Übungen Advanced SQL

Klaus-Georg Deck

2025-07-24

Einleitung

In diesem Skript findest du typische SQL-Übungen mit Beispielcode und den erwarteten Ergebnistabellen. Ziel ist es, fortgeschrittene SQL-Techniken wie Outer Join, Group-by, Aggregationen und einfache Subselects zu erlernen und vertiefen.

Voraussetzungen

Du verfügst über einen Zugang zu einem Datenbank-Server (Oracle oder Postgres) mit den Tabellen des Bike-Verleih-Szenarios und hast Grundkenntnisse in SQL, etwa indem Du die vorherigen Übungen erfolgreich absolviert hast.

Aufgabe 1: Verknüpfung von Tabellen

a) Einfache OUTER-Join Anweisung

Für diesen Übungsteil benötigst Du die Tabellen `EMPLO` und `SHOP`. Gib eine SQL-Anweisung an, die den Namen der Mitarbeitenden zusammen mit dem Namen des Shops, dem sie zugeordnet sind, anzeigt. Es sollen alle Mitarbeitenden angezeigt werden, auch jene, die keinem Shop angehören. In diesem Fall soll der Wert des Shop-Namens leer sein. Sortiere die Datensätze nach `ID` aus `EMPLO` absteigend.

Die Resultatmenge hat folgende Gestalt:

ID	ENAME	SNAME
16055	Leon	
16051	Bea	
16050	Felix	Vaduz CyclePoint
16049	Anna	Bern Rollt
…	…	…

Lösung:

```
SELECT E.ID, E.NAME ENAME, S.NAME SNAME
FROM EMPLO E
LEFT OUTER JOIN SHOP S ON E.SHOP = S.ID
ORDER BY E.ID DESC
```

b) Vergleich zwischen OUTER- und INNER-Join

Ersetzst Du in der Lösung der vorherigen Teilaufgabe `LEFT OUTER JOIN` durch `JOIN`, dann sind die Resultate unterschiedlich. Zeige dies, indem Du in beiden Fällen die Anzahl der Datensätze ermittelst. Was kann man grundsätzlich über die Anzahl der Datensätze in solchen Fällen aussagen? Welche Datensätze kommen in welchem Resultat nicht vor?

Verwende anschliessend als Verknüpfung `FULL OUTER JOIN`. Wie viele Datensätze erhält man dann?

Verwende einen Teil der Lösung der vorherigen Teilaufgabe:

```
SELECT E.ID, E.NAME ENAME, S.NAME SNAME
FROM EMPLO E
LEFT OUTER JOIN SHOP S ON E.SHOP = S.ID
```

Lösung:

Die Anzahl der Datensätze beim `LEFT OUTER JOIN` ist `24`, beim `INNER JOIN` ist sie `22` und beim `FULL OUTER JOIN` ist sie '25'.

```
SELECT COUNT(*)
FROM EMPLO E
LEFT OUTER JOIN SHOP S ON E.SHOP = S.ID;

SELECT COUNT(*)
FROM EMPLO E
INNER JOIN SHOP S ON E.SHOP = S.ID;

SELECT COUNT(*)
FROM EMPLO E
FULL OUTER JOIN SHOP S ON E.SHOP = S.ID;
```

Wie Du Dich leicht überzeugen kannst, gibt es genau 22 Mitarbeitende, die einem Shop zugeordnet sind, 2 Mitarbeitende, die keinem Shop zugeordnet sind und genau ein Shop, dem kein Mitarbeitender zugeordnet ist.

Die Datensätze des `INNER`-Join sind in der Resultatmenge des `LEFT OUTER`-Join enthalten, der mehr Datensätze haben kann, nämlich die Datensätze der ersten (linken) Tabelle, für die die `ON`-Bedingung nicht zutrifft. Bei einem `RIGHT OUTER`-Join kommen möglicherweise Datensätze der zweiten (rechten) Tabelle hinzu. Betrachtet man die Anzahl der Datensätze, so ist die Anzahl eines `INNER`-Join kleiner oder gleich der Anzahl des entsprechenden `LEFT OUTER`-Join sowie des `RIGHT OUTER`-Join und die Anzahl der Datensätze des `FULL`-Join ist grösser oder gleich der Anzahl des `LEFT OUTER`- resp. `RIGHT OUTER`-Join. Der `FULL OUTER`-Join enthält die Datensätze des `INNER JOIN` sowie die der linken und der rechten Tabelle, für die die `ON`-Bedingung nicht zutrifft.

c) Join: INNER, LEFT, RIGHT, FULL und OUTER

Begründe, warum ein `LEFT INNER`- oder `RIGHT INNER`-Join keinen Sinn macht und warum bei `LEFT OUTER`-, `RIGHT OUTER`- und `FULL OUTER`-Join das Schlüsselwort `OUTER` weggelassen werden kann. Erstelle eine Übersichtstabelle, in der die jeweilige Lang- und Kurzform der verschiedenen Joins aufgelistet sind.

Lösung:

Die Resultatmenge eines `INNER`-Join ist von der Reihenfolge der Verknüpfung unabhängig, d.h. die Anweisung `SELECT ... FROM A JOIN B` ergibt bei gleicher `ON`-Bedingung das gleiche Resultat

wie `SELECT ... FROM B JOIN A` (die Reihenfolge der Attribute oder der Datensätze kann variieren). Daher macht die Verwendung von `LEFT` oder `RIGHT` hier keinen Sinn.

Die folgende Tabelle stellt die verschiedenen Verknüpfungen mit ihrer Kurzform dar:

Verknüpfung	Kurzform	Bemerkung
`INNER JOIN`	`JOIN`	symmetrisch
`LEFT OUTER JOIN`	`LEFT JOIN`	
`RIGHT OUTER JOIN`	`RIGHT JOIN`	
`FULL OUTER JOIN`	`FULL JOIN`	symmetrisch
`CROSS JOIN`	, (Komma)	keine `ON`-Bedingung

Aus jeder Kurzform ergibt sich eindeutig die Verknüpfung, da `LEFT` resp. `RIGHT` nur bei den nicht symmetrischen `OUTER`-Verknüpfungen Sinn macht, und ein `JOIN` allein, also ohne `LEFT`, `RIGHT` oder `FULL` dann zwangsläufig ein `INNER JOIN` ist.

Aufgabe 2: Verknüpfungen und `EXISTS`

Für diese Aufgabe benötigst Du die Tabellen `CUSTOMER`, `RENTAL` und `BIKE`, die Informationen über Bike-Ausleihen von Kunden enthalten.

a) Einfache Verknüpfung

Welche Kunden haben am 2. Januar 2025 oder früher Bikes ausgeliehen? Gib die Ausleih-ID, den Namen des Kunden und das Ausleihdatum, sortiert nach Ausleih-ID an. Das erwartete Resultat:

R_ID	NAME	DATUM
60023	Emma	2025-01-01
60046	Lena	2025-01-02
60078	Francesco	2025-01-01

Lösung:

```
SELECT R.ID R_ID, C.NAME, R.RENTAL_DATE DATUM
FROM RENTAL R
JOIN CUSTOMER C ON R.CUSTOMER = C.ID
WHERE RENTAL_DATE <= DATE '2025-01-02'
ORDER BY R.ID
```

b) Kunden mit und ohne Ausleihen

Welche Kunden haben bisher (irgendwann einmal) Bikes ausgeliehen?

Welche Kunden haben bisher noch keine Bikes ausgeliehen?

Gib jeweils die Kunden-ID und den Namen der Person an, aufsteigend sortiert nach Kunden-ID.

Tipp:
Verwende für den ersten Teil die vorherige Teilaufgabe. Beachte, dass Kunden nicht mehrfach genannt werden sollen. Für den zweiten Teil kann man einen `LEFT JOIN` verwenden und diejenigen Datensätze aus dem Resultat herausfiltern, für die die `ON`-Bedingung nicht zutrifft. (Alternativ kann man auch einen Subselect mit der SQL-Klausel `EXISTS` resp. `NOT EXISTS` verwenden.)

Lösung:

Kunden, die Bikes ausgeliehen haben:

```
SELECT DISTINCT C.ID, C.NAME
FROM RENTAL R
JOIN CUSTOMER C ON R.CUSTOMER = C.ID
ORDER BY C.ID;

--mit EXISTS:
SELECT C.ID, C.NAME FROM
CUSTOMER C
WHERE EXISTS( SELECT 1 FROM RENTAL R WHERE R.CUSTOMER = C.ID )
ORDER BY C.ID;
```

Kunden, die bisher noch keine Bikes ausgeliehen haben:

```
SELECT C.ID, C.NAME
FROM CUSTOMER C
LEFT JOIN RENTAL R ON R.CUSTOMER = C.ID
WHERE R.CUSTOMER IS  NULL
ORDER BY C.ID;

--mit EXISTS:
SELECT C.ID, C.NAME FROM
CUSTOMER C
WHERE NOT EXISTS( SELECT 1 FROM RENTAL R WHERE R.CUSTOMER = C.ID )
ORDER BY C.ID;
```

Der Trick bei der ersten Anweisung besteht darin, diejenigen Datensätze nach dem `LEFT JOIN` herauszufiltern, die mit `NULL` ergänzt wurden, d. h. die, für die es keine wahre `ON`-Bedingung gibt, also keine Ausleihen.

c) Anzahl der Kunden

Wie viele Kunden gibt es insgesamt, wie viele haben ein Bike des Modells ‘Pathway’ ausgeliehen und wie viele haben kein solches Modell ausgeliehen.

Lösung:

Insgesamt gibt es `167` Kunden, von denen `14` ein Bike eines solchen Modells ausgeliehen haben und `153` nicht.

```
-- Alle Kunden
SELECT COUNT(*) AS CT FROM CUSTOMER;

-- Kunden die ein 'Pathway' ausgeliehen haben
SELECT COUNT(*) AS CT_PW FROM CUSTOMER C
WHERE EXISTS(
   SELECT 1 FROM RENTAL R
   JOIN BIKE B ON R.BIKE = B.ID
   WHERE B.MODELL = 'Pathway'
      AND C.ID = R.CUSTOMER
);

-- Kunden, die kein 'Pathway' ausgeliehen haben
--    wie zuvor, anstelle 'EXISTS' verwende 'NOT EXISTS'
SELECT COUNT(*) AS CT_NO_PW FROM CUSTOMER C
WHERE NOT EXISTS(
```

```
   SELECT 1 FROM RENTAL R
   JOIN BIKE B ON R.BIKE = B.ID
   WHERE B.MODELL = 'Pathway'
      AND C.ID = R.CUSTOMER
);
```

In einer einzigen Anweisung (hier gibt es unterschiedliche Varianten):

```
SELECT
   COUNT(CUST.ID) AS CT,
   COUNT(TAB.ID) AS CT_PW,
   COUNT(CUST.ID)-COUNT(TAB.ID) AS CT_NOPW
FROM CUSTOMER CUST
LEFT JOIN
(SELECT C.ID FROM CUSTOMER C
 WHERE EXISTS (
   SELECT 1
   FROM RENTAL R
   JOIN BIKE B ON R.BIKE = B.ID
   WHERE B.MODELL = 'Pathway'
      AND C.ID = R.CUSTOMER )
) TAB ON TAB.ID = CUST.ID
```

Aufgabe 3: Gruppierung und Aggregatsfunktionen

a) Average: Gehalt und Tage im Unternehmen

Gib eine SQL-Anweisung an, die das Durchschnitts-Gehalt und die Durchschnitts-Zeit im Unternehmen aller Mitarbeitenden ausgibt. Das Gehalt soll gerundet auf ganze Werte, die Zeit im Unternehmen gerundet als ganze Tage ausgegeben werden.
Zusätzlich soll angegeben werden, wie viele Mitarbeitenden es insgesamt gibt und wie viele jeweils in die Gehalts- resp. Zeit-Berechnung einfliessen. Beachte, dass fehlende Werte (`NULL`) bei den meisten Aggregatsfunktionen ignoriert werden.

Das Ergebnis sieht wie folgt aus:

CT_ALL	AVG_SAL	CT_SAL	AVG_DAYS	CT_DAYS
24	5'457	23	1'119	21

Lösung:

```
SELECT
   COUNT(*) CT_ALL,
   ROUND( AVG( SALARY ) ) AVG_SAL,
   COUNT( SALARY ) CT_SAL,
   ROUND( AVG( TRUNC(CURRENT_DATE - HIRE_DATE )) ) AVG_DAYS ,
   COUNT( HIRE_DATE) CT_DAYS
FROM EMPLO
```

Der Ausdruck `AVG( TRUNC(CURRENT_DATE - HIRE_DATE ))` kann wie folgt vereinfacht werden:

Postgres:	`AVG( CURRENT_DATE - HIRE_DATE )`
Oracle:	`AVG( TRUNC(CURRENT_DATE) - HIRE_DATE )`

b) Average: Gehalt und Tage im Unternehmen pro Shop

Ermittle die gleichen Kennzahlen wie in der vorherigen Teilaufgabe, jedoch nicht für das gesamte Unternehmen, sondern pro Shop, dem die Mitarbeitenden zugeordnet sind. Gib dazu zusätzlich die Shop-ID aus und sortiere die Ausgabe nach der Shop-ID.
Das Resultat hat die folgende Form:

SHOP_ID	CT_ALL	AVG_SAL	CT_SAL	AVG_DAYS	CT_DAYS
10	4	5'157	4	1'300	3
12	4	5'425	4	935	4
...	...	...	...	...	...

Lösung:

```
SELECT
   SHOP SHOP_ID,
   COUNT(*)  CT_ALL,
   ROUND( AVG( SALARY ) ) AVG_SAL,
   COUNT( SALARY ) CT_SAL,
   ROUND( AVG( TRUNC(CURRENT_DATE - HIRE_DATE )) ) AVG_DAYS ,
   COUNT( HIRE_DATE) CT_DAYS
FROM EMPLO
GROUP BY SHOP
ORDER BY SHOP
```

c) Average: Mehr Gehalt als der Durchschnitt

Gebe eine SQL-Anweisung an, die ermittelt, wer mehr als der Durchschnitt verdient. Von den Mitarbeitenden soll die `ID`, der Name, das Salär, die Shop-ID ausgegeben werden, sortiert nach `ID`.
Gebe in einer weiteren SQL-Anweisung zusätzlich in jedem Datensatz das Durchschnittsgehalt (gerundet) aus.

Lösung:

```
SELECT ID, NAME, SALARY, SHOP FROM EMPLO
WHERE SALARY > (SELECT AVG(SALARY) FROM EMPLO)
ORDER BY ID;

SELECT ID, NAME, SALARY, SHOP, ROUND(AVG_S) AVG_SAL FROM emplo
JOIN (SELECT AVG(SALARY) AVG_S FROM EMPLO)
ON SALARY > AVG_S
ORDER BY ID;
```

d) Average: Mehr Gehalt als der Durchschnitt pro Shop

Formuliere SQL-Anweisungen, mit denen analog zur vorherigen Teilaufgabe alle Mitarbeitenden ausgegeben werden, die *mindestens so viel* wie der Durchschnitt der Mitarbeitenden des gleichen Shops erhalten. Die auszugebenden Attribute sind die gleichen, nur dass sich das Durchschnittsgehalt jetzt auf den Shop bezieht.

Lösung:

Wir beginnen mit dem Ermitteln des Durchschnittsgehalts pro Shop:

```
SELECT SHOP, AVG(SALARY)
FROM EMPLO
GROUP BY SHOP
```

Nun verknüpft man jede Datenzeile aus `EMPLO` mit dem Resultat dieses `SELECT`, das für den passenden Shop das Durchschnittsgehalt liefert.

```
SELECT ID, NAME, SALARY, E.SHOP FROM EMPLO E
JOIN (SELECT SHOP, AVG(SALARY) AVG_SAL_SH
      FROM EMPLO
      GROUP BY SHOP) SUBSEL
ON SUBSEL.SHOP = E.SHOP
WHERE SALARY >= AVG_SAL_SH
```

Da die Subselect-Anweisung bereits das Durchschnittsgehalt pro Shop ermittelt, lässt sich das Attribut `AVG_SAL_SH` einfach in die `SELECT`-Klausel der Gesamtanweisung aufnehmen.
Ergänzt werden muss die `ON`-Bedingung noch um den Zusatz, dass - wenn man die Aufgabenstellung so verstehen möchte - auch Mitarbeitende ohne Shop-Zuordnung ausgegeben werden, sofern sie mindestens so viel verdienen, wie der Durchschnitt aller solcher Mitarbeitenden.
Du kannst Dich gerne vom Effekt dieses Zusatzes überzeugen, indem Du ihn wahlweise aktiviert oder deaktivierst.

```
SELECT ID, NAME, SALARY, E.SHOP, ROUND(AVG_SAL_SH) AVG_SAL_SH FROM EMPLO E
JOIN (SELECT SHOP, AVG(SALARY) AVG_SAL_SH
      FROM EMPLO
      GROUP BY SHOP) SUBSEL
ON SUBSEL.SHOP = E.SHOP
   --die folgende Zeile aktivieren/deaktivieren
   --OR (SUBSEL.SHOP IS NULL AND E.SHOP IS NULL)
WHERE SALARY >= AVG_SAL_SH
```

Aufgabe 4: Mehrere GROUP-BY Attribute

Diese Aufgabe befasst sich mit den Bike-Ausleihen aus der Tabelle `RENTAL`. Beachte, dass Bikes mehrmals ausgeliehen werden können. Beantworte mittels SQL-Anweisungen die folgenden Fragen:

- Wie viele Ausleihen gab es pro Jahr und Monat und wie viele Bikes wurden pro Jahr und Monat ausgeliehen?
- Wie viele Ausleihen gab es pro Jahr, Monat und Shop, und wie viele Bikes wurden pro Jahr, Monat und Shop ausgeliehen?

Das Ergebnis soll jeweils die beteiligten Gruppierungsattribute sowie die Anzahl Bikes enthalten. Die Ausgabereihenfolge erfolgt nach den Gruppierungsattributen.

Das erwartete Resultat sieht wie folgt aus:

RENTAL_YEAR	RENTAL_MONTH	CT_RENTALS	CT_BIKES
2025	1	40	27
2025	2	31	25
...	...	...	...

RENTAL_YEAR	RENTAL_MONTH	SHOP	CT_RENTALS	CT_BIKES
2025	1	10	2	2
2025	1	12	13	8
...	...	...	...	...

Tipps: Mit `EXTRACT( YEAR FROM RENTAL_DATE )` kann man das Jahr aus einem Datumswert ermitteln, analog auch den Monat.
Der Shop einer Ausleihe lässt sich ermitteln, indem man beachtet, welchem Shop das ausgeliehene Bike zugeordnet ist.

Lösung:

```
SELECT
  EXTRACT(YEAR FROM RENTAL_DATE) RENTAL_YEAR,
  EXTRACT(MONTH FROM RENTAL_DATE) RENTAL_MONTH,
  COUNT(*) CT_AUSLEIHEN,
  COUNT(DISTINCT BIKE) CT_BIKES
FROM RENTAL R
GROUP BY RENTAL_YEAR, RENTAL_MONTH
ORDER BY RENTAL_YEAR, RENTAL_MONTH;

SELECT
  EXTRACT(YEAR FROM RENTAL_DATE) RENTAL_YEAR,
  EXTRACT(MONTH FROM RENTAL_DATE) RENTAL_MONTH,
  SHOP,
  COUNT(*) CT_RENTALS,
  COUNT(DISTINCT BIKE) CT_BIKES
FROM RENTAL R
JOIN BIKE B ON B.ID = R.BIKE
GROUP BY RENTAL_YEAR, RENTAL_MONTH, SHOP
ORDER BY RENTAL_YEAR, RENTAL_MONTH, SHOP;
```

Aufgabe 5: GROUP-BY und HAVING

Welche Bikes wurden vor dem 4. März 2025 mindestens 3 mal ausgeliehen? Formuliere ein SQL-Anweisung, mit der die Bike-ID, Modell und die Häufigkeit der Ausleihe für dieses Bike ausgegeben werden. Die Reihenfolge soll nach Bike-ID erfolgen.

Lösung:

Zunächst lassen sich mit der `HAVING`-Klausel Gruppierungen mittels Aggregatfunktionen filtern.

```
SELECT BIKE, COUNT(*) CT
FROM RENTAL R
WHERE R.RENTAL_DATE < DATE '2025-03-04'
GROUP BY BIKE
HAVING COUNT(*)>3
```

Jetzt benötigt man noch den Modellnamen, den man über eine Verknüpfung mit der Tabelle `BIKE` erhält:

```
SELECT BIKE B_ID, MODELL, CT
FROM BIKE
JOIN
( SELECT BIKE, COUNT(*) CT
```

```
  FROM RENTAL R
  WHERE R.RENTAL_DATE < DATE '2025-03-04'
  GROUP BY BIKE
  HAVING COUNT(*)>3
) T ON T.BIKE = BIKE.ID
ORDER BY T.BIKE
```

Resultat:

B_ID	MODELL	CT
30029	Loadster	4
30040	TrailKing	4

Aufgabe 6: Subselects

Beantworte via SQL-Anweisungen die folgenden Fragen:

- An welchem Tag wurden erstmals Bikes ausgeliehen? Gib von allen Ausleihen, die an diesem Tag getätigt wurden, die ID, die Bike-ID und das Datum aus.
- Welche Kunden haben die meisten Ausleihen getätigt? Gib alle diese Kunden (Customer-ID) aus, zusammen mit der Anzahl der Ausleihen.
- Welche Kunden haben die meisten Bikes ausgeliehen? Gib alle diese Kunden (Customer-ID) aus, zusammen mit der Anzahl entliehener Bikes.

Lösung:

Beachte, dass es mehrere solche Ausleihen geben kann. Daher wäre eine Lösung, die den ersten Datensatz aus einer nach Ausleihdatum aufsteigend sortierten Liste auswählt, nicht korrekt.

```
SELECT ID, BIKE, RENTAL_DATE FROM RENTAL
WHERE RENTAL_DATE = (SELECT MIN(RENTAL_DATE) FROM RENTAL);

-- alternative Lösung: Ausleihen, zu denen es keine früheren gibt
SELECT T.ID, T.BIKE, T.RENTAL_DATE FROM RENTAL T
LEFT JOIN RENTAL T_NOTEXIST
ON T.RENTAL_DATE > T_NOTEXIST.RENTAL_DATE
WHERE T_NOTEXIST.ID IS NULL;
```

Für die Lösung der zweiten Teilaufgabe betrachten wir die folgende Anweisung, die pro Kunde/Kundin die Anzahl seiner/ihrer Ausleihen ermittelt:

```
SELECT CUSTOMER, COUNT(*) CT_RENTALS
FROM RENTAL R
GROUP BY CUSTOMER
```

Da diese Anweisung in der gesuchten Anweisung mehrmals auftritt, geben wir ihr als *Common Table Expression* (CTE) den Namen `CT_CUST` und können diesen Ausdruck wie einen gewöhnlichen Tabellennamen mehrmals verwenden.

```
WITH CT_CUST(CUSTOMER, CT_RENTALS) AS
(SELECT
       CUSTOMER,
       COUNT(*) CT_RENTALS
       FROM RENTAL R
```

```
        GROUP BY CUSTOMER
)
SELECT CUSTOMER, CT_RENTALS FROM CT_CUST
WHERE CT_RENTALS = (SELECT MAX(CT_RENTALS) FROM CT_CUST)
ORDER BY CUSTOMER
```

Der Vollständigkeit wegen hier die SQL-Anweisung ohne Verwendung von `WITH`. An der Markierung erkennt man die genannte `SELECT`-Anweisung:

```
SELECT CUSTOMER, CT_RENTALS
FROM
-->(SELECT CUSTOMER, COUNT(*) CT_RENTALS
--> FROM RENTAL R GROUP BY CUSTOMER)
WHERE CT_RENTALS =
 ( SELECT MAX(CT_RENTALS) FROM
   -->(SELECT CUSTOMER, COUNT(*) CT_RENTALS
   --> FROM RENTAL R GROUP BY CUSTOMER)
 )
ORDER BY CUSTOMER;
```

Die Lösung für die dritte Teilaufgabe besteht darin, in der vorherigen Anweisung `COUNT(*)` durch `COUNT(DISTINCT BIKE)` zu ersetzen.

Die Resultate sind:

CUSTOMER	CT_RENTALS
50136	6
50151	6

CUSTOMER	CT_BIKES
50117	4
50118	4
50136	4
...	...