Übung S2: Advanced SQL

Gruppe 8: Lukas Arnold, Patrick Bucher, Christopher James Christensen, Jonas Kaiser, Melvin Werthmüller

1. Metadaten

Frage 1

Beschreiben Sie einen Use Case, in dem Metadaten für Sie notwendig sind, und wo Sie dann das Data Dictionary verwenden können, um Beschreibungen von Tabellen und Spalten zu durchsuchen.

- Wenn in eine Applikation mit generischen Operationen wie Einfügen, Bearbeiten, Löschen entwickeln möchte, kann ich den Code für den Datenbankzugriff oder sogar User Interfaces anhand der Meta-Informationen generieren lassen.
- Wenn ich verschiedene Versionen einer Datenbank miteinander vergleichen möchte (welche Tabellen/Spalten sind hinzugekommen/weggefallen).
- Wenn ich mittels `psql` oder einem vergleichbaren CLI auf eine mir unbekannte Datenbank zugreife und zuerst deren Struktur verstehen muss.

Frage 2

Welche Tabelle enthält die Spalte Fachgebiet?

```
select table_name
from information_schema.columns
where column_name = 'fachgebiet';
```

```
assistenten
```

Frage 3

Welche Spalten in der Uni-Datenbank haben den Datentyp Integer?

```
select concat(table_name, '.', column_name) as Spalte
from information_schema.columns
where
    table_catalog = 'unidb'
    and table_schema = 'public'
    and data_type like '%int%'
order by Spalte asc
```

```
assistenten.boss
assistenten.persnr
hoeren.matrnr
hoeren.vorlnr
professoren.persnr
professoren.raum
pruefen.matrnr
pruefen.persnr
pruefen.vorlnr
studenten.matrnr
studenten.semester
voraussetzen.nachfolger
voraussetzen.vorgänger
vorlesungen.gelesenvon
vorlesungen.sws
vorlesungen.vorlnr
```

Frage 4

Welche Tabelle enthält das Information Schema?

```
select table_name
from information_schema.tables
where table_schema = 'information_schema'
order by table_name asc;
```

```
administrable_role_authorizations
applicable_roles
attributes
character_sets
check_constraint_routine_usage
check_constraints
collation_character_set_applicability
collations
column_domain_usage
column_options
column_privileges
columns
column_udt_usage
constraint_column_usage
constraint_table_usage
data_type_privileges
domain_constraints
domains
```

```
domain_udt_usage
element_types
enabled_roles
foreign_data_wrapper_options
foreign_data_wrappers
foreign_server_options
foreign_servers
foreign_table_options
foreign_tables
information_schema_catalog_name
key_column_usage
parameters
_pg_foreign_data_wrappers
_pg_foreign_servers
_pg_foreign_table_columns
_pg_foreign_tables
_pg_user_mappings
referential_constraints
role_column_grants
role_routine_grants
role_table_grants
role_udt_grants
role_usage_grants
routine_privileges
routines
schemata
sequences
sql_features
sql_implementation_info
sql_languages
sql_packages
sql_parts
sql_sizing
sql_sizing_profiles
table_constraints
table_privileges
tables
transforms
triggered_update_columns
triggers
udt_privileges
usage_privileges
user_defined_types
user_mapping_options
```

```
user_mappings
view_column_usage
view_routine_usage
views
view_table_usage
```

2. Nullwerte

Frage 1

```
select 1, count(*) from studenten
where semester < 13 or semester >= 13
union
select 2, count(*) from studenten;
```

Ergebnis:

```
1   9
2  10
```

Die beiden Abfragen geben eine unterschiedliche Anzahl Zeilen zurück, da die `WHERE`-Klausel `< 13 or >= 13` zwar für alle numerischen Werte zutrifft, nicht aber auf `NULL`. Die zweite Abfrage müsste um folgende `WHERE`-Klausel ergänzt werden, damit das Ergebnis konsitent wäre: `where not semester is null`.

3. Existenz

Was ist der Unterschied zwischen Mengenvergleichen mit `in` und `exists`? Wann kann man etwas nur mit dem `exist`-Operator abfragen?

- Der `in`-Operator prüft, ob ein Spaltenwert im Ergebnis einer Abfrage auftaucht.
- Der `exists`-Operator prüft, ob das Ergebnis einer Abfrage nicht leer ist.

Prüfungen für Studenten mit mehr als 10 Semstern, die von Professoren mit Rang C4 durchgeführt wurden:

```
/* Lösung mit exists-Operator */
select * from pruefen
where exists (
    select persnr
    from professoren
    where rang = 'C4' and pruefen.persnr = professoren.persnr
) and exists (
    select matrnr
    from studenten
```

```
    where semester > 10 and pruefen.matrnr = studenten.matrnr
);

/* Lösung mit in-Operator */
select * from pruefen
where persnr in (
    select persnr
    from professoren
    where rang = 'C4'
) and matrnr in (
    select matrnr
    from studenten
    where semester > 10
);
```

4. Fallunterscheidung

```
select matrnr,
case
    when note < 1.5
        then 'sehr gut'
    when note < 2.5
        then 'gut'
    when note < 3.5
        then 'befriedigend'
    when note <= 4.0
        then 'ausreichend'
    else
        'nicht bestanden'
end as bewertung
from pruefen
```

Warum wird hier jeweils nur geprüft, ob der Wert kleiner als die Untergrenze ist, aber nicht, ob er auch grösser als die Obergrenze ist?

- Die Untergrenze wird implizit durch die aufsteigende Reihenfolge der Fälle geprüft. Wenn die Ausführung beispielsweise bei `Note <= 4.0` angelangt ist, muss die vorherige Prüfung (`Note <= 3.5`) vorher schon gescheitert sein, ansonsten wäre die Prüfung `Note <= 4.0` gar nicht mehr ausgeführt worden. (Im Gegensatz zum `switch-case`-Konstrukt in Java, C etc. wird die Fallunterscheidung nach einem zutreffenden Fall automatisch beendet, ohne dass ein `break`-Statement notwendig wäre.)

5. Rekursion

```
with recursive r as (
    select vg.titel as v, nf.titel as n, 1 as l
    from voraussetzen vr
    join vorlesungen vg
        on vg.vorlnr = vr.vorgänger
    join vorlesungen nf
        on nf.vorlnr = vr.nachfolger
), pfad (von, nach, laenge, folge) as (
    select v, n, 1, v || ',' || n
    from r
    union all
    select p.von, e.n, p .laenge + 1, p.folge || ',' || e.n
    from r e join pfad p
        on p.nach = e.v
)
select * from pfad;
```

Lassen Sie die Query laufen. Was macht dies genau? Wo befindet sich der Rekursionsschritt? Erklären Sie die Funktionsweise dieser Query.

- Das Query gibt eine Tabelle mit vier Spalten zurück.
  - `von`: eine Vorlesungen am Anfang eines Pfades
  - `nach`: eine Vorlesung am Ende eines Pfades
  - `laenge`: die Länge des Pfades zwischen den beiden Vorlesungen
  - `folge`: der Pfad als Auflistung einzelner Vorlesungen
- Der Rekursionsschritt befindet sich auf der Zeile `from r e join pfad e`, wo der Ausdruck `r` (eine Vorgänger-Nachfolger-Beziehung) auf die bereits ermittelten Pfade gejoint wird.
- Die Tabelle wird solange mit sich selber “gejoint”, bis es in der Tabelle keine Zeilen keine mehr gibt. In diesem Fall ist das, wenn die Vorlesung keinen Nachfolger mehr hat. Dann ist man an einem Punkt angekommen, von dem kein Weg weiter führt. Dann wird ein Schritt zurückgegngen und das Vorgehen beginnt von Neuem.

6. Windowing

Schreiben Sie eine Query, welche pro Professor den Namen, die Anzahl Semesterwochenstunden (SWS) und dazu den SWS-Rang angibt. Der SWS-Rang gibt an, welcher Professor am meisten Vorlesungsstunden pro Semester gibt (SWS-Rank = 1), welcher am zweitmeisten unterrichtet (SWS-Rank = 2), usw. Professoren, die gleichviel unterrichten, sind auf dem gleichen SWS-Rang. Verwenden Sie dafür eine Window Function.

```sql
select name, sum(sws), dense_rank() over(order by sum(sws) desc) as swsrang
from professoren p
inner join vorlesungen v on v.gelesenVon = p.persnr
group by name
```