	Subselect		AEuP	V 1.0
	Name	Klasse	Datum	

1 Ergebnistypen

Für die folgenden Überlegungen gehen wir von diesen Datenstrukturen aus:

Kunde

KID	VorN	NachN
10	Peter	Kurz
21	Rita	Müller
34	Hans	Lang

Bestellung

BID	Datum	KID
31	2021-02-14	21
21	2021-04-21	10
34	2021-09-04	10

Bestellpos

ZID	BID	PID	Anz
102	31	67	1
103	21	19	2
104	21	67	1
105	34	67	2

Produkt

PID	Bez	Preis
19	SenseSound	349.00
67	Mobil Blu	229.00
99	SlimSound	187.00

Selectergebnisse kann man grob in drei Gruppen unterteilen:

Ergebnistyp **Wert**:

Select:	**Ergebnis:**	**Erklärung:**

`SELECT COUNT(*)`
`FROM Kunde k;`
	`+----------+`
	`| COUNT(*) |`
	`+----------+`
	`| 3 |`
	`+----------+`
		Ein einzelner **Wert**.

Ergebnistyp **Liste**:

Select:	**Ergebnis:**	**Erklärung:**

`SELECT KID FROM`
`Bestellung b;`
	`+------+`
	`| KID |`
	`+------+`
	`| 10 |`
	`| 21 |`
	`| 10 |`
	`+------+`
		Eine **Liste** von gleichen Werten.

Ergebnistyp **Matrix**:

Select:	**Ergebnis:**	**Erklärung:**

`SELECT k.NachN,`
`b.Datum FROM`
`Bestellung b`
`JOIN Kunde k ON`
`k.KID = b.KID;`
	`+--------+------------+`
	`| NachN | Datum |`
	`+--------+------------+`
	`| Kurz | 2021-04-21 |`
	`| Kurz | 2021-09-04 |`
	`| Müller | 2021-02-14 |`
	`+--------+------------+`
		Eine **Matrix** von Werten.

2 Arten von „Eingangsdaten“ eines Select Statements

Ein Select Statement benötigt auch Daten, um das Ergebnis zu erzeugen. Sehen wir uns hierfür folgendes Statement an:

```
SELECT * FROM Bestellpos bp
WHERE ANZ = 1
  AND BID IN (31, 21);
```

Wir finden hier ebenfalls die Strukturen **Wert** (die „1“ bei `ANZ = 1`), die **Liste** („31, 21“ in `IN (31, 21)`) und die **Matrix** (das ist die Tabelle „Bestellpos“).

3 Verknüpfung im Subselect

Beginnen wir mit Typ „**Wert**“. Ziel soll es sein, die Daten des günstigsten Produktes (oder der günstigsten Produkte) zu ermitteln. Dies können wir nun erledigen, indem wir das SELECT Ergebnis als Eingangswert eines zweiten SELECT Statements wiederverwenden:

1. Schritt: **Wert** ermitteln:

```
SELECT MIN(Preis) FROM Produkt p;
```

2. Schritt: **Wert** als Eingangsgröße in das zweite SELECT Statement einsetzen:

Das Ergebnis „187.00“ (also der *günstigste Preis*) wird nun als Eingangsgröße in das zweite SELECT übertragen:

```
SELECT * FROM Produkt WHERE Preis = 187.00;
```

Dadurch erhalten wir Produktdaten der *günstigsten Produkte*. Nun verknüpfen wir die beiden Statements in eines, wobei wir die Ermittlung des günstigsten Preiswertes als Subselect (dt. „Unterabfrage“) formulieren. Hierzu müssen wir es in **runde Klammern** setzen:

```
SELECT * FROM Produkt
WHERE Preis = (
      SELECT MIN(Preis) FROM Produkt p
)
```

Dies ist die Standardabfrage, um die Daten für das *günstigste Produkt* in einem einzigen Statement zu ermitteln.

Sehen wir uns nun Statements mit Typ „Liste“ an – wieder zuerst in zwei Schritten, danach in einem. Hierzu wollen wir die Namen der Kunden ermitteln, welche etwas bestellt haben[1]:

1. Schritt: **Liste** ermitteln:

```
SELECT DISTINCT KID FROM Bestellung b;
```

2. Schritt: **Liste** als Eingangsgröße in das zweite SELECT Statement einsetzen:

Das Ergebnis „10, 21“ (also die *KundenIDs der Kunden mit einer Bestellung*) wird nun als Eingangsgröße in das zweite SELECT übertragen:

```
SELECT VorN, NachN FROM Kunde k WHERE KID IN (10, 21);
```

Dadurch erhalten wir die Vor- und Nachnamen der *KundenIDs mit einer Bestellung*. Nun verknüpfen wir die beiden Statements wieder in eines und erhalten wieder das Ergebnis in nur einem Statement:

```
SELECT VorN, NachN FROM Kunde k
WHERE KID IN (SELECT DISTINCT KID FROM Bestellung b);
```

Zum Schluss sehen wir uns ein Beispiel für den Ergebnistyp **Matrix** an. Wir wollen gerne pro Bestellung wissen, wie viele Produkttypen und wie viele Produkte gekauft wurden – allerdings nur für Bestellungen, bei denen mehrere Produkte gekauft wurden. Anschließend wollen wir die Ergebnismenge mit den Bestelldatumswerten anreichern. Da wir eine Matrix nur in Tabellenform kennen, ist es nicht möglich dieses Select manuell in zwei Schritte zu Teilen, jedoch können wir uns zur Orientierung die Matrix vorher ansehen.

Matrix ermitteln:

```
SELECT BID, SUM(Anz) AnzItem, COUNT(*) AnzProd FROM BestellPos
GROUP BY BID HAVING AnzItem > 1;
```

Das Ergebnis sieht dann wie folgt aus:

[1] Dies geht zwar mit Joins performanter, aber das Prinzip des Subselects vom Typ „Liste“ ist durch dieses Beispiel am einfachsten nachvollziehbar

```
+------+---------+---------+
| BID  | AnzItem | AnzProd |
+------+---------+---------+
|   21 |       3 |       2 |
|   34 |       2 |       1 |
+------+---------+---------+
```

Nun gehen wir davon aus, dass diese Datenmenge als Tabelle existiert. Somit müssen wir zwei Dinge beachten:

- Die Spaltennamen in dieser virtuellen Tabelle müssen eindeutig sein
- Die virtuelle Tabelle benötigt einen Namen

Sehen wir uns also an, wie wir das vorausgegangene Statement als virtuelle Tabelle in einem zweiten Statement behandeln:

```
SELECT virtTab.*, b.Datum FROM (
      SELECT BID, SUM(Anz) AnzItem, COUNT(*) AnzProd
      FROM BestellPos GROUP BY BID HAVING AnzItem > 1
) virtTab
JOIN Bestellung b ON b.BID = virtTab.BID;
```

Wir joinen also die Bestellungstabelle zu der virtuellen Tabelle hinzu, um pro Eintrag das Bestelldatum hinzuzufügen.

4 Sonderformen des Subselects

Bis jetzt wurden die Subselect Statements immer nach dem „FROM“ des äußeren Statements platziert. Es ist jedoch auch möglich, es zwischen „SELECT“ und „FROM“ einzutragen, wodurch eine neue Funktionsweise entsteht. Sehen wir uns hierfür folgendes Statement an:

```
SELECT b.Datum,
      (SELECT SUM(bp.Anz * p.Preis) FROM Bestellpos bp
      JOIN Produkt p ON p.PID = bp.PID
      WHERE bp.BID = b.BID)
FROM Bestellung b;
```

Hier lesen wir das Statement von außen nach innen. Außen wird ein Select auf der Bestellung Tabelle durchgeführt. Das heißt, dass MySQL Datensatz für Datensatz der Bestellungstabelle abarbeitet. Somit wird *für jeden Datensatz* das Subselect durchgeführt. Da wir dort aber über das WHERE Statement auf b.BID (also der BestellID des aktuellen Datensatzes einschränken, erhalten wir somit Pro Datensatz den summierten Preis. Gehen wir die drei Datensätze der Bestellungstabelle durch.

1. Datensatz. BID = „31“. Dadurch wird das innere Select wie folgt ausgeführt:

```
SELECT SUM(bp.Anz * p.Preis) FROM Bestellpos bp
JOIN Produkt p ON p.PID = 31.
```
(das Ergebnis ist hier 229.00)

2. Datensatz. BID = „21“. Dadurch wird das innere Select wie folgt ausgeführt:

```
SELECT SUM(bp.Anz * p.Preis) FROM Bestellpos bp
JOIN Produkt p ON p.PID = 21.
```
(das Ergebnis ist hier 927.00)

3. Datensatz. BID = „34“. Dadurch wird das innere Select wie folgt ausgeführt:

```
SELECT SUM(bp.Anz * p.Preis) FROM Bestellpos bp
JOIN Produkt p ON p.PID = 34.
```
(das Ergebnis ist hier 458.00)

Die Ergebniswerte werden dann einfach in das äußere Select Statement eingefügt, wodurch wir folgendes Ergebnis erhalten:

```
+------------+--------+
| Datum      | Betrag |
+------------+--------+
| 2021-02-14 | 229.00 |
| 2021-04-21 | 927.00 |
| 2021-09-04 | 458.00 |
+------------+--------+
```

Eine Lösung mit einem JOIN ist dieser Lösung aber vorzuziehen, da es in der Regel performanter ist. Trotzdem ist das Verständnis wichtig, da dieses Konzept auch für UPDATE Statements aus SELECT (vor allem in Prüfungen) genutzt wird.

Gehen wir hierfür davon aus, dass die Tabelle „Bestellung“ eine neue Spalte namens „Betrag“ erhält und wir die oben berechneten Betragswerte über ein UPDATE in dieses neue Feld eintragen wollen. Hierzu verwenden wir das gleiche Statement wie oben und ändern es in ein UPDATE ab:

```
UPDATE Bestellung b SET Betrag =
      (SELECT SUM(bp.Anz * p.Preis) FROM Bestellpos bp
      JOIN Produkt p ON p.PID = bp.PID
      WHERE bp.BID = b.BID);
```

Genau wie beim SELECT Statement wird nun Datensatz für Datensatz der Bestellung Tabelle abgearbeitet und pro Datensatz das UPDATE durchgeführt.