SQL-Abfragen und ihre Erklärungen

1. Studenten, die die Vorlesung "Ethik" hören

```
SELECT hoeren.MatrNr
FROM hoeren
WHERE hoeren.VorlNr = (SELECT Vorlesungen.VorlNr FROM Vorlesungen WHERE Titel = 'Et
```

Erklärung:

1. **Unterabfrage:** `(SELECT Vorlesungen.VorlNr FROM Vorlesungen WHERE Titel = 'Ethik')` Diese Unterabfrage sucht die Vorlesungsnummer (VorlNr) der Vorlesung mit dem Titel "Ethik" in der Tabelle "Vorlesungen".
2. **Vergleich:** `WHERE hoeren.VorlNr = (SELECT Vorlesungen.VorlNr FROM Vorlesungen WHERE Titel = 'Ethik')` Hier wird die Vorlesungsnummer in der Tabelle "hoeren" mit dem Ergebnis der Unterabfrage verglichen, um die Matrikelnummern (MatrNr) der Studenten zu finden, die die Vorlesung "Ethik" hören.

Beziehungen:

- **Primärschlüssel:** Vorlesungen.VorlNr identifiziert eindeutig jede Vorlesung.
- **Fremdschlüssel:** hoeren.VorlNr verweist auf Vorlesungen.VorlNr und stellt sicher, dass jede Vorlesungsnummer in "hoeren" gültig ist.

Ergebnisse:

matrnr
28106
29120
29999
29998
29997
29996

2. Studenten, die mit Schopenhauer eine Vorlesung gehört haben

```
SELECT DISTINCT s.Name
FROM Studenten s
JOIN hoeren h1 ON s.MatrNr = h1.MatrNr
JOIN hoeren h2 ON h1.VorlNr = h2.VorlNr
JOIN Studenten schopenhauer ON h2.MatrNr = schopenhauer.MatrNr
WHERE schopenhauer.Name = 'Schopenhauer' AND s.Name != 'Schopenhauer';
```

Erklärung:

1. **JOIN hoeren h1 ON s.MatrNr = h1.MatrNr:** Verknüpft die Tabelle "Studenten" mit "hoeren" basierend auf der Matrikelnummer, um die Vorlesungen (VorlNr) zu erhalten, die jeder Student hört.
2. **JOIN hoeren h2 ON h1.VorlNr = h2.VorlNr:** Verknüpft die Tabelle "hoeren" erneut, um alle Studenten zu finden, die dieselben Vorlesungen hören.
3. **JOIN Studenten schopenhauer ON h2.MatrNr = schopenhauer.MatrNr:** Verknüpft die Tabelle "hoeren" mit der Tabelle "Studenten", um die Matrikelnummern der Studenten zu finden, die mit Schopenhauer Vorlesungen hören.
4. **WHERE schopenhauer.Name = 'Schopenhauer' AND s.Name != 'Schopenhauer':** Filtert die Ergebnisse, um nur die Studenten zu erhalten, die mit Schopenhauer Vorlesungen hören und selbst nicht Schopenhauer sind.

Beziehungen:

- **Primärschlüssel:** Studenten.MatrNr identifiziert eindeutig jeden Studenten.
- **Fremdschlüssel:** hoeren.MatrNr verweist auf Studenten.MatrNr und stellt sicher, dass jede Matrikelnummer in "hoeren" gültig ist.

Ergebnisse:

name
Theophrastos
David
Fichte
Fabian

Lukas
Feuerbach
Abraham

3. Studenten, die alle Vorlesungen von Schopenhauer hören

```
SELECT s.Name
FROM Studenten s
WHERE NOT EXISTS (
    SELECT VorlNr
    FROM hoeren h
    JOIN Studenten schopenhauer ON h.MatrNr = schopenhauer.MatrNr
    WHERE schopenhauer.Name = 'Schopenhauer'
    EXCEPT
    SELECT VorlNr
    FROM hoeren
    WHERE MatrNr = s.MatrNr
) AND s.Name != 'Schopenhauer';
```

Erklärung:

1. **NOT EXISTS:** Überprüft, ob die Unterabfrage keine Ergebnisse zurückgibt, um sicherzustellen, dass der Student alle Vorlesungen von Schopenhauer hört.
2. **Unterabfrage Teil 1:** `SELECT VorlNr FROM hoeren h JOIN Studenten schopenhauer ON h.MatrNr = schopenhauer.MatrNr WHERE schopenhauer.Name = 'Schopenhauer'` Findet alle Vorlesungsnummern, die Schopenhauer hört.
3. **Unterabfrage Teil 2:** `EXCEPT` Agiert als MINUS
4. **Unterabfrage Teil 3:** `SELECT VorlNr FROM hoeren WHERE MatrNr = s.MatrNr` Findet alle Vorlesungsnummern, die der aktuelle Student hört.
5. **Ausgabe ohne Schopenhauer:** `AND s.Name != 'Schopenhauer'`.

Beziehungen:

- **Primärschlüssel:** Studenten.MatrNr identifiziert eindeutig jeden Studenten.
- **Fremdschlüssel:** hoeren.MatrNr verweist auf Studenten.MatrNr und stellt sicher, dass jede Matrikelnummer in "hoeren" gültig ist.

Ergebnisse:

name
Lukas
Fabian
Abraham
David

---

4. Vorlesungen mit mindestens zwei Voraussetzungen

```
SELECT Nachfolger
FROM voraussetzen
GROUP BY Nachfolger
HAVING COUNT(*) >= 2;
```

Erklärung:

1. **GROUP BY Nachfolger:** Gruppiert die Ergebnisse nach der Nachfolger-Vorlesung.
2. **HAVING COUNT(*) >= 2:** Filtert die Gruppen, um nur die Nachfolger-Vorlesungen zu behalten, die mindestens zwei Voraussetzungen haben.

Beziehungen:

- **Primärschlüssel/Fremdschlüssel:** voraussetzen.Vorgaenger und voraussetzen.Nachfolger verweisen auf Vorlesungen.VorlNr.

Weitere Quellen:

- https://youtu.be/feZwQ8LD7YI?si=If6mXAZcfOZ-QM4l

Ergebnisse:

nachfolger
5052

---

5. Vorlesungen und Anzahl der Prüfungen

```
SELECT v.VorlNr, COUNT(p.MatrNr) AS Anzahl
FROM Vorlesungen v
LEFT JOIN pruefen p ON v.VorlNr = p.VorlNr
GROUP BY v.VorlNr
ORDER BY Anzahl DESC;
```

Erklärung:

1. **LEFT JOIN pruefen p ON v.VorlNr = p.VorlNr:** Verknüpft die Tabelle "Vorlesungen" mit "pruefen" basierend auf der Vorlesungsnummer, um die Prüfungen zu zählen.
2. **GROUP BY v.VorlNr:** Gruppiert die Ergebnisse nach der Vorlesungsnummer.
3. **ORDER BY Anzahl DESC:** Sortiert die Ergebnisse nach der Anzahl der Prüfungen in absteigender Reihenfolge.

Beziehungen:

- **Primärschlüssel:** Vorlesungen.VorlNr identifiziert eindeutig jede Vorlesung.
- **Fremdschlüssel:** pruefen.VorlNr verweist auf Vorlesungen.VorlNr und stellt sicher, dass jede Vorlesungsnummer in "pruefen" gültig ist.

Ergebnisse:

vorlnr	anzahl
5001	6
5041	5
4052	4
5043	4
4630	1
5049	1
5216	0
5259	0
5022	0

5052	0

6. Professor(en) mit den meisten Assistenten

```
SELECT p.Name
FROM Professoren p
JOIN Assistenten a ON p.PersNr = a.Boss
GROUP BY p.PersNr, p.Name
HAVING COUNT(*) = (
    SELECT MAX(AssistentenAnzahl)
    FROM (
        SELECT COUNT(*) AS AssistentenAnzahl
        FROM Assistenten
        GROUP BY Boss
    ) AS Counts
);
```

Erklärung:

1. **JOIN Assistenten a ON p.PersNr = a.Boss:** Verknüpft die Tabellen "Professoren" und "Assistenten" basierend auf der Bedingung, dass die Personalnummer des Professors (p.PersNr) mit der Boss-Nummer in der Tabelle "Assistenten" (a.Boss) übereinstimmt.
2. **GROUP BY p.PersNr, p.Name:** Gruppiert die Ergebnisse nach der Personalnummer und dem Namen des Professors.
3. **HAVING COUNT(*) = (:** Vergleicht die Anzahl der Assistenten vom aktuellen Professor mit dem Wert von den Professoren mit den meisten Assistenten.
4. **SELECT MAX(AssistentenAnzahl):** Größter Wert der Unterabfrage (Unterpunkt 5).
5. **FROM (SELECT COUNT(*) AS AssistentenAnzahl FROM Assistenten GROUP BY Boss) AS Counts):** (Erwähnte Unterabfrage aus Unterpunkt 4) Gleicher Boss (Professor PersNr) werden zusammengefasst und gezählt.
6. **Punkt 3-5 zusammengefasst:** Filtert die Gruppen, um nur die Professoren mit der maximalen Anzahl an Assistenten zu behalten. Die Unterabfrage zählt die Anzahl der Assistenten für jeden Professor und wählt den höchsten Wert aus.

Beziehungen:

- **Primärschlüssel:** Professoren.PersNr identifiziert eindeutig jeden Professor.

- **Fremdschlüssel:** Assistenten.Boss verweist auf Professoren.PersNr und stellt sicher, dass jede Boss-Nummer in der Tabelle "Assistenten" einem gültigen Professor entspricht.

Ergebnisse:

name
Sokrates
Kopernikus

---

7. Var.1 Studenten, die alle Vorlesungen hören

```
SELECT s.Name
FROM Studenten s
JOIN hoeren h ON s.MatrNr = h.MatrNr
GROUP BY s.MatrNr, s.Name
HAVING COUNT(DISTINCT h.VorlNr) = (
    SELECT COUNT(*)
    FROM Vorlesungen
);
```

Erklärung:

1. **JOIN hoeren h ON s.MatrNr = h.MatrNr:** Verknüpft die Tabelle "Studenten" mit "hoeren" basierend auf der Matrikelnummer, um die Vorlesungen zu erhalten, die jeder Student hört.
2. **GROUP BY s.MatrNr, s.Name:** Gruppiert die Ergebnisse nach der Matrikelnummer und dem Namen der Studenten.
3. **HAVING COUNT(DISTINCT h.VorlNr) = (:** Zählt die Anzahl der Vorlesungen, die ein Student hört (und dann wird verglichen mit =).
4. *SELECT COUNT() FROM Vorlesungen):** Zählt die Gesamtanzahl der Vorlesungen.

Beziehungen:

- **Primärschlüssel:** Studenten.MatrNr identifiziert eindeutig jeden Studenten.
- **Fremdschlüssel:** hoeren.MatrNr verweist auf Studenten.MatrNr und stellt sicher, dass jede Matrikelnummer in "hoeren" gültig ist.

Ergebnisse:

name
Abraham
Fabian

---

7. Var.2 Studenten, die alle Vorlesungen hören

```
SELECT s.Name
FROM Studenten s
WHERE NOT EXISTS (
    SELECT VorlNr
    FROM Vorlesungen
    EXCEPT
    SELECT VorlNr
    FROM hoeren
    WHERE MatrNr = s.MatrNr
);
```

Erklärung:

1. **NOT EXISTS:** Überprüft, ob die Unterabfrage keine Ergebnisse zurückgibt, um sicherzustellen, dass der Student alle existierenden Vorlesungen hört.
2. **Unterabfrage:** `SELECT VorlNr FROM Vorlesungen` Diese Unterabfrage findet alle existierenden Vorlesungsnummern.
3. **EXCEPT SELECT VorlNr FROM hoeren WHERE MatrNr = s.MatrNr:** Diese Unterabfrage findet alle Vorlesungsnummern, die der aktuelle Student hört, und vergleicht sie mit allen existierenden Vorlesungen.

Beziehungen:

- **Primärschlüssel:** Studenten.MatrNr identifiziert eindeutig jeden Studenten.
- **Fremdschlüssel:** hoeren.MatrNr verweist auf Studenten.MatrNr und stellt sicher, dass jede Matrikelnummer in "hoeren" gültig ist.

Ergebnisse:

name
Abraham

Fabian

---

8. Anzahl der Prüfungen mit Note 1 oder 2

```
SELECT COUNT(*) AS AnzahlGutePruefungen
FROM pruefen
WHERE Note < 3.0;
```

Erklärung:

1. **WHERE Note < 3.0:** Filtert die Ergebnisse, um nur die Prüfungen mit einer Note kleiner als 3.0 zu zählen.
2. *SELECT COUNT() AS AnzahlGutePruefungen:** Zählt die Anzahl der Prüfungen, die die Bedingung erfüllen, und gibt das Ergebnis als "AnzahlGutePruefungen" zurück.

Beziehungen:

- **Primärschlüssel:** pruefen.MatrNr und pruefen.VorlNr identifizieren eindeutig jede Prüfung.

Ergebnisse:

anzahlgutepruefungen
11

---

9. Übersicht der Studierenden mit Durchschnittsnote und Varianz

```
SELECT s.MatrNr, s.Name,
       ROUND(AVG(p.Note), 2) AS Durchschnittsnote,
       ROUND(VAR_POP(p.Note), 2) AS Varianz
FROM Studenten s
LEFT JOIN pruefen p ON s.MatrNr = p.MatrNr
GROUP BY s.MatrNr, s.Name;
```

Erklärung:

1. **LEFT JOIN pruefen p ON s.MatrNr = p.MatrNr:** Verknüpft die Tabelle "Studenten" mit

"pruefen" basierend auf der Matrikelnummer, um die Prüfungsnoten zu erhalten.

2. **SELECT s.MatrNr, s.Name, ROUND(AVG(p.Note), 2) AS Durchschnittsnote, ROUND(VAR_POP(p.Note), 2) AS Varianz:** Wählt die Matrikelnummer und den Namen der Studenten sowie den Durchschnitt und die Varianz der Prüfungsnoten. Durchschnitt und Varianz sind auf 2 Nachkommastellen gerundet mittels `ROUND( ... , 2)`.

3. **GROUP BY s.MatrNr, s.Name:** Gruppiert die Ergebnisse nach der Matrikelnummer und dem Namen der Studenten.

Beziehungen:

- **Primärschlüssel:** Studenten.MatrNr identifiziert eindeutig jeden Studenten.
- **Fremdschlüssel:** pruefen.MatrNr verweist auf Studenten.MatrNr und stellt sicher, dass jede Matrikelnummer in "pruefen" gültig ist.

Ergebnisse:

matrnr	name	durchschnittsnote	varianz
29998	Fabian	3.50	1.25
28106	Carnap	1.00	0.00
29120	Theophrastos	NULL	NULL
29997	Abraham	2.60	1.84
26120	Fichte	3.00	0.00
29996	David	2.75	2.19
26830	Aristoxenos	NULL	NULL
29999	Lukas	2.75	2.19
29555	Feuerbach	NULL	NULL
29995	Popper	NULL	NULL
24002	Xenokrates	NULL	NULL
25403	Jonas	2.00	0.00
27550	Schopenhauer	2.00	0.00

10. Namen, die in mindestens zwei verschiedenen Tabellen auftreten

```
SELECT Name
FROM (
    SELECT Name FROM Studenten
    UNION ALL
    SELECT Name FROM Professoren
    UNION ALL
    SELECT Name FROM Assistenten
) AS AllNames
GROUP BY Name
HAVING COUNT(*) >= 2;
```

Erklärung:

1. **UNION ALL:** Kombiniert die Namen aus den Tabellen "Studenten", "Professoren" und "Assistenten".
2. **GROUP BY Name:** Gruppiert die kombinierten Namen.
3. *HAVING COUNT() >= 2:** Filtert die Gruppen, um nur die Namen zu behalten, die in mindestens zwei verschiedenen Tabellen auftreten.

Beziehungen:

- **Primärschlüssel:** Studenten.Name, Professoren.Name und Assistenten.Name identifizieren eindeutig jeden Eintrag in ihren jeweiligen Tabellen.

Info: `UNION ALL` weniger effizient als INNER JOIN, aber für diese Aufgabe geeignet!

Ergebnisse:

name
Popper

---

11. Vorlesungen und ihre direkten und indirekten Voraussetzungen

```
WITH RECURSIVE VoraussetzungenRekursiv AS (
```

```
    SELECT Vorgaenger, Nachfolger
    FROM voraussetzen
    UNION
    SELECT v.Vorgaenger, vr.Nachfolger
    FROM voraussetzen v
    JOIN VoraussetzungenRekursiv vr ON v.Nachfolger = vr.Vorgaenger
)
SELECT DISTINCT Nachfolger, Vorgaenger
FROM VoraussetzungenRekursiv
ORDER BY Nachfolger, Vorgaenger;
```

Erklärung:

1. **WITH RECURSIVE VoraussetzungenRekursiv AS (SELECT Vorgaenger, Nachfolger FROM voraussetzen UNION SELECT v.Vorgaenger, vr.Nachfolger FROM voraussetzen v JOIN VoraussetzungenRekursiv vr ON v.Nachfolger = vr.Vorgaenger):** Definiert eine rekursive CTE (Common Table Expression) namens "VoraussetzungenRekursiv", die die direkten und indirekten Voraussetzungen für Vorlesungen ermittelt.
2. **SELECT DISTINCT Nachfolger, Vorgaenger:** Wählt die eindeutigen Paare von Nachfolger- und Vorgänger-Vorlesungen aus der rekursiven CTE.
3. **ORDER BY Nachfolger, Vorgaenger:** Sortiert die Ergebnisse nach Nachfolger und Vorgänger.

Beziehungen:

- **Primärschlüssel/Fremdschlüssel:** voraussetzen.Vorgaenger und voraussetzen.Nachfolger verweisen auf Vorlesungen.VorlNr.

Ergebnisse:

nachfolger	vorgaenger
5041	5001
5043	5001
5049	5001
5052	5001
5052	5041
5052	5043
5216	5001

5216	5041
5259	5001
5259	5041
5259	5043
5259	5052