Übungen Grundlagen SQL

Klaus-Georg Deck

2025-07-15

Einleitung

In diesem Skript findest du typische SQL-Übungen mit Beispielcode und den erwarteten Ergebnistabellen. Ziel ist es, grundlegende SQL-Techniken wie Selektion, Bedingungen, Aggregation und Joins zu üben.

Voraussetzungen

Du verfügst über einen Zugang zu einem Datenbank-Server (Oracle oder Postgres), auf dem die Realisierung des Bike-Verleih-Szenarios erfolgreich installiert wurde. Eine Beschreibung, wie dies funktioniert, sowie eine Übersicht über diesen Anwendungsfall findest Du hier (Link/Referenz noch ergänzen).

Aufgabe 1: Überblick über die Tabellen `EMPLO` und `SHOP`

a) Anzahl der Datensätze

Ermittle die Anzahl der Datensätze dieser Tabellen in einer jeweils separaten SQL-Anweisung.

Lösung:

```sql
SELECT COUNT(*) FROM EMPLO;

SELECT COUNT(*) FROM SHOP;
```

b) Sortieren und Limitieren

Zeige für beide Tabellen jeweils alle Datensätze an, sortiert nach dem Attribut `ID`.
Tipp: Mit `FETCH FIRST 20 ROWS ONLY` lässt sich die Anzahl beschränken.

Lösung:

```sql
SELECT * FROM EMPLO
ORDER BY ID
FETCH FIRST 20 ROWS ONLY;

SELECT * FROM SHOP
ORDER BY ID;
```

c) Verweise

In jeder der beiden Tabellen gibt es ein Attribut, das auf das Attribut `ID` der anderen Tabelle verweist. Welche Attribute sind das jeweils und was ist vermutlich deren Bedeutung? (Für diese Aufgabe kann in beiden Tabellen das Attribut `ADDRESS` ignoriert werden.)

In der Tabelle `EMPLO` gibt es das Attribut `REPORTS_TO`. Worauf verweist dies und was ist vermutlich dessen Bedeutung?

Lösung:

In der Tabelle `EMPLO` verweist das Attribut `SHOP` auf das Attribut `ID` der Tabelle `SHOP`. Dieses Attribut beinhaltet die Information, in welchem Shop die jeweilige Person tätig ist.
In der Tabelle `SHOP` verweist das Attribut `MANAGER` auf die `ID` derjenigen Person aus der Tabelle `EMPLO`, die diesen Shop leitet.

Wenn man sich einige Datensätze ansieht, lässt sich erkennen, dass die Werte von `REPORTS_TO` als Attributwerte von `ID` der gleichen Tabelle `EMPLO` vorkommen. Hierbei handelt es sich um die Information, wer der oder die direkte Vorgesetzte ist.

Aufgabe 2: Selektion von Attributen und `ORDER BY`

a) Selektion von Attributen

Welche SQL-Anweisung liefert eine Liste aller Shops, wobei nur die Attribute `ID` und `NAME` angezeigt werden? Die Ausgabereihenfolge soll nach `ID` absteigend erfolgen.

Lösung:

```
SELECT ID, NAME FROM SHOP
ORDER BY ID DESC
```

b) ORDER BY mit Funktionen

Welche SQL-Anweisung liefert eine Liste aller Namen von Mitarbeitenden? Es soll nach der Länge des Namens, die längsten zuerst, und bei gleicher Länge zeichencodebasiert sortiert werden. Demnach kommt `Pauline` vor `Lea` vor `Leo`.
Wir sind nur an den Namen und nicht an der Häufigkeit ihres Vorkommens interessiert, Mehrfachvorkommen bitte ignorieren. Tipp: Die SQL-Funktion `LENGTH()` liefert die Anzahl der Zeichen einer Zeichenkette. Verwende diese in `ORDER BY`.

Lösung:

Zunächst liefert
`SELECT DISTINCT NAME FROM EMPLO`
eine Liste der Namen von Mitarbeitenden ohne Duplikate. Diese ist noch zu sortieren:

```
SELECT DISTINCT NAME
FROM EMPLO
ORDER BY LENGTH(NAME) DESC, NAME
```

c) Limitieren

Ermittle die Namen und das Salär der 8 bestbezahlten Mitarbeitenden. Gehe dabei davon aus, dass es nicht mehrere Mitarbeitende gibt, die sich die 8. Position teilen.
Zusatzfrage: Wie könnte man das Resultat nach Salär aufsteigend sortiert ausgeben?

Lösung:

Diese Anweisung liefert das gewünschte Resultat

```
SELECT NAME, SALARY FROM EMPLO
ORDER BY SALARY DESC, ID
FETCH FIRST 8 ROWS ONLY
```

Dieses könnte man noch wie folgt umsortieren (Zusatzfrage):

```
SELECT NAME, SALARY FROM (
   SELECT NAME, SALARY FROM EMPLO
   ORDER BY SALARY DESC, ID
   FETCH FIRST 8 ROWS ONLY
) ORDER BY SALARY
```

d) Berechnete Attribute

Ermittle eine Liste aller Mitarbeitenden, wobei jeweils `ID`, Name und das Jahresgehalt als `ANNUAL_SALARY` (entspricht 12 Monatsgehältern) in beliebiger Reihenfolge angezeigt werden.

Lösung:

```
SELECT ID, NAME, SALARY * 12 AS ANNUAL_SALARY FROM EMPLO
```

Das Schlüsselwort `AS` für Spaltenaliase ist optional, erhöht jedoch die Lesbarkeit.

Aufgabe 3: Datentyp `DATE` in SQL

a) Anzahl Tage

Die Tabelle `EMPLO` beinhaltet mit `HIRE_DATE` das Einstellungsdatum von Mitarbeitenden. Erstelle eine SQL-Anweisung, mit der Mitarbeitende ausgegeben werden, die am 31.12.2025 weniger als 900 Tage im Unternehmen tätig waren. Die Ausgabe soll neben der `ID` den Namen, das Einstellungsdatum und die Anzahl der Tage im Unternehmen umfassen. Die Ausgabereihenfolge ist beliebig.
Anmerkungen:
Mit dem Ausdruck `DATE '2025-12-31'` erhält man den Wert für das gewünschte Datum. Die Differenz zwischen zwei Werten des Typs `DATE` liefert die Anzahl Tage, die zwischen diesen Kalenderdaten liegen.

Lösung:

```
SELECT ID, NAME, HIRE_DATE, DATE '2025-12-31' - HIRE_DATE AS DAYS
FROM EMPLO
WHERE DATE '2025-12-31' - HIRE_DATE < 900
```

Falls der Differenzausdruck nur einmal verwendet werden soll:

```
SELECT ID, NAME, HIRE_DATE, DAYS FROM
( SELECT ID, NAME, HIRE_DATE, DATE '2025-12-31' - HIRE_DATE AS DAYS
  FROM EMPLO
) WHERE DATE DAYS < 900
```

b) Aktuelles Datum

Die Tabelle `RENTAL` enthält Informationen über Ausleihen von Velos, insbesondere das Ausleihdatum und Rückgabedatum. Ermittle via SQL die Ausleihen, bei denen das Velo noch nicht zurückgegeben wurde, und gebe deren Ausleihe-ID an sowie das Ausleihedatum und die Anzahl der Tage, die das Velo bis heute ausgeliehen war. Die Ausgabereihenfolge ist beliebig.
Anmerkungen:
Ausleihen, bei denen das Velo noch nicht zurückgegeben wurde, erkennt man daran, dass das Attribut `RETURN_DATE` leer (`NULL`) ist.
Um in SQL zur Laufzeit das aktuelle Datum zu ermitteln, kann man den Ausdruck `CURRENT_DATE` verwenden. Im Fall von Oracle wird `TRUNC(CURRENT_DATE)` empfohlen (Begründung siehe Lösung dieser und folgender Teilaufgabe).

Lösung:

Die folgenden Anweisungen liefern das gewünschte Resultat:

```
-- Postgres
SELECT ID, RENTAL_DATE, CURRENT_DATE - RENTAL_DATE DAYS
FROM RENTAL WHERE RETURN_DATE IS NULL;

-- Oracle
SELECT ID, RENTAL_DATE, TRUNC(CURRENT_DATE) - RENTAL_DATE DAYS
FROM RENTAL WHERE RETURN_DATE IS NULL;

-- Postgres und Oracle
SELECT ID, RENTAL_DATE, TRUNC(CURRENT_DATE - RENTAL_DATE) DAYS
FROM RENTAL WHERE RETURN_DATE IS NULL;
```

c) Nullwertfunktion `COALESCE()`

Die Anweisung

```
SELECT AVG( RETURN_DATE - RENTAL_DATE ) FROM RENTAL
```

liefert die durchschnittliche Ausleihdauer in Tagen. Dabei werden noch nicht zurückgegebene Velos ignoriert. Erstelle eine modifizierte SQL-Anweisung, welche die noch nicht zurückgegebenen Velos berücksichtigt. Diese sollen in das Resultat einfliessen, indem man als Rückgabedatum das heutige Datum annimmt.
Tipp:
Verwende `CURRENT_DATE` zusammen mit der Funktion `COALESCE()`. Der Ausdruck `COALESCE(A, B)` liefert den Wert von `A`, sofern dieser nicht `NULL` ist, sonst `B`.

Lösung:

Eine Lösung erhält man dadurch, dass im obigen Statement der Ausdruck `RETURN_DATE` durch `COALESCE( RETURN_DATE, CURRENT_DATE )` resp. `COALESCE( RETURN_DATE, TRUNC(CURRENT_DATE) )` (für Oracle) ersetzt wird. Dieser Ausdruck liefert `RETURN_DATE`, falls dieses gesetzt ist, resp. das aktuelle Datum, falls `RETURN_DATE` leer ist.
Hier die Lösungen:

```
-- Postgres
SELECT AVG( COALESCE( RETURN_DATE, CURRENT_DATE ) - RENTAL_DATE )
FROM RENTAL;

-- Oracle
SELECT AVG( COALESCE( RETURN_DATE, TRUNC(CURRENT_DATE) ) - RENTAL_DATE )
```

```
FROM RENTAL;
```

Unter Oracle führt das erste Statement zu unterschiedlichen Ergebnissen, wenn es mehrmals nacheinander ausgeführt wird. Dies liegt daran, dass Oracle im Datentyp `DATE` auch die Uhrzeit ablegt und diese bei der Berechnung berücksichtigt. Der Aufruf von `TRUNC()` entfernt den Zeitwert, indem er auf `0` gesetzt wird.

Aufgabe 4: Einfache Stringverarbeitung

Für die Mitarbeitenden sollen Namenskürzel eingeführt und möglichst automatisiert berechnet werden. Ein erster Versuch besteht darin, die ersten 3 Buchstaben (Kleinschreibung) des Namens zu verwenden. Die folgende Anweisung erzeugt eine Liste aller Mitarbeitenden mit `ID`, Name und Kürzel:

```
SELECT
   ID,
   NAME,
   SUBSTR( LOWER(NAME), 1, 3 ) AS KURZ
FROM EMPLO
ORDER BY NAME
```

Beachte die Verwendung der Funktionen `LOWER()` und `SUBSTR()`. Wir gehen hier davon aus, dass jeder Name mindestens 3 Zeichen enthält.

Wie man am Ergebnis erkennt, gibt es jedoch Duplikate.

ID	NAME	KURZ
16011	Anna	ann
16043	Anna	ann
16010	Ben	ben
…	…	…

Ergänze die SQL-Anweisung, so dass die folgenden Varianten für Namenskürzel als Attribute `KURZ_V1` und `KURZ_V2` ausgegeben werden:

- `KURZ_V1`: `KURZ` ergänzt um die letzten beiden Ziffern von `ID`
- `KURZ_V2`: `KURZ` ergänzt um das Einstellungsdatum in der Form `YYYYMMDD`

Die SQL-Anweisung soll also folgendes Resultat erzeugen:

ID	NAME	KURZ	KURZ_V1	KURZ_V2
16011	Anna	ann	ann11	ann20250101
16043	Anna	ann	ann43	ann20220101
16010	Ben	ben	ben10	ben20220101
…	…	…	…	…

Was ist von einer solchen Vergabe von Namenskürzel zu halten? Gibt es Alternativen?

Tipp:

Strings lassen sich mit der Funktion `CONCAT()` oder dem Operator `||` verketten. Zur Umwandlung des Datums in die gewünschte Form verwende die Funktion `TO_CHAR(HIRE_DATE, 'YYYYMMDD')`

und für die Extraktion der letzten beiden Ziffern aus der `ID` kann die Funktion `MOD(ID, 100)` (Rest bei Division durch 100) eingesetzt werden, wobei das Resultat noch via `TO_CHAR()` mit Formatwert `FM00` zu formatieren ist. (Hier gibt es auch alternative Lösungswege.)

Lösung:

```
SELECT
  ID,
  NAME,
  SUBSTR( LOWER(NAME), 1, 3 ) AS KURZ,
  SUBSTR( LOWER(NAME), 1, 3 ) || TO_CHAR(MOD(ID, 100), 'FM00') AS KURZ_V1,
  SUBSTR( LOWER(NAME), 1, 3 ) || TO_CHAR(HIRE_DATE, 'YYYYMMDD') AS KURZ_V2
FROM EMPLO
ORDER BY NAME
```

Dazu ist zu sagen, dass die Verwendung der vollständigen `ID` im Namenskürzel eine eindeutige Identifikation ermöglicht, das Kürzel wäre aber alles andere als *kurz*. Verwendet man stattdessen nur natürliche Personenmerkmale, besteht die Gefahr von Kollisionen. In der Praxis kann man etwa Teile des Vor- und Nachnamens verwenden und diese im Fall von Kollisionen über einen Zähler unterscheiden. Dies setzt natürlich voraus, dass bereits generierte Kürzel in der Datenbank gespeichert werden.

Aufgabe 5: `WHERE`-Klausel

a) Gleichheit - Ungleichheit

Erstelle eine `SELECT`-Anweisung, die alle Mitarbeitenden mit `ID`, Name und Einstellungsdatum ausgibt, die `Tim` heissen, sowei eine Anweisung, die Mitarbeitenden ausgibt, die *nicht* `Tim` heissen.

Erstelle eine weitere `SELECT`-Anweisung, die alle Informationen über diejenigen Mitarbeitenden ausgibt, deren Salär über 6'000 liegt.

In allen Fällen ist die Ausgabereihenfolge unwichtig.

Lösung:

```
SELECT ID, NAME, HIRE_DATE
FROM EMPLO
WHERE NAME = 'Tim';

SELECT ID, NAME, HIRE_DATE
FROM EMPLO
WHERE NAME <> 'Tim';

SELECT * FROM EMPLO
WHERE SALARY > 6000;
```

Anstelle `NAME <> 'Tim'` kann man auch `NAME != 'Tim'` verwenden.

b) BETWEEN

Erstelle eine Anweisung, die alle Informationen über diejenigen Mitarbeitenden ausgibt, deren Salär zwischen 6'000 und 6'500 liegt. Dabei sollen Mitarbeitende, deren Salär direkt an der Grenze (6'000 resp. 6'500) liegt, ebenfalls berücksichtigt werden. Verwende dabei das Schlüsselwort `BETWEEN`.

Erstelle eine Anweisung ohne `BETWEEN`, die das gleiche Resultat liefert. Verwende Vergleiche und den Booleschen Operator `AND`.

In beiden Fällen sollen die Datensätze nach Salär aufsteigend sortiert werden.

Lösung:

```
SELECT * FROM EMPLO
WHERE SALARY BETWEEN 6000 AND 6500
ORDER BY SALARY;

SELECT * FROM EMPLO
WHERE SALARY >= 6000 AND SALARY <=6500
ORDER BY SALARY;
```

Beachte, dass das Schlüsselwort `AND` in `SQL` zwei unterschiedliche Funktionen erfüllt: Zum einen als logischer Operator, zum anderen wird es bei der Angabe von Wertebereichen in Verbindung mit `BETWEEN` verwendet. Auch wenn in komplexen `WHERE`-Bedingungen beide Verwendungen zugleich auftreten können und die jeweilige Bedeutung syntaktisch eindeutig ist, wirkt dies aus Sicht des Autors etwas unglücklich.

Aufgabe 6: `NULL` - Das unbekannte Nichts

a) `NULL` in Vergleichen

Das Schlüsselwort `NULL` steht in SQL-Datenbankumfeld für *leer*, *unbekannt*, *kein Wert* etc. Dies tritt dann auf, wenn etwa bei optionalen Angaben zu Geschlecht, Telefonnummer oder Anzahl der Kinder nichts angegeben wird. So macht es einen Unterschied, ob jemand bei *Anzahl Kinder* den Zahlenwert `0` angibt oder keine Angabe macht. Im ersten Fall steht fest, dass die Person keine Kinder hat; im zweiten Fall ist unklar, ob sie Kinder hat – und wenn ja, wie viele.

In der Tabelle `EMPLO` ist das Attribut `REPORTS_TO` optional, d.h. es kann Datensätze geben, bei denen dieses Attribut leer ist.

Die folgende Anweisung zeigt, dass es genau einen solchen gibt:
`SELECT ID, NAME, JOB, REPORTS_TO FROM EMPLO WHERE REPORTS_TO IS NULL`

ID	NAME	JOB	REPORTS_TO
16051	Nole	CEO	

Hierbei handelt es sich um den oder die CEO des Bike-Verleih, was erklärt, warum `REPORTS_TO` leer ist. Denn bekanntermassen haben CEOs keine direkten Vorgesetzten.

Gib mit einer SQL-Anweisung die Anzahl der Mitarbeitenden mit direkten Vorgesetzten (als Attribut `W_MGR`) und in einer weiteren Anweisung die Anzahl der Mitarbeitenden ohne direkten Vorgesetzten (als Attribut `WO_MGR`) aus.

Versuche in einer dritten SQL-Anweisung, die Gesamtzahl der Mitarbeitenden, die Zahl der Mitarbeitenden mit sowie die Zahl der Mitarbeitenden ohne direkten Vorgesetzten auszugeben. Tipp: Verwende auch `COUNT( REPORTS_TO )`.

Lösung:

Die ersten beiden Anweisungen liefern die Zahl der Mitarbeitenden mit resp. ohne direkten Vorgesetzten:

```
SELECT COUNT(*) AS W_MGR
FROM EMPLO
WHERE REPORTS_TO IS NOT NULL;
```

```
SELECT COUNT(*) AS WO_MGR
FROM EMPLO
WHERE REPORTS_TO IS NULL;
```

Hier ist zu beachten, dass Vergleiche wie `REPORTS_TO = NULL` oder `REPORTS_TO <> NULL` nicht funktionieren, denn `NULL` repräsentiert keinen Wert im eigentlichen Sinn.

Mit der folgenden Anweisung kann die gesamte Information in einer einzigen Zeile ausgegeben werden, man kommt sogar ohne `WHERE`-Klausel aus.

```
SELECT
   COUNT(*) AS ALLE,
   COUNT( REPORTS_TO ) AS W_MGR,
   COUNT(*) - COUNT( REPORTS_TO ) AS WO_MGR
FROM EMPLO
```

Der Ausdruck `COUNT( REPORTS_TO )` zählt nämlich genau die Datensätze, für die `REPORTS_TO` nicht `NULL` ist.

Aufgabe 7: JOIN - Verbinden von Tabellen

a) INNER JOIN - Verschiedene Varianten

Gib für jeden Mitarbeitenden die `ID`, den Namen als `ENAME` sowie den Namen des Shops als `SNAME` aus, dem er oder sie zugeordnet ist. Die Reihenfolge der Ausgabe soll nach der `ID` der Mitarbeitenden aufsteigend erfolgen.

Erstelle zwei SQL-Anweisungen in unterschiedlicher JOIN-Syntax, die dieses Ergebnis liefern.

Lösung:

```
-- JOIN
SELECT EMPLO.ID, EMPLO.NAME AS ENAME, SHOP.NAME AS SNAME
FROM EMPLO
JOIN SHOP ON EMPLO.SHOP = SHOP.ID
ORDER BY EMPLO.ID;

-- impliziter JOIN: Tabellen mit Komma getrennt; Bedingung in WHERE
SELECT EMPLO.ID, EMPLO.NAME AS ENAME, SHOP.NAME AS SNAME
FROM EMPLO, SHOP
WHERE EMPLO.SHOP = SHOP.ID
ORDER BY EMPLO.ID;
```

Bei einem `JOIN` ohne weitere Spezifikation handelt es sich um einen `INNER JOIN`, d.h. das Schlüsselwort `INNER` kann immer weggelassen werden.

b) JOIN - Attribute mit und ohne Tabellen-Spezifikation

Wir betrachten hier die Tabellen `EMPLO` und `ADDRESS`, die über `EMPLO.ADDRESS` und `ADDRESS.ID` inhaltlich verbunden sind.

Welche Mitarbeitenden wohnen im Kanton (`STATE`) Wallis (`VS`), aber nicht in der Stadt (`CITY`) `Sion`? Gib die Namen der Mitarbeitenden und ihren Wohnort (`CITY`) aus, sortiert nach Mitarbeitendenname.

Erstelle eine erste Anweisung, bei der jedes Attribut den voll qualifizierten Namen enthält, wie z.B. `EMPLO.NAME`.

In der zweiten Anweisung soll auf die Angabe von Tabellennamen nach Möglichkeit verzichtet werden. In welchen Fällen ist dies möglich, in welchen nicht?

Folgendes Resultat wird erwartet:

NAME	CITY
Corinne	Visp
Melanie	Visp
Pascal	Visp
Silvan	Zermatt

Lösung:

```sql
-- voll qualifizierte Attributnamen
SELECT EMPLO.NAME, ADDRESS.CITY
FROM EMPLO
JOIN ADDRESS ON EMPLO.ADDRESS = ADDRESS.ID
WHERE ADDRESS.STATE = 'VS' AND ADDRESS.CITY <> 'Sion'
ORDER BY EMPLO.NAME;

-- Tabellennamen, nur wo notwendig
SELECT NAME, CITY
FROM EMPLO
JOIN ADDRESS ON ADDRESS = ADDRESS.ID
WHERE STATE = 'VS' AND CITY <> 'Sion'
ORDER BY NAME;
```

Wenn der Attributname innerhalb der beteiligten Tabellen eindeutig ist (z.B. `NAME`, `CITY`, `ADDRESS`), dann kann auf die Angabe der Tabelle verzichtet werden. Das einzige Attribut, das im Beispiel in beiden Tabellen vorkommt, ist `ID`, so dass die Angabe von `ADDRESS.ID` in der `ON`-Bedingung notwendig ist. Würde man nur `ID` verwenden, wäre nicht klar, auf welche Tabelle sich das Attribut bezieht.

Aufgabe 8: JOIN - Komplexeres Beispiel

Im Bike-Verleih-Szenario können Kunden verschiedene Bikes ausleihen. Die beteiligten Tabellen sind `CUSTOMER`, `RENTAL` und `BIKE`. Wir interessieren uns für Kunden, die vor dem 15. Januar 2025 (mindestens einmal) ein Bike mit Rahmengrösse XL geliehen haben, das kein E-Bike ist.

a) JOIN mehrerer Tabellen

Formuliere eine SQL-Anweisung, mit der die `ID` der Ausleihen, `ID` der Kunden, Name der Kunden und Modell des ausgeliehenen Bikes ausgegeben werden kann, die der genannten Bedingung entsprechen. Die Ausgabe soll nach `ID` der Ausleihen sortiert sein. Das Resultat hat die folgende Form (es wird nur die erste Datenzeile angezeigt):

RID	KUNDE_ID	KUNDE_NAME	MODELL
60004	50151	Tom	CityLite
...	...	...	...

Lösung:

```
SELECT
  R.ID AS RID,
  C.ID AS KUNDE_ID,
  C.NAME AS KUNDE_NAME,
  B.MODELL
FROM RENTAL R
JOIN CUSTOMER C ON R.CUSTOMER = C.ID
JOIN BIKE B ON R.BIKE = B.ID
WHERE B.FRAME_SIZE = 'XL' AND B.EBIKE=0
AND R.RENTAL_DATE < DATE '2025-01-15'
ORDER BY R.ID
```

Hier lassen sich Aliasnamen für Tabellen einsetzen, um die Anweisung übersichtlicher zu gestalten. Um eine solche mehrfach verknüpfte Anweisung zu erstellen, wird man zunächst diejenigen Tabellen ermitteln, die für die gewünschte Information benötigt werden. Das sind hier die bereits genannten `RENTAL`, `CUSTOMER` und `BIKE`.

Danach kümmert man sich um die Verknüpfungen, die für die jeweiligen `JOIN`-Bedingungen relevant sind, und formuliert die Verknüpfungen. Hier kann man auch schrittweise vorgehen, solange man in der `SELECT`-Klausel nicht die vollständige Information verlangt.

Den ersten Schritt könnte man etwa so gestalten:

```
SELECT
   R.ID AS RID,
   C.ID AS KUNDE_ID,
   C.NAME AS KUNDE_NAME
FROM RENTAL R
JOIN CUSTOMER C ON R.CUSTOMER = C.ID
```

Anschliessend fährt man mit der Tabelle `BIKE` in einem weiteren `JOIN` fort und ergänzt das noch fehlende Attribut `B.MODELL` in der `SELECT`-Klausel.

b) Eindeutigkeit mit COUNT(DISTINCT)

Verwende das Resultat der vorherigen Teilaufgabe und formuliere eine SQL-Anweisung, mit der die Anzahl der Ausleihen (als `ANZ_RT`), die Anzahl der Kunden (`ANZ_C`), die Anzahl der unterschiedlichen Namen von Kunden (`ANZ_NAME`) und die Anzahl der unterschiedlichen Bike-Modelle (`ANZ_MODELL`) ausgegeben werden. Erwartet wird das folgende Resultat:

ANZ_RT	ANZ_C	ANZ_NAME	ANZ_MODELL
8	7	6	7

Tipp: Verwende Ausdrücke wie `COUNT(DISTINCT R.ID)`, um die Anzahl unterschiedlicher Werte zu ermitteln.

Lösung:

```
SELECT
  COUNT(DISTINCT R.ID)   ANZ_RT,   COUNT(DISTINCT C.ID)     ANZ_C,
  COUNT(DISTINCT C.NAME) ANZ_NAME, COUNT(DISTINCT B.MODELL) ANZ_MODELL
FROM RENTAL R
JOIN CUSTOMER C ON R.CUSTOMER = C.ID
JOIN BIKE B ON R.BIKE = B.ID
WHERE B.FRAME_SIZE = 'XL' AND B.EBIKE=0
```

```
AND R.RENTAL_DATE < DATE '2025-01-15'
```