähnlich der Stored Procedures und können Parameter haben. Sie besitzen einen Rückgabewert und werden aber nicht mit CALL aufgerufen.
Der Aufruf vom unteren Beispiel erfolgt mit `SELECT FortyTwo();`.

```
DELIMITER //

CREATE FUNCTION FortyTwo() RETURNS TINYINT DETERMINISTIC
BEGIN
 DECLARE x TINYINT;
 SET x = 42;
 RETURN x;
END

//

DELIMITER ;
```

Mehr zu dem DELIMITER Schlüsselwort.

Stored Routines können einiges mehr

https://mariadb.com/kb/en/stored-routines/

DBMS sind sehr Umfangreich

Darum immer gut die Dokumentation studieren und vor Implementierungen, prüfen, ob es nicht eine hilfreiche Funktion dafür gibt.

- https://www.mariadbtutorial.com/
- https://mariadb.com/kb/en/