Datenbanken und SQL



(Woche 3 - Tag 3)



Agenda

Aggregatfunktionen

- Definition + Motivation
- Ausgabe nicht-gruppierter Einzelwerte
 - > MIN
 - ➤ MAX
 - ➤ COUNT
 - ➤ SUM
 - AVG
 - Stolperfallen + "schlechter Stil"
- Gruppierung
 - Definition + Motivation
 - Beispiele
- HAVING-Klausel
 - Definition + Motivation
 - Beispiele



Aggregatfunktionen



Definition + Motivation

- "Aggregieren" bedeutet "anhäufen" oder auch "zusammenfügen". Genau dies leisten Aggregatfunktionen, die Werte (z.B. zu einer Summe) zusammenfassen.
- Im Gegensatz zu den meisten bisherigen SQL-Befehlen, bei denen eine Schleife gestartet wurde, um (unter Berücksichtigung etwaiger Bedingungen) die jeweils verlangte Funktionalität PRO Datensatz auszuführen, kann die Ausgabe bei Aggregatfunktion nicht für jeden einzelnen Datensatz geschehen, sondern gelingt erst **NACH Durchsicht** aller (anzusprechenden) Datensätze.
- Aggregatfunktionen sind mathematische Funktionen. Damit ist die Motivation für deren Einsatz aber offensichtlich, denn natürlich ist eine **mathematische Analyse** der in einer Datenbank gelisteten (Zahlen)-Werte oft von großer Bedeutung.
- Aggregatfunktionen können in Einzelfällen auch auf Nicht-Zahlen Typen angewendet werden. Möglichkeiten und Grenzen werden wir im Folgenden betrachten.



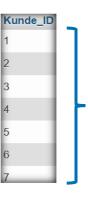
Nicht-gruppierte Einzelwerte



Aggregatfunktion MIN(...)

(Nicht-gruppierte) Aggregatfunktionen beziehen sich auf alle Datensätze, die <u>ohne</u> Aggregatfunktion ausgegeben worden wären. Hierzu ein erstes Beispiel an Hand der MIN-Funktion, die (natürlich) das **Minimum** ermittelt:

SELECT Kunde_ID **FROM** Kunde;



Dies sind die Datensätze, die (in diesem Fall) ausgegeben werden, wenn man auf den Einsatz einer Aggregatfunktion **verzichtet**.

SELECT MIN(Kunde_ID) **FROM** Kunde;



Dies ist das **Minimum** aller (oben dargestellten) Werte

Falls einem die Überschrift nicht gefällt:

SELECT MIN(Kunde_ID) **AS** "Kleinste Kunden-ID" **FROM** Kunde;





Aggregatfunktion MIN(...)

Für das Ergebnis einer Aggregatfunktion ist es (wie bei jeder mathematischen Funktion) von zentraler Bedeutung, auf welche Werte sich die jeweilige Aggregatfunktion bezieht. Daher ein weiteres Beispiel dazu:

SELECT Euro_Preis **FROM** Produkt **WHERE** Produkt_ID > 3;



Dies sind die Datensätze, die (in diesem Fall) ausgegeben werden, wenn man auf den Einsatz einer Aggregatfunktion **verzichtet**.

SELECT MIN(Euro_Preis) **FROM** Produkt **WHERE** Produkt_ID > 3;



Dies ist das Minimum aller (oben dargestellten) Werte

Keine Angst vor langen Überschriften ;-)

SELECT MIN(Euro_Preis) **AS** "Kleinster Preis aller Produkte mit ID > 3" **FROM** Produkt **WHERE** Produkt ID > 3;





MIN(...) – andere Typen

Wie schon bei ORDER BY können wir auch bei der Aggregatfunktion MIN vom "kleinsten" Datum (=ältestes Datum) oder auch von einem "kleinsten" Text (alphabetisch betrachtet "der erste") sprechen:

SELECT MIN(Datum) FROM Abrechnung;

MIN(Datum) 2021-05-05

SELECT MIN(Hersteller_Name) **FROM** Hersteller;

MIN(Hersteller_Name)
AntiByte



Aggregatfunktion MAX(...)

Die Aggregatfunktion **MAX** ermittelt (wie zu erwarten ©) das **Maximum** aller betrachteten Werte. Es gelten die entsprechenden Aussagen, die wir bereits bei der Funktion MIN kennenlernten. Wir betrachten 3 Beispiele:

SELECT MAX(Euro_Preis) **FROM** Produkt;

MAX(Euro_Preis) 1000.00

SELECT MAX(Datum) **FROM** Abrechnung;

MAX(Datum) 2022-02-14

SELECT MAX(Hersteller_Name) **FROM** Hersteller;

MAX(Hersteller_Name)
UltraBug

Bemerkung:



Aggregatfunktion MAX(...)

Die Aggregatfunktion **MAX** ermittelt (wie zu erwarten ©) das **Maximum** aller betrachteten Werte. Es gelten die entsprechenden Aussagen, die wir bereits bei der Funktion MIN kennenlernten. Wir betrachten 3 Beispiele:

SELECT MAX(Euro_Preis) **FROM** Produkt;

MAX(Euro_Preis) 1000.00

SELECT MAX(Datum) **FROM** Abrechnung;

MAX(Datum) 2022-02-14

SELECT MAX(Hersteller_Name) **FROM** Hersteller;



Bemerkung:

Die MIN- und MAX-Funktionen können vordergründig durch ORDER BY ... [DESC] LIMIT 1 ersetzt werden. Folgendes Beispiel zeigt aber, dass sie auch eine eigenständige Berechtigung haben:



Aggregatfunktion MAX(...)

Die Aggregatfunktion **MAX** ermittelt (wie zu erwarten ©) das **Maximum** aller betrachteten Werte. Es gelten die entsprechenden Aussagen, die wir bereits bei der Funktion MIN kennenlernten. Wir betrachten 3 Beispiele:

SELECT MAX(Euro_Preis) **FROM** Produkt;

MAX(Euro_Preis) 1000.00

SELECT MAX(Datum) **FROM** Abrechnung;

MAX(Datum) 2022-02-14

SELECT MAX(Hersteller_Name) **FROM** Hersteller;

MAX(Hersteller_Name)
UltraBug

Bemerkung:

Die MIN- und MAX-Funktionen können vordergründig durch ORDER BY ... [DESC] LIMIT 1 ersetzt werden. Folgendes Beispiel zeigt aber, dass sie auch eine eigenständige Berechtigung haben:

SELECT MIN(Datum**), MAX(**Datum**) FROM** Abrechnung;

MIN(Datum) MAX(Datum) 2021-05-05 2022-02-14



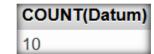
Aggregatfunktion COUNT(...)

Die Aggregatfunktion COUNT(Attribut) zählt alle Einträge des Attributs, deren Wert ungleich NULL ist.

SELECT COUNT(Abrechnung_ID) **FROM** Abrechnung;



SELECT COUNT(Datum) **FROM** Abrechnung;



Es erscheint das identische Ergebnis, da **kein** Datensatz in Abrechnung den Kalenderdatums-Wert **NULL** hat.

SELECT COUNT(*) FROM Abrechnung;



Alternativ kann man auch die Anzahl der Datensätze zählen.

Mit **COUNT(DISTINCT** ...) kann die Anzahl der <u>unterschiedlichen</u> Einträge (ungleich NULL) ermittelt werden.



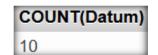
Aggregatfunktion COUNT(...)

Die Aggregatfunktion COUNT(Attribut) zählt alle Einträge des Attributs, deren Wert ungleich NULL ist.

SELECT COUNT(Abrechnung_ID) **FROM** Abrechnung;

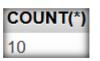


SELECT COUNT(Datum) **FROM** Abrechnung;



Es erscheint das identische Ergebnis, da **kein** Datensatz in Abrechnung den Kalenderdatums-Wert **NULL** hat.

SELECT COUNT(*) FROM Abrechnung;

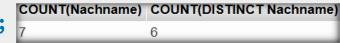


Alternativ kann man auch die Anzahl der Datensätze zählen.

Mit **COUNT(DISTINCT** ...) kann die Anzahl der <u>unterschiedlichen</u> Einträge (ungleich NULL) ermittelt werden.

Beispiel:

SELECT COUNT(Nachname), COUNT(DISTINCT Nachname) FROM Kunde;





Aggregatfunktion COUNT(...)

Die Aggregatfunktion COUNT(Attribut) zählt alle Einträge des Attributs, deren Wert ungleich NULL ist.

SELECT COUNT(Abrechnung_ID) **FROM** Abrechnung;

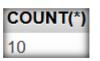


SELECT COUNT(Datum) **FROM** Abrechnung;



Es erscheint das identische Ergebnis, da **kein** Datensatz in Abrechnung den Kalenderdatums-Wert **NULL** hat.

SELECT COUNT(*) FROM Abrechnung;

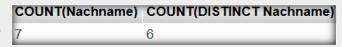


Alternativ kann man auch die Anzahl der Datensätze zählen.

Mit **COUNT(DISTINCT** ...) kann die Anzahl der <u>unterschiedlichen</u> Einträge (ungleich NULL) ermittelt werden.

Beispiel:

SELECT COUNT(Nachname), COUNT(DISTINCT Nachname) FROM Kunde;



(Es gibt 7 Kundennachnamen ungleich NULL, aber nur 6 unterschiedliche, da es zwei Kunden mit Nachnamen "Myrnow" gibt).



Die Aggregatfunktionen **SUM** und **AVG** ermitteln **Summe** und **Durchschnitt** (Average) der betrachteten Werte.

Durchschnittspreis aller von "Geld_her" angeboten Waren:

SELECT AVG(Euro_Preis) **FROM** Produkt;

AVG(Euro_Preis) 202.245000

Gesamtbestellsume (aller bisher bei "Geld_her" gekauften Waren):

SELECT SUM(Euro_Preis) **FROM** Abrechnung_Produkt, Produkt **WHERE** Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID;

SUM(Euro_Preis) 802.51



Die Aggregatfunktionen **SUM** und **AVG** ermitteln **Summe** und **Durchschnitt** (Average) der betrachteten Werte.

Durchschnittspreis aller von "Geld_her" angeboten Waren:

SELECT AVG(Euro_Preis) **FROM** Produkt;

AVG(Euro_Preis) 202.245000

Gesamtbestellsume (aller bisher bei "Geld_her" gekauften Waren):

SELECT SUM(Euro_Preis) **FROM** Abrechnung_Produkt, Produkt **WHERE** Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID;

SUM(Euro_Preis) 802.51

Der Einsatz von SUM oder AVG auf Attribute vom Typ DATE oder VARCHAR ist sinnlos, führt aber nicht zu einer Fehlermeldung:



Die Aggregatfunktionen **SUM** und **AVG** ermitteln **Summe** und **Durchschnitt** (Average) der betrachteten Werte.

Durchschnittspreis aller von "Geld_her" angeboten Waren:

SELECT AVG(Euro_Preis) **FROM** Produkt;

AVG(Euro_Preis) 202.245000

Gesamtbestellsume (aller bisher bei "Geld_her" gekauften Waren):

SELECT SUM(Euro_Preis) **FROM** Abrechnung_Produkt, Produkt **WHERE** Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID;

SUM(Euro_Preis) 802.51

Der Einsatz von SUM oder AVG auf Attribute vom Typ DATE oder VARCHAR ist sinnlos, führt aber nicht zu einer Fehlermeldung:

SELECT SUM(Datum), **AVG(**Datum) **FROM** Abrechnung;

SUM(Datum) AVG(Datum) 202119212 20211921.2000



Die Aggregatfunktionen **SUM** und **AVG** ermitteln **Summe** und **Durchschnitt** (Average) der betrachteten Werte.

Durchschnittspreis aller von "Geld_her" angeboten Waren:

SELECT AVG(Euro_Preis) **FROM** Produkt;



Gesamtbestellsume (aller bisher bei "Geld_her" gekauften Waren):

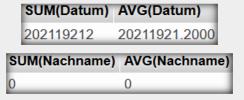
SELECT SUM(Euro_Preis) **FROM** Abrechnung_Produkt, Produkt **WHERE** Abrechnung_Produkt.Produkt_ID=Produkt.Produkt_ID;

```
SUM(Euro_Preis)
802.51
```

Der Einsatz von SUM oder AVG auf Attribute vom Typ DATE oder VARCHAR ist sinnlos, führt aber nicht zu einer Fehlermeldung:

SELECT SUM(Datum), **AVG(**Datum) **FROM** Abrechnung;

SELECT SUM(Nachname), **AVG(**Nachname) **FROM** Kunde;





- Sobald die Aufgabe darin besteht, **aggregierte** und **nicht-aggregierte** Werte **gemeinsam** auszugeben, sollte man "hellhörig" werden. Hier drohen Stolperfallen.
- Gelegentlich kann man sich zwar über diese Probleme hinwegsetzen, dies gilt aber in der Regel als **schlechter Stil**, oder ist in anderen Fällen sogar schlicht **falsch**.
- In beiden Fällen stellt dies ein Verweis auf **weiterführende Inhalte**, mit denen wir uns zum Teil bereits heute, oder aber in einigen Tagen befassen werden wir betrachten zunächst 2 Beispiele:

Name und Preis des <u>billigsten</u> Produkts:

SELECT Produkt_Name, MIN(Euro_Preis) FROM Produkt;

Name und Preis des <u>teuersten</u> Produkts:

SELECT Produkt_Name, MAX(Euro_Preis) FROM Produkt;

Produkt_Name	MIN(Euro_Preis)
tool 2.0	15.98

Produkt_Name	MAX(Euro_Preis)
tool 2.0	1000.00

Die erste Ausgabe ist nur **zufällig korrekt**, da die erste Entität der Tabelle Produkt zufälligerweise das billigste Produkt ist.

Dies zeigt sich zum einen an der zweiten Ausgabe, zum anderen kann die gesamte Fragestellung problematisiert werden: "Gibt es überhaupt **das eine** billigste Produkt?" (oder gibt es mehrere gleich-billige Produkte, die sich diesen "Titel" teilen?)

- Sobald die Aufgabe darin besteht, **aggregierte** und **nicht-aggregierte** Werte **gemeinsam** auszugeben, sollte man "hellhörig" werden. Hier drohen Stolperfallen.
- Gelegentlich kann man sich zwar über diese Probleme hinwegsetzen, dies gilt aber in der Regel als **schlechter Stil**, oder ist in anderen Fällen sogar schlicht **falsch**.
- In beiden Fällen stellt dies ein Verweis auf **weiterführende Inhalte**, mit denen wir uns zum Teil bereits heute, oder aber in einigen Tagen befassen werden wir betrachten zunächst 2 Beispiele:

Name und Preis des billigsten Produkts:

SELECT Produkt_Name, MIN(Euro_Preis) FROM Produkt;

Name und Preis des <u>teuersten</u> Produkts:

SELECT Produkt_Name, MAX(Euro_Preis) FROM Produkt;

Produkt_Name	MIN(Euro_Preis)
tool 2.0	15.98

Produkt_Name	MAX(Euro_Preis)
tool 2.0	1000.00

Die erste Ausgabe ist nur **zufällig korrekt**, da die erste Entität der Tabelle Produkt zufälligerweise das billigste Produkt ist.

Dies zeigt sich zum einen an der zweiten Ausgabe, zum anderen kann die gesamte Fragestellung problematisiert werden: "Gibt es überhaupt **das eine** billigste Produkt?" (oder gibt es mehrere gleich-billige Produkte, die sich diesen "Titel" teilen?)

Erläuterung:

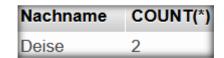
Wir sehen hier ein schönes Beispiel für den "Widerstreit" zweier Prinzipien. Die interne Schleife startet, gibt den ersten gefundenen Produktnamen aus, und muss dann abbrechen, da die Aggregierte Funktion erst am Ende der Schleife einen Wert ausgeben kann. Wir werden in einigen Tagen eine korrekte Lösung dieser Aufgabenstellung kennenlernen.

Nachname und Anzahl der Abrechnungen von Kunde 2:

SELECT Nachname, **COUNT(*) FROM** Kunde, Abrechnung

WHERE Kunde.Kunde_ID=2

AND Kunde_ID=Abrechnung.Kunde_ID;



Diese Lösung ist zwar korrekt, gilt aber (zumindest aus strenger Sicht) als "schlechter Stil".

Auch hier wird nämlich einfach der erste gefundene Nachname ausgegeben, der in diesem Fall aber auf Grund der Bedingung **Kunde_ID=2** identisch zu den Namen aller anderen Datensätze ist.

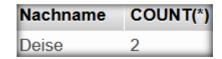


Nachname und Anzahl der Abrechnungen von Kunde 2:

SELECT Nachname, **COUNT(*) FROM** Kunde, Abrechnung

WHERE Kunde.Kunde_ID=2

AND Kunde_ID=Abrechnung.Kunde_ID;



Diese Lösung ist zwar korrekt, gilt aber (zumindest aus strenger Sicht) als "schlechter Stil".

Auch hier wird nämlich einfach der erste gefundene Nachname ausgegeben, der in diesem Fall aber auf Grund der Bedingung **Kunde_Kunde_ID=2** identisch zu den Namen aller anderen Datensätze ist.

Es ist daher "kein Zufall", dass der Nachname (Deise) hier tatsächlich zu dem aggregierten Wert (2) gehört. Wir wollen aber "kombinierte Ausgaben" von aggregierten und nicht-aggregierten Werten zukünftig mittels "Gruppierungen" umsetzen.

Dies wird Gegenstand der folgenden Folien sein. Bleiben Sie gespannt ;-)



Gruppierung



Definition + Motivation

- Wir haben bisher Aggregat-Funktionen genutzt, um einzelne Werte zu ermitteln.
- Entsprechend waren daher die Ausgaben auch jeweils nur 1-zeilig.
- Dies lag daran, dass wir bei den bisherigen Abfragen alle betrachteten Datensätze

 anschaulich gesprochen "in einen Topf warfen" und von diesen dann "die"
 Summe (oder "den" Durchschnittswert, "das" Minimum … etc.) berechnen ließen.
- Im Folgenden wollen wir uns hierzu eine Variante anschauen, bei der wir die zu berücksichtigenden Datensätze "gruppieren, um daraufhin die zu ermittelnden Aggregatwerte **pro Gruppe** berechnen zu lassen.
- Auf diese Weise werden wir die ermittelten Ergebnisse pro Gruppe vergleichen (oder gegebenenfalls auch sortieren) können und jedenfalls in die Lage versetzt, deutlich **interessantere Abfragen** zu formulieren.



Vorname, Nachname und Anzahl der Abrechnungen PRO Kunde (repräsentiert durch Vor- und Nachname):

SELECT Vorname, Nachname, **COUNT(*) FROM** Kunde, Abrechnung **WHERE** Kunde.Kunde_ID=Abrechnung.Kunde_ID **GROUP BY** Vorname, Nachname;

Vorname	Nachname	COUNT(*)
Elli	Rot	2
Eva	Hahn	1
Peter	Kaufnix	1
Rita	Myrnow	1
Vera	Deise	2
Witali	Myrnow	3

Vorname, Nachname und Anzahl der Abrechnungen **PRO Kunde** (repräsentiert durch Vor- und Nachname): (sortiert nach Anzahl der Abrechnungen absteigend)

SELECT Vorname, Nachname, COUNT(*) FROM Kunde, Abrechnung
WHERE Kunde.Kunde_ID=Abrechnung.Kunde_ID
GROUP BY Vorname, Nachname
ORDER BY COUNT(*) DESC;

Vorname	Nachname	COUNT(*)
Witali	Myrnow	3
Elli	Rot	2
Vera	Deise	2
Rita	Myrnow	1
Eva	Hahn	1
Peter	Kaufnix	1

Die **Repräsentation eines Kunden** mittels Vor- und Nachnamen ist nicht unproblematisch, da es unterschiedliche, aber gleichnamige Kunden geben könnte, deren Abrechnungen dann fälschlicherweise "in einen (gemeinsamen) Topf" geworfen werden würden.

Nachname und Anzahl der Abrechnungen PRO Kunde (repräsentiert durch Vor- und Nachname):

SELECT Vorname, Nachname, **COUNT(*) FROM** Kunde, Abrechnung **WHERE** Kunde.Kunde_ID=Abrechnung.Kunde_ID **GROUP BY** Vorname, Nachname;

Vorname	Nachname	COUNT(*)
Elli	Rot	2
Eva	Hahn	1
Peter	Kaufnix	1
Rita	Myrnow	1
Vera	Deise	2
Witali	Myrnow	3

Nachname und Anzahl der Abrechnungen **PRO Kunde** (repräsentiert durch Vor- und Nachname): (sortiert nach Anzahl der Abrechnungen absteigend)

SELECT Vorname, Nachname, COUNT(*) FROM Kunde, Abrechnung
WHERE Kunde.Kunde_ID=Abrechnung.Kunde_ID
GROUP BY Vorname, Nachname
ORDER BY COUNT(*) DESC;

Vorname	Nachname	COUNT(*
Witali	Myrnow	3
Elli	Rot	2
Vera	Deise	2
Rita	Myrnow	1
Eva	Hahn	1
Peter	Kaufnix	1

Die **Repräsentation eines Kunden** mittels Vor- und Nachnamen ist nicht unproblematisch, da es unterschiedliche, aber gleichnamige Kunden geben könnte, deren Abrechnungen dann fälschlicherweise "in einen (gemeinsamen) Topf" geworfen werden würden.

Im obigen Fall wäre es also günstiger:

- a) die Kunden-ID ausgeben zu lassen
- b) nach der Kunden-ID zu gruppieren

Nachname und Anzahl der Abrechnungen PRO Kunde (repräsentiert durch Vor- und Nachname):

SELECT Vorname, Nachname, **COUNT(*) FROM** Kunde, Abrechnung WHERE Kunde.Kunde_ID=Abrechnung.Kunde_ID
GROUP BY Vorname, Nachname;

Vorname	Nachname	COUNT(*)
Elli	Rot	2
Eva	Hahn	1
Peter	Kaufnix	1
Rita	Myrnow	1
Vera	Deise	2
Witali	Myrnow	3

Nachname und Anzahl der Abrechnungen **PRO Kunde** (repräsentiert durch Vor- und Nachname): (sortiert nach Anzahl der Abrechnungen absteigend)

SELECT Vorname, Nachname, COUNT(*) FROM Kunde, Abrechnung
WHERE Kunde.Kunde_ID=Abrechnung.Kunde_ID
GROUP BY Vorname, Nachname
ORDER BY COUNT(*) DESC;

	N. I	OOLINIT(*)
Vorname	Nachname	COUNT(*)
Witali	Myrnow	3
Elli	Rot	2
Vera	Deise	2
Rita	Myrnow	1
Eva	Hahn	1
Peter	Kaufnix	1

Die **Repräsentation eines Kunden** mittels Vor- und Nachnamen ist nicht unproblematisch, da es unterschiedliche, aber gleichnamige Kunden geben könnte, deren Abrechnungen dann fälschlicherweise "in einen (gemeinsamen) Topf" geworfen werden würden.

Im obigen Fall wäre es also günstiger:

- a) die Kunden-ID ausgeben zu lassen
- b) nach der Kunden-ID zu gruppieren

Dies ist aber dann bereits Teil der Aufgabenstellung und muss bzgl. einer IHK-Aufgabe nicht berücksichtigt werden. So kann für die IHK die folgende Empfehlung ausgesprochen werden: Notieren Sie hinter GROUP BY einfach alle Attribute, die auch hinter SELECT notiert wurden, MIT AUSNAHME der Attribute aller aufgeführten Aggregatfunktionen.

HAVING



Definition + Motivation

- Wie schon WHERE und ON ist auch HAVING eine "Bedingungs-Klausel".
- Sie ist **obligatorisch**, wenn die Bedingung von einem **aggregierten Wert** spricht.
- Auch diese Funktionalität wird uns erlauben, interessantere Abfragen zu formulieren. Dies werden wir im Folgenden durch einige Beispiele illustrieren.



PRO Hersteller: Hersteller-Name und Preis seines teuersten Produkts (im Sortiment von "Geld_her"). (Es sollen aber nur Hersteller berücksichtigt werden, deren **teuerstes Produkt mehr als 30 Euro** kostet.)

SELECT Hersteller_Name, **MAX**(Euro_Preis) **FROM** Produkt, Hersteller **WHERE** Produkt.Hersteller_ID=Hersteller.Hersteller_ID **GROUP BY** Hersteller_Name **HAVING MAX**(Euro_Preis) > 30;

Hersteller_Name	MAX(Euro_Preis)
Contrabit	45.05
Ladenhut AG	1000.00
UltraBug	98.00

PRO Hersteller: Hersteller-Name und Preis seines teuersten Produkts (im Sortiment von "Geld_her"). (Es sollen aber nur Hersteller berücksichtigt werden, deren **Produkte im Durchschnitt weniger als 500 Euro** kosten.)

SELECT Hersteller_Name, **MAX**(Euro_Preis) **FROM** Produkt, Hersteller **WHERE** Produkt.Hersteller_ID=Hersteller.Hersteller_ID **GROUP BY** Hersteller_Name **HAVING AVG**(Euro_Preis) < 500;

Hersteller_Name	MAX(Euro_Preis)
AntiByte	22.75
Contrabit	45.05
UltraBug	98.00



Gemeinsame Übung ("Live-Coding") -> A_03_03_01



Aufgabe_03_03_01

Formulieren Sie bitte entsprechende SQL-Anweisungen für folgende Aufgabestellungen:

- a) Ausgabe der kleinsten und größten Speditions-ID, sowie der Anzahl der Speditionen (bzw. die Anzahl der Speditions-IDs, die nach Definition ja alle ungleich NULL sind).
- b) Durchschnittlicher Preis aller bisher verkauften Produkte. Ausgabe unter der Überschrift "Durchschnittspreis".
- c) Pro Abrechnung: Kalenderdatum und Gesamtbestellsumme. Sortiert nach Gesamtbestellsumme abfallend.
- d) Pro Kunde: Kunden-ID, Nachname und Anzahl der von ihm bestellten Produkte. Ausgabe nach 1.) Anzahl abfallend und 2.) Nachname aufsteigend sortiert.
- e) Pro Hersteller: Herstellername und Anzahl der Produkte im Sortiment von "Geld_her". Ausgabe sortiert nach Anzahl abfallend. Es sollen aber nur Hersteller berücksichtigt werden, die mindestens 1 Produkt im Sortiment haben.
- f) Pro Produkt: Produktname und Anzahl der bestellten Exemplare.
 Ausgabe sortiert nach Anzahl abfallend, begrenzt auf 3. (Es sollen aber nur Produkte berücksichtigt werden, die mindestens 1-mal bestellt wurden.)

WBS TRAINING AG Lorenzweg 5 D-12099 Berlin Amtsgericht Berlin HRB 68531 Sitz der Gesellschaft: Berlin

Vorstand: Heinrich Kronbichler, Joachim Giese Aufsichtsrat (Vorsitz): Dr. Daniel Stadler USt-IdNr.: DE 209 768 248

GL5 Gemeinschaftsbank eG IBAN: DE18 4306 0967 1146 1814 00 BIC: GENODEM1GL5





Vielen Dank für Ihre Aufmerksamkeit!



