DB PRAKTIKUM INFO BOOK

1. UMGEBUNG AUFBAUEN

2. SQUIRRELSQL

Windows, Linux, Mac

Tutorial findet ihr unter:

https://www.cs.hs-rm.de/~knauf/Datenbanken2021/index.html

3. HEIDSQL

WINDOWS

Wird im Video ab Minute~5 gezeigt

https://video.cs.hs-rm.de/course/131/lecture/1401

4. DATAGRIP

WINDOWS, LINUX, MAC

Über die HSRM-Mail Adresse kann man (Stand 29.05.2021) eine Studenten Lizenz für Jetbrains Produkte beantragen. Mit dieser Lizenz kann die Vollversion von DataGrip verwendet werden.

STUDENTEN LIZENZ BEANTRAGEN:

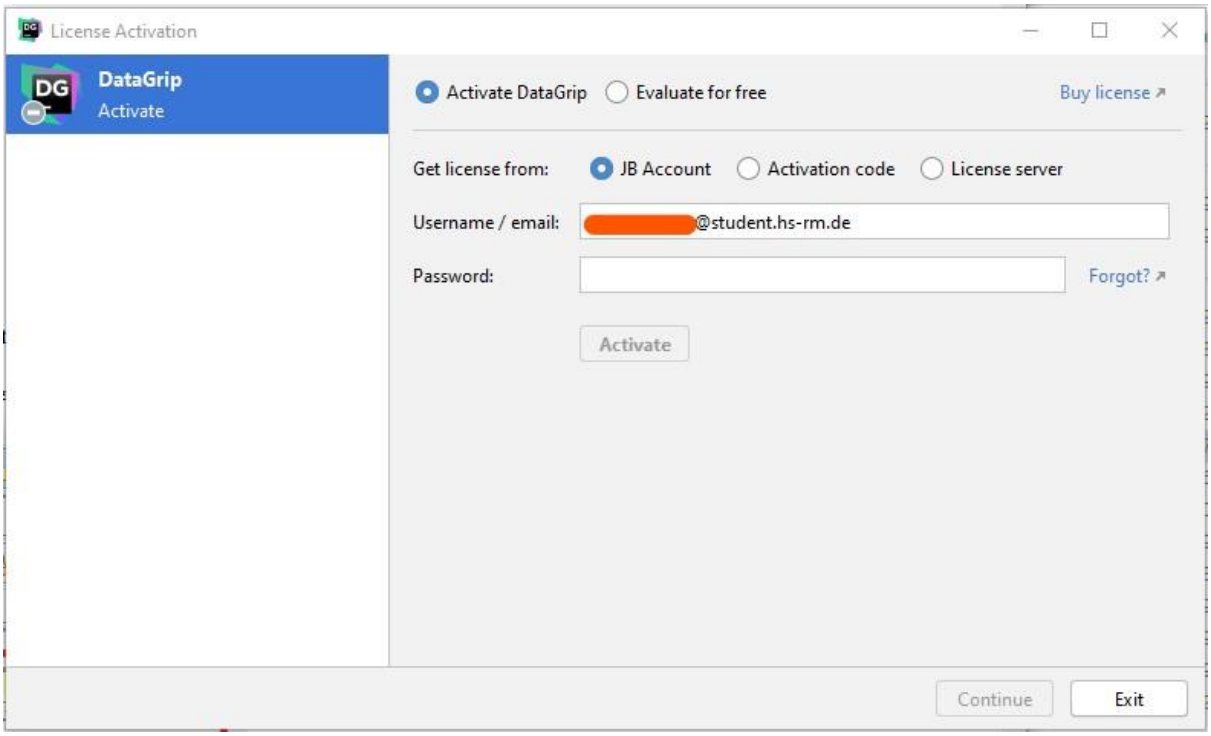
https://www.jetbrains.com/de-de/community/education/#students

DATAGRIP DOWNLOAD:

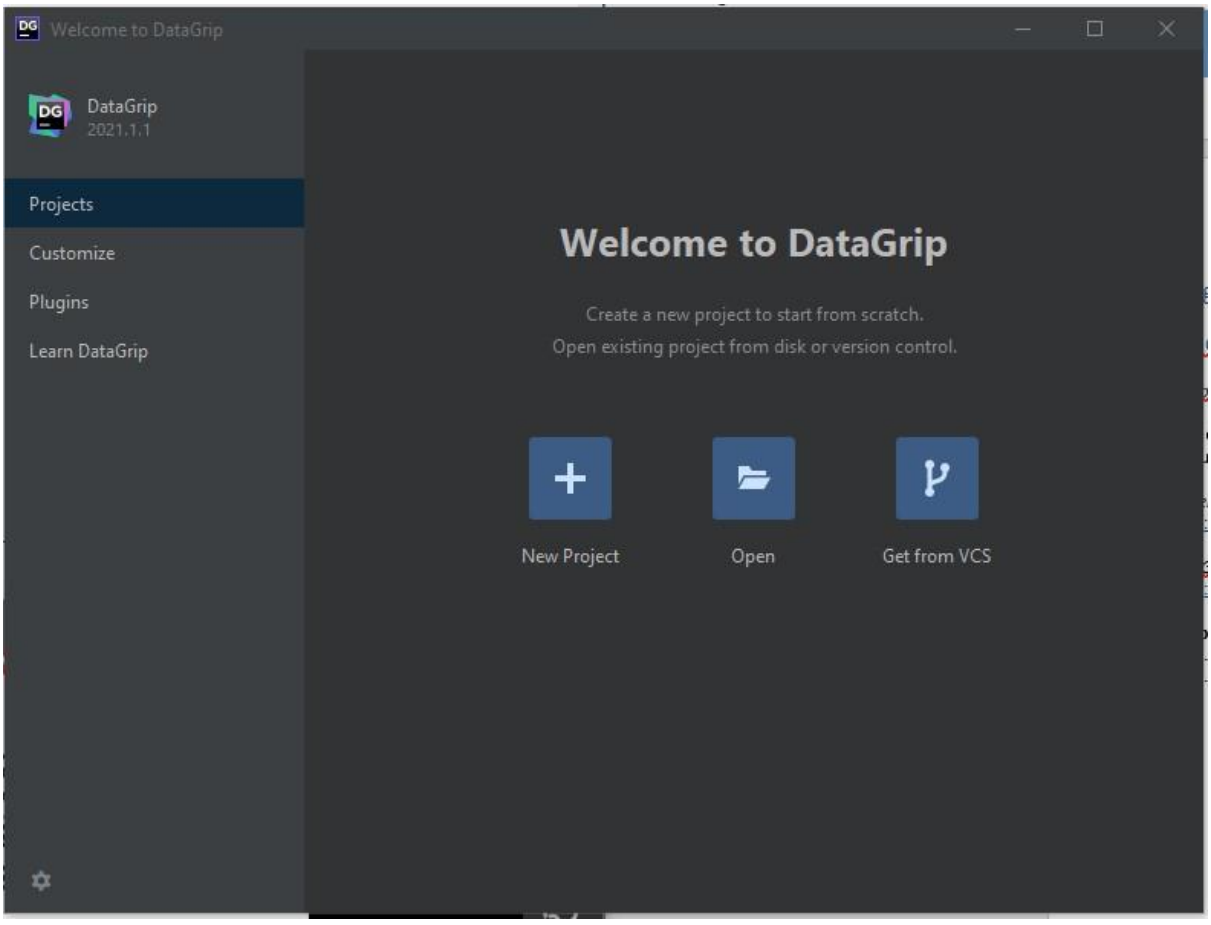
https://www.jetbrains.com/de-de/datagrip/

5. SETUP

1. DataGrip herunterladen und die exe ausführen
2. HSRM Mail Adresse angeben + Passwort und auf ‚Activate' klicken

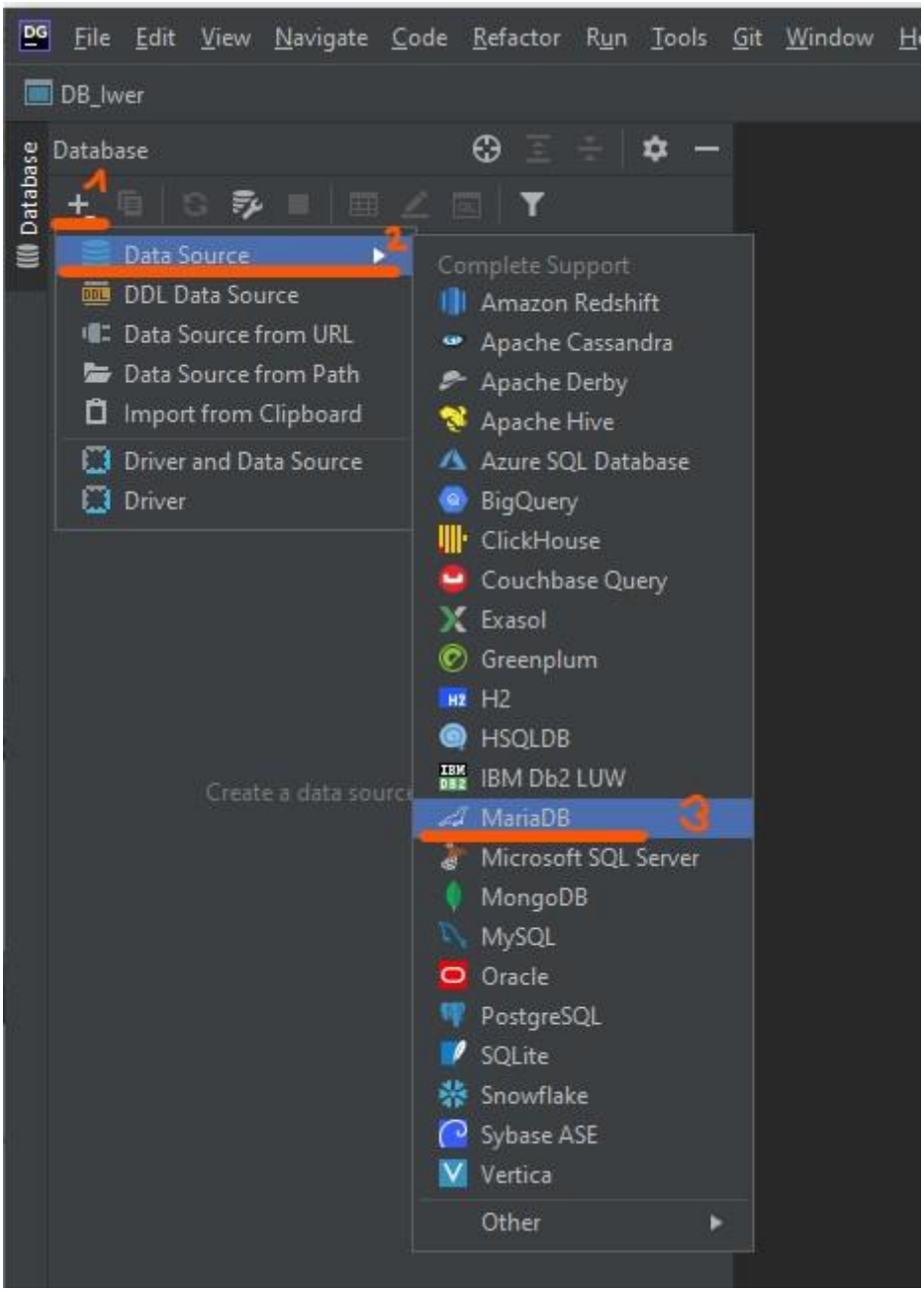


3. Auf ‚New Project' klicken

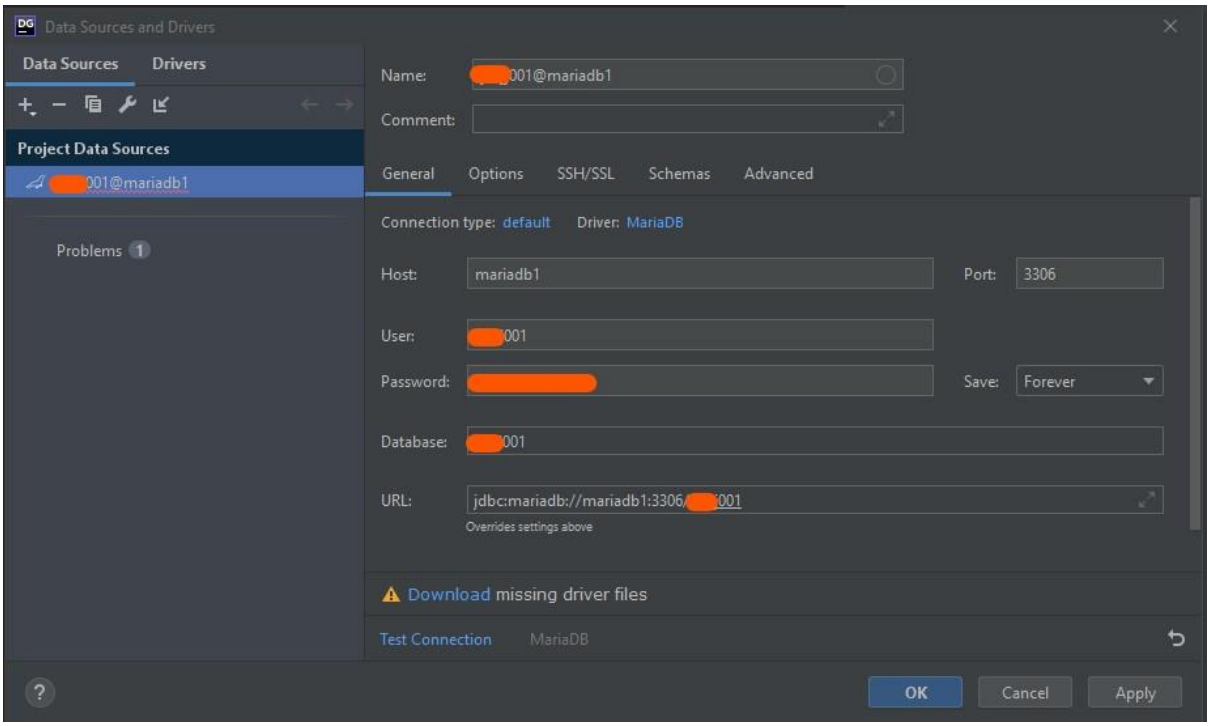


6. ARBEITEN MIT DER HOCHSCHUL-DB

1. Sicherstellen, dass die VPN Verbindung zur Informatik (nicht zur Hochschule!) steht Anleitung: https://doku.cs.hs-rm.de/doku.php?id=openvpn
2. In DataGrip oben links auf das Plus klicken und eine neue DataSource hinzufügen, wählt hier ‚MariaDB'

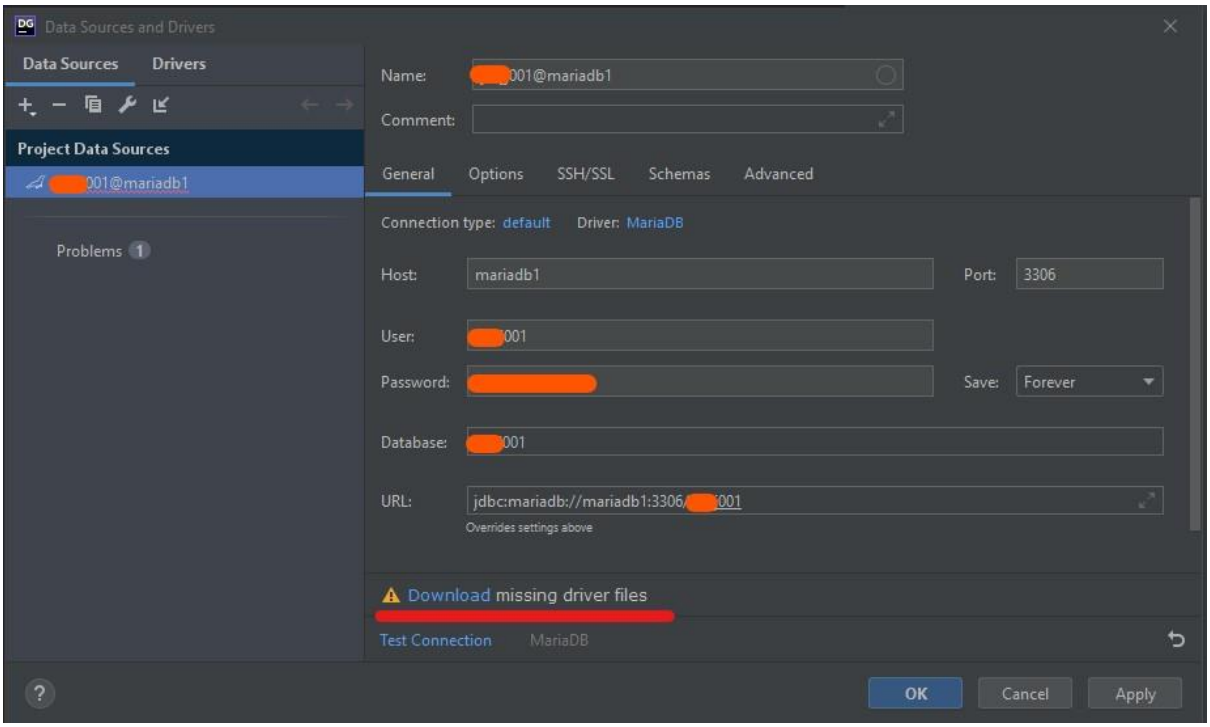


3. Unter Host ‚mariadb1' eingeben und für User und Database den Informatik Login Namen + Passwort nicht vergessen

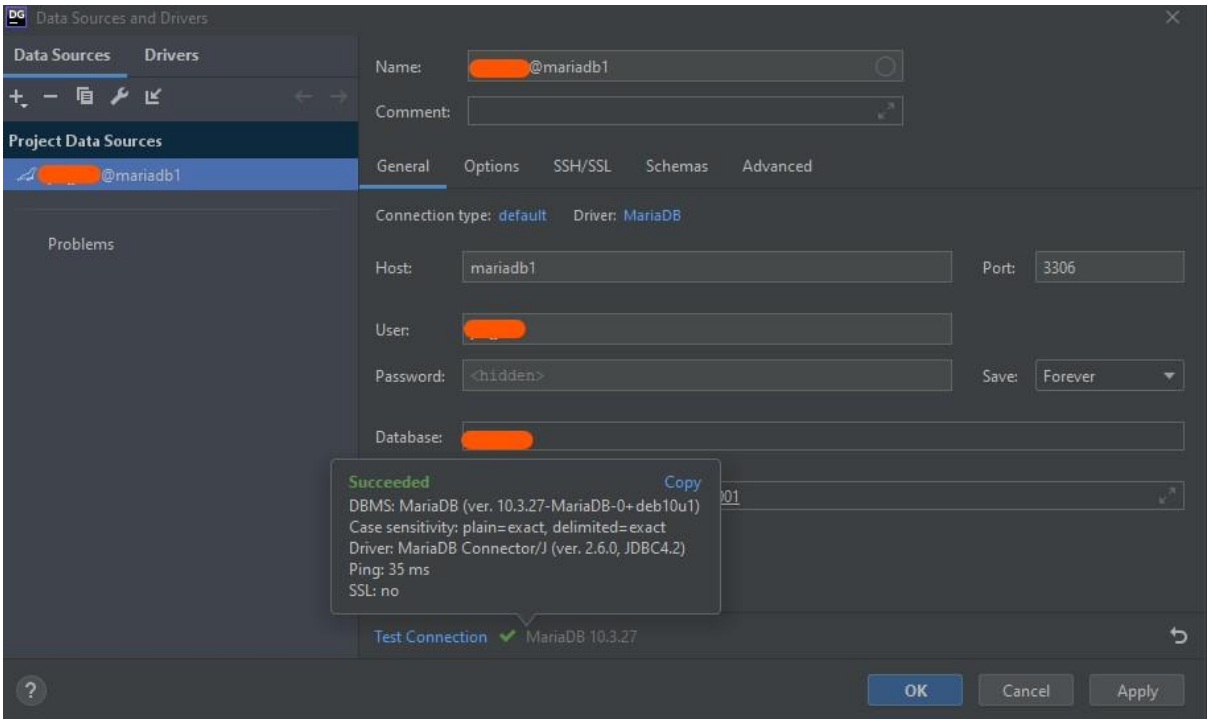


4. Falls die MariaDB Driver noch nicht heruntergeladen wurden, bitte auf ‚Download missing drover

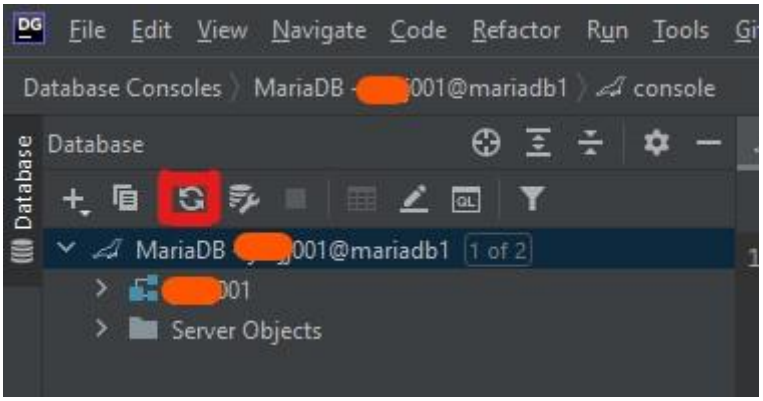
   files' klicken



5. Auf ‚Test Connection' klicken

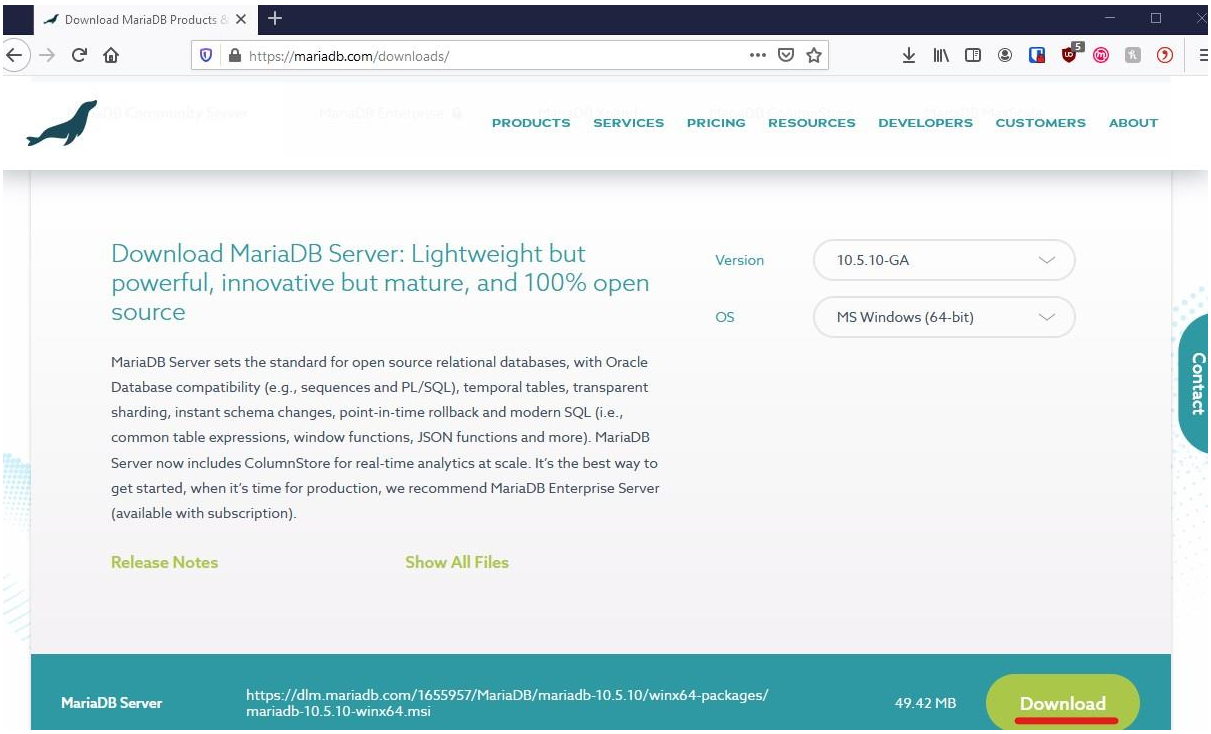


6. Apply und dann Ok
7. Wählt eure Datenbank Verbindung aus und klickt auf die beiden Pfeile, um eine Connection aufzubauen

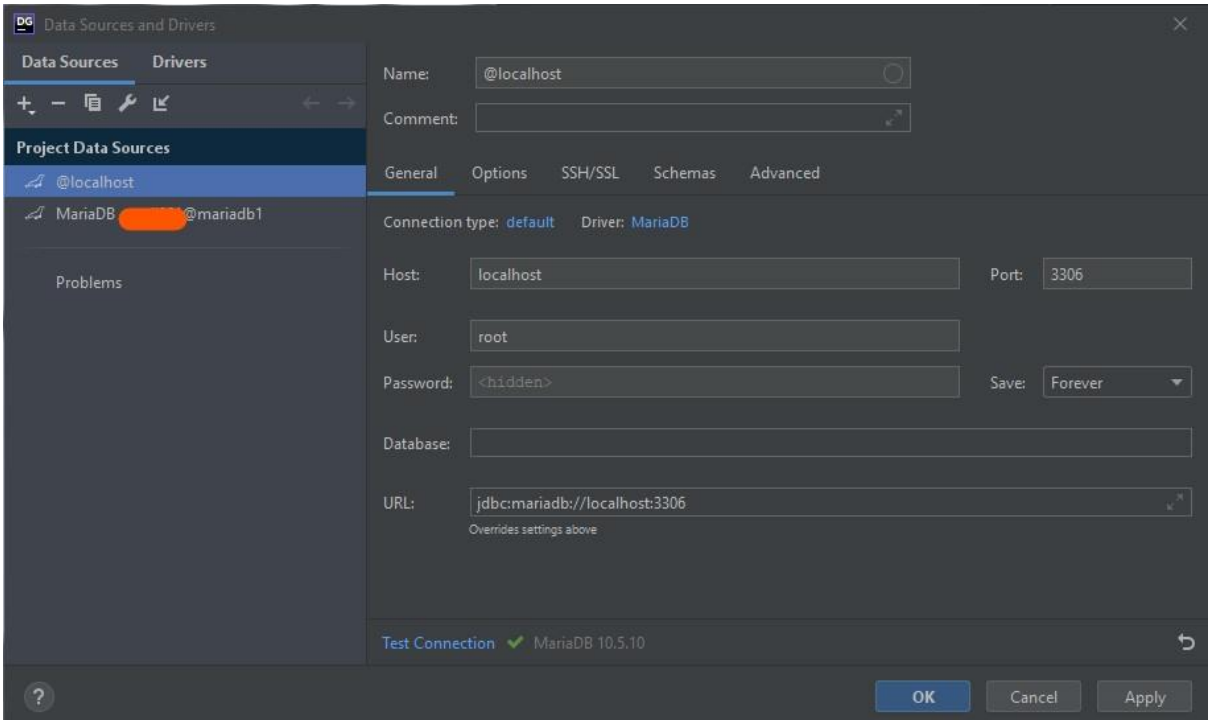


7. MIT EINER LOKALEN DB ARBEITEN

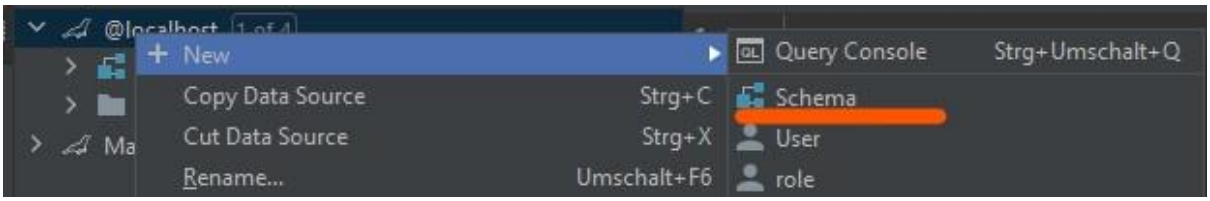
1. MariaDB Server herunterladen https://mariadb.com/downloads/



2. Beim Installieren des Servers für MariaDB muss ein Passwort für den ‚root' User festgelegt werden, diese Anmeldedaten werden in DataGrip für die DataSource verwendet

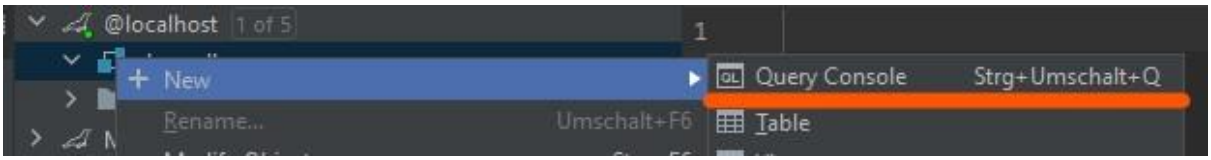


3. Die Verbindung mit localhost sollte nun stehen, es kann über einen Rechtsklick entweder manuell eine Datenbank (ein Schema) angelegt werden, oder per SQL Befehl :)

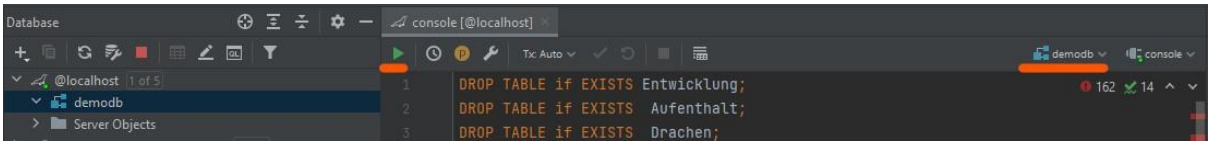


8. SQL BEFEHLE

1. Rechtsklick auf euer Schema und New Query Console auswählen



2. Hier können SQL Befehle eingegeben werden und mit dem Play Button ausgeführt werden



3. Achtung:

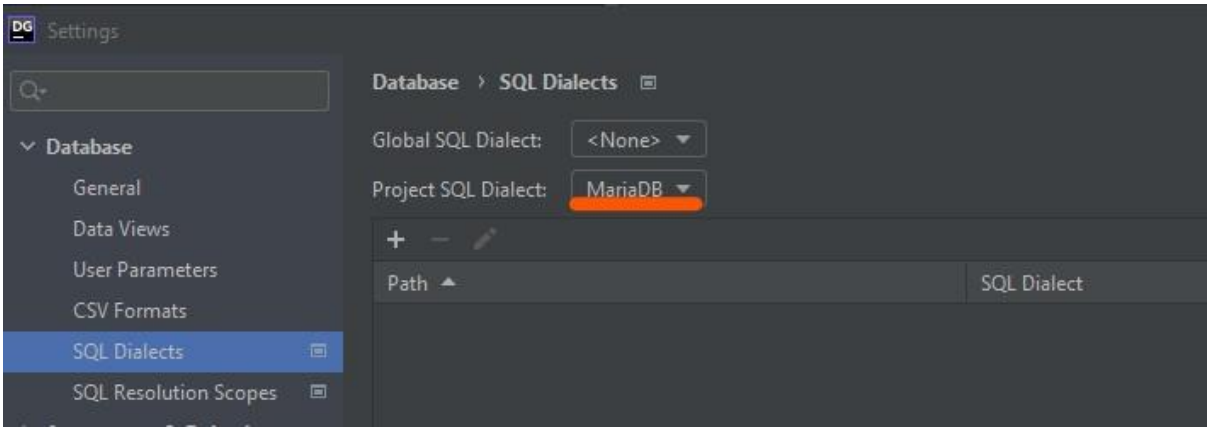
   Es wird immer nur der markierte Befehl ausgeführt (in diesem Fall grün umrandet)

   Um alle Befehle auszuführen müsst ihr alles markieren und dann auf ‚Ausführen' klicken

   Oben rechts muss beim Schema Symbol außerdem die Datenbank ausgewählt werden, auf der die Befehle ausgeführt werden sollen

9. SQL DIALECT EINSTELLEN

1. Über File -> Settings -> Database -> SQL Dialect
2. Über Project SQL Dialect ‚MariaDB' einstellen



10. MARIADB TREIBER VERSION CHECKEN

Über File -> DataSources -> Drivers -> MariaDB kann die Version überprüft und geändert werden.

GRUPPIERUNG

Gegeben ist folgende Relation:

Menschen

Name	Vorname	Lieblingsfarbe	Alter
Müller	Heinz	Blau	23
Baum	Claudia	Lila	19
NoName	Test	Rosa	55
Baum	Hans	Lila	20
Berg	Claudia	Blau	23

Um eine Gruppering über ein bestimmtes Attribut zu machen, überlegen wir uns erst welche Gruppen wir finden können.

γ_{name} (Menschen)

Name
Müller
Baum
NoName
Berg

Die Grupperiung geschieht pro Zeile anhand der spezifizierten Attribute.

Gefundene Gruppen:

{Müller, Baum, NoName, Berg}

Inhalt der Gruppen:

Müller = { (Müller, Heins, Blau, 23) }
Baum = { (Baum, Claudia, Lila, 19),
(Baum, Hans, Lila, 20) }
NoName = { (NoName, Test, Rosa, 55) }
Berg = { (Berg, Clausia, Blau,23) }

$\gamma_{name, COUNT(*)}$ (Menschen)

Name	COUNT(*)
Müller	1
Baum	2
NoName	1
Berg	1

Pro Gruppe werden alle Einträge für das Attribut „Alter“ summiert. Dadurch erhält die Gruppe „Baum“ den Wert 38 -> 19 + 20

$\gamma_{name, SUM(Alter)}$ (Menschen)

Name	SUM(Alter)	
Müller	23	
Baum	39	
NoName	55	
Berg	23	

Pro Gruppe werden alle Einträge für das Attribut „Alter“ summiert. Dadurch erhält die Gruppe „Baum“ den Wert 38 -> 19 + 20

GRUPPIERUNG ÜBER ZWEI ATTRIBUTE

γLieblingsfarbe, Alter, COUNT(*) (Menschen)

Wird eine Gruppierung über zwei Attribute gemacht, müssen BEIDE Attribute gleich sein um die Zeilen zu gruppieren.

Gefundene Gruppen:

{(Blau, 23), (Lila, 19), (Rosa, 55), (Lila, 20)}

Lieblingsfarbe	Alter	COUNT(*)
Blau	23	2
Lila	19	1
Rosa	55	1
Lila	20	1

Wichtig: Die Gruppierung passiert hier also nicht einfach pro Attribut, sondern pro Kombination der angegebenen Attribute. Die COUNT(*) Spalte zeigt wie viele Einträge wir in den Gruppen haben. Für die erste Zeile haben wir also zwei Einträge die unter (Blau, 23) gruppiert werden können.

JOINS

Gegeben ist folgende Relation:

Menschen

Id	Name	**Vorname**	**Lieblingsfarbe**	**Alter**
1	Müller	Heinz	1	23
2	Baum	Claudia	17	19
3	NoName	Test	4	55
4	Baum	Hans	17	20
5	Berg	Claudia	1	23

Farben

Id	**Bezeichnung**
1	Blau
17	Lila
3	Grün
2	Gelb

R

A	***B***	***C***
1	*2*	*3*
1	*2*	*4*
3	*5*	*3*

S

B	***C***	***D***
2	*7*	*9*
5	*4*	*10*
2	*3*	*8*

KARTESISCHES PRODUKT (KREUZPRODUKT)

Menschen × Farben

Alles wird mit allem gejoint! Es kommen also 5 * 4 = 20 Tupel raus!

Menschen.Id	Name	Vorname	Lieblingsfarbe	Alter	Farben.Id	Bezeichnung
1	Müller	Heinz	1	23	1	Blau
1	Müller	Heinz	1	23	17	Lila
1	Müller	Heinz	1	23	3	Grün
1	Müller	Heinz	1	23	2	Gelb
2	Baum	Claudia	17	19	1	Blau
2	Baum	Claudia	17	19	17	Lila
2	Baum	Claudia	17	19	3	Grün
2	Baum	Claudia	17	19	2	Gelb
3	NoName	Test	4	55	1	Blau
3	NoName	Test	4	55	17	Lila
3	NoName	Test	4	55	3	Grün
3	NoName	Test	4	55	2	Gelb
4	Baum	Hans	17	20	1	Blau
4	Baum	Hans	17	20	17	Lila
4	Baum	Hans	17	20	3	Grün
4	Baum	Hans	17	20	2	Gelb
5	Berg	Claudia	1	23	1	Blau
5	Berg	Claudia	1	23	17	Lila
5	Berg	Claudia	1	23	3	Grün
5	Berg	Claudia	1	23	2	Gelb

Wenn man eine Tabelle mit sich selbst joint erhält man alle möglichen Kombinationen von Paaren

```
SELECT * FROM Menschen, Farben;

SELECT * FROM Menschen CROSS JOIN Farben;

SELECT * FROM Menschen JOIN Farben;
```

➔ Normaler Join ohne Bedingung ist ein Kreuzprodukt

Farben × Farben

Id	Bezeichnung	Id	Bezeichnung
1	Blau	1	Blau
1	Blau	17	Lila
1	Blau	3	Grün
1	Blau	2	Gelb
17	Lila	1	Blau
17	Lila	17	Lila
17	Lila	3	Grün
17	Lila	2	Gelb
3	Grün	1	Blau
3	Grün	17	Lila
3	Grün	3	Grün
3	Grün	2	Gelb
2	Gelb	1	Blau
2	Gelb	17	Lila
2	Gelb	3	Grün
2	Gelb	2	Gelb

```
SELECT * FROM Farben, Farben AS f;
```

➔ Eine Tabelle muss unbenannt werden um im SELECT die Spalten unterscheiden zu können, sonst heißen alle Spalten 1:1 gleich

```
SELECT * FROM Farben CROSS JOIN Farben AS f;
```

```
SELECT * FROM Farben JOIN Farben AS f;
```

NATURAL JOIN

Hier werden gleichnamige Spalten gejoint. Es werden nur die Tupel mit gleichen Werten ausgegeben.

Menschen ⋈ Farben

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün

Achtung:

In diesem Beispiel ist der Natural Join total unsinnig! Es bringt nichts die beiden Id Spalten zu joinen weil sie zwei semantisch verschiedene Bedeutungen haben! So kann man sehen, dass bei Claudia Baum z.B. ‚Gelb' als Lieblingsfarbe raus kommt, dabei ist ihre Lieblingsfarbe Lila!

```
SELECT * FROM Menschen NATURAL JOIN Farben;
```

Bei einem natural Join müssen alle Werte der gleichnamigen Attribute übereinstimmen! D.h. Wenn S und R gejoint werden, müssen die Werte aus den Spalten B und C ***beide*** *gleich sein!*

R ⋈ S

A	*B*	*C*	*D*
1	*2*	*3*	*8*

```
SELECT * FROM R NATURAL JOIN S;
```

THETA JOIN

Der Theta Join ist ein Join mit zusätzlicher Join-Bedingung. Wir können also selbst bestimmen unter welchen Bedingungen zwei Tupel gejoint werden. Ein Theta Join ist also eigentlich ein Kreuzprodukt bei dem wir ‚nachträglich' mit Bedingungen Zeilen rausfiltern.

Menschen $\bowtie_{\text{Menschen.Lieblingsfarbe = Farben.Id}}$ Farben

Mit diesem Join können wir die Bezeichnungen für die Lieblingsfarben unserer Menschen finden!

Menschen.Id	Name	Vorname	Lieblingsfarbe	Alter	Farben.Id	Bezeichnung
1	Müller	Heinz	1	23	1	Blau
2	Baum	Claudia	17	19	17	Lila
4	Baum	Hans	17	20	17	Lila
5	Berg	Claudia	1	23	*1*	*Blau*

Hier sind die Werte in den Spalten ‚Lieblingsfarbe' und ‚Farben.Id' gleich. Man kann sehen, dass der Mensch ‚NoName Test' nicht aufgeführt ist. Seine Lieblingsfarbe ist die 4, für 4 haben wir jedoch keinen Eintrag in der Farben Tabelle, weshalb er weg gelassen wird.

Achtung:
Es ist bei der Bedingung wichtig, dass vor ‚Id' die Tabelle spezifiziert wird also ‚Farben.Id', da die Spalte ‚Id' in beiden Tabellen existiert.

```
SELECT * FROM Menschen JOIN Farben
ON(Lieblingsfarbe=Farbe.Id);

SELECT * FROM Menschen JOIN Farben
WHERE Lieblingsfarbe = Farbe.Id;
```

➔ Die ON Variante ist zu bevorzugen, da es sich um eine JOIN Bedingung handelt und das semantisch korrekt ist. Wenn man immer die WHERE Variante verwendet macht man schneller Fehler, da oft ungewollte Kreuzprodukte entstehen.

Menschen $\bowtie_{\text{Menschen.Lieblingsfarbe = Farben.Id AND Bezeichnung ='Blau'}}$ Farben

Es können zusätzliche Bedingungen für den Join definiert werden. Hier wollen wir alle Einträge, dessen Bezeichnung ‚Blau' ist.

Menschen.Id	Name	Vorname	Lieblingsfarbe	Alter	Farben.Id	Bezeichnung
1	Müller	Heinz	1	23	1	Blau
5	Berg	Claudia	1	23	*1*	*Blau*

Achtung:
‚Blau' muss hier in Anführungszeichen stehen da es ein String ist!

SEMI-JOIN

Ein Links-Semi-Join bzw. ein Rechts-Semi-Join kann man sich als natural Join vorstellen, bei dem man nur die Attribute von der linken oder rechten Relation ausgibt.

Für den natural Join von R und S kam folgendes raus:

R ⋈ S

A	B	C	D
1	2	3	8

```
SELECT * FROM R NATURAL JOIN S;
```

R ⋉ S - Links- Semi-Join

Wir nehmen die Attribute von der Relation wo das X geschlossen wird.

A	B	C
1	2	3

```
SELECT R.* FROM R NATURAL JOIN S;
```

R ⋊ S - Rechts- Semi-Join

Wir nehmen die Attribute von der Relation wo das X geschlossen wird.

B	C	D
2	3	8

```
SELECT S.* FROM R NATURAL JOIN S;
```

Menschen ⋈ Farben

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün

Menschen ⋉ Farben

Name	Vorname	Lieblingsfarbe	Alter	Id
Müller	Heinz	1	23	1
Baum	Claudia	17	19	2
NoName	Test	4	55	3

```
SELECT Menschen.* FROM Menschen NATURAL JOIN Farben;
```

Menschen ⋊ Farben

Id	Bezeichnung
1	Blau
2	Gelb
3	Grün

```
SELECT Farben.* FROM Menschen NATURAL JOIN Farben;
```

ANTI JOIN

Den Anti Join könnte man als umgekehrten natural Join bezeichnen oder aber auch als ‚Rest' des natural Joins.

Wir nehmen genau die Tupel, die beim natural Join **keinen** Partner gefunden haben.

$R \bowtie S$

A	B	C	D
1	2	3	8

Die Tupel die keinen Partner gefunden haben

R

A	B	C
1	2	4
3	5	3

S

B	C	D
2	7	9
5	4	10

R |> S – Links-Anti

A	B	C
1	2	4
3	5	3

```
SELECT * FROM R NATURAL
LEFT OUTER JOIN S WHERE
S.b IS NULL;
```

R <| S – Rechts-Anti-Join

B	C	D
2	7	9
5	4	10

```
SELECT * FROM R NATURAL
RIGHT OUTER JOIN S WHERE
R.b IS NULL;
```

Menschen |> Farben

Id	Name	Vorname	Lieblingsfarbe	Alter
4	Baum	Hans	17	20
5	Berg	Claudia	1	23

```
SELECT * FROM Menschen NATURAL LEFT OUTER JOIN
Farben WHERE Farben.Id IS NULL;
```

Menschen <| Farben

Id	Bezeichnung
3	Grün
2	Gelb

```
SELECT * FROM Menschen NATURAL LEFT OUTER JOIN
Farben WHERE Menschen.Id IS NULL;
```

Beispiel Medienhandel:
Alle Medienartikel, die nicht in der Tabelle dvd enthalten sind:

```
SELECT * FROM medienartikel NATURAL LEFT OUTER JOIN
    dvd WHERE dvd.a_nr IS NULL;
```

OUTER JOIN

Der Outer Join sind die Ergebnisse eines natural Joins vereinigt mit den Ergebnissen eines Anti Joins!

R ⋈ S

A	*B*	*C*	*D*
1	*2*	*3*	*8*

R |> S – Links-Anti-Join

A	*B*	*C*
1	*2*	*4*
3	*5*	*3*

R ⟕o S - Links- Outer-Join

A	*B*	*C*	*D*
1	*2*	*3*	*8*
1	*2*	*4*	*NULL*
3	*5*	*3*	*NULL*

Die Ergebnisse, die vom Anti Join übernommen wurden erhalten bei der zusätzlichen Spalte NULL Werte.

```
SELECT * FROM R NATURAL LEFT OUTER JOIN S;
```

$R \rtimes o\ S$ - Rechts- Outer-Join

A	B	C	D
1	2	3	8
NULL	2	7	9
NULL	5	4	10

```
SELECT * FROM R NATURAL RIGHT OUTER JOIN S;
```

$R \bowtie o\ S$ – Full-Outer-Join

Beim Full Outer Join nimmt man die Ergebnisse beider Anti-Joins:

A	B	C	D
1	2	3	8
1	2	4	NULL
3	5	3	NULL
NULL	2	7	9
NULL	5	4	10

```
(SELECT * FROM R NATURAL LEFT OUTER JOIN S)
UNION
(SELECT * FROM R NATURAL RIGHT OUTER JOIN S);
```

Menschen ⟕o Farben - Links- Outer-Join

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün
Baum	Hans	17	20	4	NULL
Berg	Claudia	1	23	5	NULL

```
SELECT * FROM Menschen NATURAL LEFT OUTER JOIN
Farben;
```

Menschen ⟖o Farben - Rechts- Outer-Join

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün
NULL	NULL	NULL	NULL	3	Grün
NULL	NULL	NULL	NULL	2	Gelb

```
SELECT * FROM Menschen NATURAL RIGHT OUTER JOIN
Farben;
```

Menschen ⟗o Farben – Full-Outer-Join

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün
Baum	Hans	17	20	4	NULL
Berg	Claudia	1	23	5	NULL
NULL	NULL	NULL	NULL	3	Grün
NULL	NULL	NULL	NULL	2	Gelb

```
(SELECT * FROM Menschen NATURAL LEFT OUTER JOIN
Farben)
UNION
(SELECT * FROM Menschen NATURAL RIGHT OUTER JOIN
Farben);
```

SQL: TIPPS

GROß- UND KLEINSCHREIBUNG

Tabellen- und Attributnamen sind case sensitive!

```
SELECT * FROM Drachen;
SELECT * FROM drachen;
```

In den Aufgaben sind die Tabellen in der Regel großgeschrieben! Die untere Abfrage funktioniert also nicht. Genauso verhält es sich mit den Attributen:

```
SELECT vater FROM Drachen;
SELECT Vater FROM Drachen;
```

In der Drachenrelation ist ‚vater' klein geschrieben, die untere Abfrage funktioniert also nicht.

SEMIKOLON

Ans Ende jeder Abfrage gehört ein Semikolon!

VERSCHACHTELTE SELECTS

Nach verschachtelten SELECTS muss ein Name angegeben werden:

```
SELECT * FROM <Relation1> NATURAL JOIN
    (SELECT * FROM <Relation2>
          NATURAL JOIN <Relation3> ) AS R2;
```

GROUP BY: HAVING

In MariaDB, zero (0) means false and non-zero means true. The BOOLEAN and BOOL are the synonym of TINYINT(1) .
(https://www.mariadbtutorial.com/mariadb-basics/mariadb-data-types/)

Oft wird die HAVING Clause missbraucht bzw. falsch verstanden, wenn man mit Aggreagtionsfunktionen arbeitet.

Nehmen wir die Menschen Tabelle:

Die Aufgabe ist es den ältesten Menschen zu finden.

Falsch:

```
SELECT id FROM Menschen GROUP BY id HAVING MAX(alter);
```

Jeder Mensch hat ein Alter != 0! D.h. die HAVING Clause wird immer eine Zahl zurück geben, die größer ist als 0. Da die HAVING Clause eine Condition, also eine Bedingung, erwartet, wird diese Zahl als TRUE interpretiert.

Dadurch enthält die Ausgabe einfach wieder jeden Menschen, da in der HAVING Clause für jeden Menschen der Wert True steht.

INTERESSANTE KEYWORDS

NOT IN

```
SELECT verlagname, ort FROM verlag
WHERE verlagname
          NOT IN (SELECT verlagname FROM buch);
```

Alle Verlagsnamen, die kein Buch in der Buch-Tabelle haben

IFNULL

Nimmt einen Wert entgegen und gibt diesen zurück. Falls der Wert NULL ist, wird ein default zurück gegeben.

Beispiel mit Menschen und Farben:

SELECT * FROM Menschen NATURAL LEFT OUTER JOIN Farben;

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün
Baum	Hans	17	20	4	NULL
Berg	Claudia	1	23	5	NULL

Ein paar Farben haben keien Bezeichnung in der Datenbank.

SELECT Name, Vorname, Lieblingsfarbe, Alter, Menschen.Id, IFNULL(Bezeichnung, ‚KENN ICH NICHT‘) FROM Menschen NATURAL LEFT OUTER JOIN Farben;

Name	Vorname	Lieblingsfarbe	Alter	Id	Bezeichnung
Müller	Heinz	1	23	1	Blau
Baum	Claudia	17	19	2	Gelb
NoName	Test	4	55	3	Grün
Baum	Hans	17	20	4	KENN ICH NICHT
Berg	Claudia	1	23	5	KENN ICH NICHT