TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS19/20

Christoph Anneser, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws1920/grundlagen/

Blatt Nr. Z1

Dieses **Zusatzblatt** wird nicht in der Übung besprochen!
Tool zum Üben von SQL-Anfragen: https://hyper-db.com/interface.html.
Onlineshop-Schema: http://db.in.tum.de/teaching/ws1920/grundlagen/schema.pdf

Zusatzaufgabe 1

Diese Aufgabe ist die vierte in einer Reihe von Aufgaben, in denen Sie lernen werden, einen gegebenen Sachverhalt zu analysieren, geeignete Modelle zu entwerfen, diese in ein Datenbankschema zu überführen, das Schema in einer (lokalen) Datenbank aufzusetzen und mit Daten zu füllen. Sie werden ebenfalls lernen, wie Sie die Datenbank für bestimmte Anfragen optimieren können.

In dieser Aufgabe importieren wir zufallsgenerierte Daten für den Onlineshop in unsere lokal installierte Postgres-Datenbank. Laden Sie dafür den auf Moodle bereitgestellten zip-Ordner herunter und entpacken Sie ihn. Importieren Sie dann die Daten mit folgendem SQL-File: https://db.in.tum.de/teaching/ws1920/grundlagen/shop.sql (Sie müssen eventuell noch die Pfade anpassen).

Beachten Sie außerdem, dass ein Teil der Daten (Kundenprofile, Namen, Adressen, etc.) mit dem Python-Paket Faker[1] zufällig generiert wurden und sich daher *nicht* auf real existierende Personen oder Unternehmen bezieht. Es handelt sich hierbei um *rein fiktive* Personen, Unternehmen und Adressen.

```
$ psql shop < shop.sql
```

Zusatzaufgabe 2

Formulieren Sie die folgenden Anfragen auf dem Schema des Onlineshops in SQL. Aufgaben, die beispielsweise Schreibrechte auf der Datenbank erfordern, sind durch * kenntlich gemacht und sollten Sie in Ihrer lokalen Datenbank testen, da sie im Webinterface nicht ausgeführt werden können.

(a) Finden Sie für jedes *Land* den *Kunden*, auf den der höchste Umsatz entfallen ist.

(b) Finden Sie für jedes *Land* den *Kunden*, auf den der zweithöchste Umsatz entfallen ist. Der zweithöchste Umsatz ist definiert als der nächstkleinere Umsatz verglichen zum höchsten im jeweiligen *Land* angefallenen Umsatz. Geben Sie eine Anfrage *mit* korrelierter Unteranfrage und eine Anfrage *ohne* korrelierter Unteranfrage an.

(c) Ermitteln Sie die Exporte der einzelnen *Länder* und schlüsseln Sie diese nach Jahr und Importland auf.

(d) Geben Sie eine Übersicht, wie viele *Kunden n Bestellungen* getätig haben und schlüsseln Sie diese nach *Kontinent* auf.

* Exportieren Sie das Ergebnis Ihrer Anfrage als CSV-Datei. Welche Verteilung vermuten Sie hinter den getätigten Bestellungen der Kunden?

[1] https://github.com/joke2k/faker/

(e) Finden Sie die *Kunden*, die von mindesten fünf unterschiedlichen *Lieferanten*, die den Sitz im gleichen *Land* haben, *Artikel* erhalten haben, und die insgesamt einen Umsatz von 1000 € oder mehr erzielt haben (unabhängig vom Lieferanten).

(f) Finden Sie die *Kunden*, die im Jahr 2018 *Artikel* von mindestens zwei *Lieferanten* a und b, die ihren Sitz in einem der *Länder* {GERMANY, BRAZIL, JAPAN, ROMANIA, UNITED KINGDOM, CANADA} haben, erhielten. Beachten Sie außerdem, dass a und b ihren Sitz in unterschiedlichen *Ländern* haben müssen.

(g) Finden Sie für jeden *Artikel* und jedes *Land* den *Lieferanten*, der den günstigsten Lieferpreis verlangt.

(h) Finden Sie für jeden *Artikel* und jedes *Land* die Preisdifferenz des günstigsten und teuersten Angebots. Geben Sie auch den prozentualen Aufschlag vom günstigeren zum teureren Angebot mit zwei Nachkommastellen an.

(i) Finden Sie die *Kunden*, die am meisten Geld hätten sparen können, wenn sie beim *Lieferanten* gekauft hätten, der den entsprechenden *Artikel* im Land des *Kunden* am günstigsten anbietet.

Wie die obige Anfrage (i) zeigt, treten negative Einsparungen auf. Dies hat den Grund, dass laut Datenbank *Kunden* von *Lieferanten* beliefert werden, die laut Relation *liefertNach* nicht möglich wären. Diese fehlerhaften Daten werden wir im Folgenden korrigieren.

(j) * Finden Sie die *Bestellpositionen*, die laut der Relation *liefertNach* gar nicht möglich wären.

  (1) * Geben Sie **eine** SQL-Anweisung an, um eine neue Tabelle *Bestellpositionen-Korrigiert* zu erstellen, die das gleiche Schema und die gleichen Einträge wie die Tabelle *Bestellpositionen* enthält.

  (2) * Updaten Sie nun die Tabelle *BestellpositionenKorrigiert*: Jede Position, bei der der *Lieferant* nicht das Land des *Kunden* beliefert, soll durch den günstigsten das Land des Kunden beliefernden Lieferanten (günstig in Bezug auf den in der *Bestellposition* referenzierten *Artikel*) ersetzt werden.

  (3) * Überprüfen Sie, ob Ihre Anweisungen zum gewünschten Ergebnis geführt haben, indem Sie bei obiger Anfrage überprüfen, dass Kunden keine negativen Einsparungen mehr erhalten würden.

Nachdem wir die Daten nun bereinigt haben, wollen wir sicher gehen, dass nicht erneut fehlerhafte *Bestellpositionen* eingefügt werden können.

(k) Erörtern Sie unterschiedliche Möglichkeiten, wie man die Relation *Bestellpositionen* modifizieren könnte, damit solche fehlerhaften Daten nicht eingefügt werden können. Gehen Sie dabei insbesondere auf *Fremdschlüssel* und `check`-constraints ein.

(l) Ermitteln Sie die Entwicklung der Marktanteile für *Länder* l innerhalb ihres *Kontinents* k für alle verschiedenen Artikeltypen an. Die Entwicklung soll auf Jahresbasis analysiert werden und auf den Preisen der *Bestellpositionen* beruhen. Dabei sollen nur die bestellten Waren berücksichtigt werden, die von *Lieferanten* mit Sitz in k geliefert wurden und deren *Kunde* ebenfalls in k beheimatet ist. Wenn der umgesetzte Wert im vorherigen Jahr 0 sein sollte, geben sie `NULL` statt des prozentualen Anstiegs aus.

Lösung:

(a) Finden Sie für jedes *Land* den *Kunden*, auf den der höchsten Umsatz entfallen ist.

```
WITH kunden_gesamt_umsatz (laendercode, kundennr, umsatz) AS (
    SELECT kunden.laendercode, kunden.kundennr, SUM(gesamtpreis)
    FROM kunden JOIN bestellungen ON kunden.kundennr = bestellungen.
       kundennr
    GROUP BY kunden.laendercode, kunden.kundennr),
     max_umsaetze_land (laendercode, umsatz) AS (
         SELECT laendercode, MAX(umsatz)
         FROM kunden_gesamt_umsatz
         GROUP BY laendercode
     )

SELECT k.name, k.laendercode, kgu.umsatz
FROM kunden_gesamt_umsatz kgu, kunden k, max_umsaetze_land mul
WHERE kgu.kundennr = k.kundennr AND k.laendercode = mul.laendercode
    AND mul.umsatz = kgu.umsatz
ORDER BY laendercode;
```

(b) Finden Sie für jedes *Land* den *Kunden*, auf den der zweithöchste Umsatz entfällt. Der zweithöchste Umsatz ist definiert als der nächstkleinere Umsatz verglichen zum höchsten im jeweiligen *Land* angefallenen Umsatz.

```
WITH kunden_gesamt_umsatz (...)

select kgu.*
FROM kunden_gesamt_umsatz kgu
WHERE 1 = (SELECT COUNT(DISTINCT kgu_hoeher.umsatz)
           FROM kunden_gesamt_umsatz kgu_hoeher
           WHERE kgu_hoeher.laendercode = kgu.laendercode AND
               kgu_hoeher.umsatz > kgu.umsatz);
```

Eine alternative Lösung ohne korrelierter Unteranfrage kann so aussehen:

```
WITH kunden_gesamt_umsatz (...),
     hochste_umsaetze (laendercode, umsatz) AS (
         SELECT laendercode, MAX(umsatz)
         FROM kunden_gesamt_umsatz
         GROUP BY laendercode
      ),
     kunden_gesamt_umsatz_ohne_hoechste (laendercode, kundennr, umsatz
        ) AS (
         SELECT kgu.*
         FROM kunden_gesamt_umsatz kgu, hochste_umsaetze hu
         WHERE kgu.laendercode = hu.laendercode AND kgu.umsatz < hu.
            umsatz
      ),
     zweithochste_umsaetze (laendercode, umsatz) AS (
         SELECT laendercode, MAX(umsatz)
         FROM kunden_gesamt_umsatz_ohne_hoechste
```

```
        GROUP BY laendercode
    )

SELECT kguoh.*
FROM kunden_gesamt_umsatz_ohne_hoechste kguoh, zweithochste_umsaetze
    zhu
WHERE kguoh.laendercode = zhu.laendercode AND kguoh.umsatz = zhu.
    umsatz
ORDER BY laendercode;
```

Einfacher kann diese Anfrage unter Verwendung von Window-Functions aussehen - diese lernen Sie in der Vorlesung ERDB kennen:

```
WITH kunden_gesamt_umsatz (...),
kunden_gesamt_umsatzrang (laendercode, kundennr, umsatz, rang) AS (
    SELECT *
    FROM (
        SELECT *, RANK() OVER (PARTITION BY kgu.laendercode ORDER BY
            kgu.umsatz DESC) AS rang
        FROM kunden_gesamt_umsatz kgu
        ) tmp
    )

SELECT *
FROM kunden_gesamt_umsatzrang kgur
WHERE rang = 2;
```

(c) Ermitteln Sie die Exporte der einzelnen *Länder* und schlüsseln Sie diese nach Jahr und Importland auf.

```
SELECT l2k.name AS exporteur, lk.name AS importeur, EXTRACT(YEAR FROM
    b.datum) AS year, SUM(bp.preis)
FROM laender lk, kunden k, bestellungen b, bestellpositionen bp,
    lieferanten l2, laender l2k
WHERE lk.laendercode = k.laendercode AND k.kundennr = b.kundennr AND b
    .bestellnr = bp.bestellnr AND l2.lieferantennr = bp.lieferantennr
    AND l2.laendercode <> k.laendercode AND l2k.laendercode = l2.
    laendercode
GROUP BY lk.name, l2k.name, EXTRACT(YEAR FROM b.datum)
ORDER BY l2k.name;
```

(d) Geben Sie eine Übersicht, wie viele *Kunden n Bestellungen* getätig haben und schüsseln Sie diese nach *Kontinent* auf.
Abbildung 1 lässt eine Normalverteilung für die Anzahl der Bestellung pro Kunde in Europa vermuten, deren Mean bei 9 oder 10 zu sein scheint.

```
COPY
    (SELECT kontinent, bestellunganzahl, COUNT(*)
    FROM (SELECT ko.name AS kontinent, COUNT(*) AS bestellunganzahl
          FROM bestellungen b, kunden ku, laender lk, kontinente ko
```

```
        WHERE b.kundennr = ku.kundennr AND ku.laendercode = lk.
            laendercode AND ko.kontinentennr = lk.kontinentennr
        GROUP BY ko.name, b.kundennr) AS kontinente_bestellungen
    GROUP BY kontinent, bestellunganzahl
    ORDER BY kontinent, bestellunganzahl)
TO './bestellungen_verteilungen.csv' WITH CSV DELIMITER ',';
```

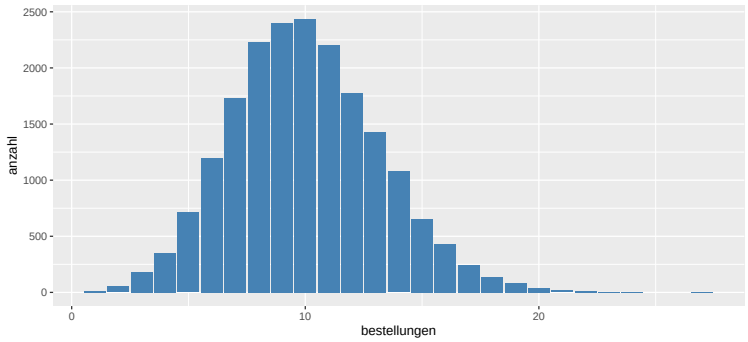

Abbildung 1: Anzahl der Bestellungen pro Kunde für Europa.

(e) Finden Sie die *Kunden*, die von mindesten fünf unterschiedlichen *Lieferanten*, die den Sitz im gleichen *Land* haben, *Artikel* erhalten haben, und die insgesamt einen Umsatz von 1000 € oder mehr erzielt haben (unabhängig vom Lieferanten).

```
SELECT k.kundennr
FROM kunden k, bestellungen b, bestellpositionen bp, lieferanten l
WHERE k.kundennr = b.kundennr AND b.bestellnr = bp.bestellnr AND bp.
    lieferantennr = l.lieferantennr AND l.laendercode = k.laendercode
GROUP BY k.kundennr
HAVING COUNT(DISTINCT l.lieferantennr) >= 5
INTERSECT
SELECT k.kundennr
FROM kunden k, bestellungen b, bestellpositionen bp
WHERE k.kundennr = b.kundennr AND b.bestellnr = bp.bestellnr
GROUP BY k.kundennr
HAVING SUM(bp.preis) >= 1000.00;
```

(f) Finden Sie die *Kunden*, die im Jahr 2018 *Artikel* von mindestens zwei *Lieferanten* a und b, die ihren Sitz in einem der *Länder* {GERMANY, BRAZIL, JAPAN, ROMANIA, UNITED KINGDOM, CANADA} haben, erhielten. Beachten Sie außerdem, dass a und b ihren Sitz in unterschiedlichen *Ländern* haben müssen.

```
WITH laender_gesucht (laendercode) AS (
    SELECT laendercode
    FROM laender
    WHERE name IN ('GERMANY', 'BRAZIL', 'JAPAN', 'ROMANIA', 'UNITED␣
        KINGDOM', 'CANADA')
)

SELECT DISTINCT k.*
```

```
FROM kunden k, bestellungen b, bestellpositionen bp, lieferanten l,
   bestellungen b2, bestellpositionen bp2, lieferanten l2
WHERE k.kundennr = b.kundennr AND
      EXTRACT(YEAR FROM b.datum) = 2018 AND
      b.bestellnr = bp.bestellnr AND
      bp.lieferantennr = l.lieferantennr AND
      l.laendercode IN (SELECT * FROM laender_gesucht) AND
      k.kundennr = b2.kundennr AND
      EXTRACT(YEAR FROM b2.datum) = 2018 AND
      b2.bestellnr = bp2.bestellnr AND
      bp2.lieferantennr = l2.lieferantennr AND
      l2.laendercode IN (SELECT * FROM laender_gesucht) AND
      l.laendercode <> l2.laendercode;
```

(g) Finden Sie für jeden Artikel und jedes Land den Lieferanten, der den günstigsten Lieferpreis verlangt.

```
WITH bereitstellen_laender (artikelnr, lieferantennr, lieferpreis,
   laendercode) AS
   (SELECT a.artikelnr, b.lieferantennr, b.lieferpreis, ln.
      laendercode
   FROM artikel a , bereitstellen b , liefertnach ln
   WHERE a.artikelnr = b.artikelnr AND ln.lieferantennr = b.
      lieferantennr
   GROUP BY a.artikelnr, b.lieferantennr, b.lieferpreis, ln.
      laendercode)

SELECT *
FROM bereitstellen_laender bl
WHERE NOT EXISTS (
   SELECT *
   FROM bereitstellen_laender bl2
   WHERE bl2.artikelnr = bl.artikelnr AND bl.laendercode = bl2.
      laendercode AND bl.lieferpreis > bl2.lieferpreis
   )
ORDER BY bl.artikelnr, laendercode;
```

(h) Finden Sie für jeden Artikel und jedes Land die Preisdifferenz des günstigsten und teuersten Angebots. Geben Sie auch den prozentualen Aufschlag vom günstigeren zum teureren Angebot mit zwei Nachkommastellen an.

```
WITH bereitstellen_laender (...),
     guenstigste_angebote (artikelnr, lieferantennr, lieferpreis,
         laendercode) AS (
     SELECT *
     FROM bereitstellen_laender bl
     WHERE NOT EXISTS (
        SELECT *
        FROM bereitstellen_laender bl2
        WHERE bl2.artikelnr = bl.artikelnr
```

```
        AND bl.laendercode = bl2.laendercode
        AND bl.lieferpreis > bl2.lieferpreis
    )),
    teuerste_angebote (artikelnr, lieferantennr, lieferpreis,
       laendercode) AS (
      SELECT *
             FROM bereitstellen_laender bl
             WHERE NOT EXISTS (
                    SELECT *
                    FROM bereitstellen_laender bl2
                    WHERE bl2.artikelnr = bl.artikelnr
                      AND bl.laendercode = bl2.laendercode
                      AND bl.lieferpreis < bl2.lieferpreis
          ))

SELECT g.*, t.lieferantennr, t.lieferpreis, t.lieferpreis - g.
   lieferpreis AS differenz, round( CAST(FLOAT8 (t.lieferpreis / g.
   lieferpreis) - 1 AS numeric), 2) AS prozentAufschlag
FROM guenstigste_angebote g, teuerste_angebote t
WHERE g.laendercode = t.laendercode AND g.artikelnr = t.artikelnr AND
   g.lieferantennr <> t.lieferantennr;
```

(i) Finden Sie die Kunden, die am meisten Geld hätten sparen können, wenn sie beim 'günstigsten' Lieferanten gekauft hätten.

```
WITH bereitstellen_laender (...), guenstigste_angebote (...)

SELECT k.name, SUM(be.lieferpreis - ga.lieferpreis)
FROM bestellpositionen bp, bereitstellen be, guenstigste_angebote ga,
   bestellungen b, kunden k
WHERE bp.artikelnr = be.artikelnr AND bp.lieferantennr = be.
   lieferantennr AND bp.bestellnr = b.bestellnr AND b.kundennr = k.
   kundennr AND ga.artikelnr = bp.artikelnr AND ga.laendercode = k.
   laendercode AND NOT EXISTS (
   SELECT *
   FROM guenstigste_angebote ga2
   WHERE ga2.artikelnr = bp.artikelnr AND ga2.lieferantennr = bp.
      lieferantennr AND ga2.laendercode = k.laendercode
   )
GROUP BY k.kundennr, k.name
ORDER BY SUM(be.lieferpreis - ga.lieferpreis);
```

(j) Finden Sie die Bestellpositionen, die laut der Relation liefertNach gar nicht möglich wären.

(1) Geben Sie **eine** SQL-Anweisung an, um eine neue Tabelle BestellpositionenKorrigiert zu erstellen, die das gleiche Schema und die gleichen Einträge wie die Tabelle Bestellpositionen enthält.

```
CREATE TABLE BestellpositionenKorrigiert AS (SELECT * FROM
   bestellpositionen);
```

(2) Updaten Sie nun die Tabelle BestellpositionenKorrigiert: Jede Position, bei der der Lieferant nicht das Land des Kunden beliefert, soll durch den günstigsten das Land des Kunden beliefernden Lieferanten ersetzt werden.

```
WITH bereitstellen_laender (...), guenstigste_angebote (...)

UPDATE BestellpositionenKorrigiert
SET lieferantennr = ga.lieferantennr
FROM bestellungen b, kunden k, guenstigste_angebote ga
WHERE BestellpositionenKorrigiert.bestellnr = b.bestellnr AND b.
    kundennr = k.kundennr AND ga.artikelnr =
    BestellpositionenKorrigiert.artikelnr AND ga.laendercode = k.
    laendercode AND NOT EXISTS (
    SELECT *
    FROM liefertnach ln
    WHERE ln.lieferantennr = BestellpositionenKorrigiert.
        lieferantennr AND ln.laendercode = k.laendercode
    );
```

(3) Ersetzen Sie in der Anfrage (i) Bestellpositionen durch BestellpositionenKorrigiert und überprüfen Sie, ob nur *nicht-negative* Einsparungen auftreten.

(k) Erörtern Sie unterschiedliche Möglichkeiten, wie man die Relation *Bestellpositionen* modifizieren könnte, damit solche fehlerhaften Daten nicht eingefügt werden können.

Um referentielle Integritäten in SQL abzubilden, sollte man primär *Fremdschlüssel* verwenden. Jedoch haben wir im vorliegenden Fall das Problem, dass die schwache Entität *Bestellpositionen* lediglich Lieferanten, Artikel und bereitstellen referenziert und die benötigte Information des Landes des Kunden daher nicht direkt in Bestellpositionen verfügbar ist. Denkbar wäre das Einführen des zusätzlichen Attributs *Zielland*, sodass *LieferantenNr* und *Zielland* zusammen einen kombinierten Fremdschlüssel auf *liefertNach* bilden. Jedoch würde dieses neue Attribut nur von einem Teil des Schlüssels abhängen und *Bestellpositionen* daher bereits die 2. Normalform verletzen, was wieder zu ANDeren Problemen und Inkonsistenzen führen kann.

Um solche komplexeren Constraints auszudrücken, kann man statt Fremdschlüsseln auch eine `check`-Anweisung in der Tabellen-Definition verwenden. Dies könnte so aussehen:

```
CREATE TABLE Bestellpositionen (
       bestellnr integer NOT NULL references bestellungen,
       ...
       constraint liefertNachExists check ( EXISTS
              (SELECT *
               FROM liefertNach ln
               WHERE ln.lieferantennr = lieferantennr
                 AND ln.laendercode IN (
                             SELECT k.laendercode
                             FROM bestellungen b, kunden k
                             WHERE b.kundennr = k.kundennr
                               AND b.bestellnr = bestellnr
                    )
```

```
            )
        )
);
```

Die oben neu eingeführte Bedingung *liefertNachExists* überprüft nun für jedes eingefügte Tupel, ob es einen Eintrag in *liefertNach* gibt, der ausdrückt, dass der *Lieferant* auch das *Land* des *Kunden* beliefert. Nur falls dieser Check erfolgreich ist, wird das neue, einzufügende Tupel der Relation *Bestellpositionen* tatsächlich hinzugefügt.

Jedoch werden Unteranfragen in `check` kaum von Datenbanksystemen unterstützt, sodass die obige Anfrage beispielsweise in Postgres so *nicht* formuliert werden kann.

Das Schema des Onlineshops unterstützt leider einige, wünschenswerte Constraints nicht auf einfachem Wege - beispielsweise könnten manche Artikel einer Altersbeschränkung unterliegen, sodass jüngere Kunden diese nicht bestellen dürfen. Auch diese Anforderung würde einen `check` mit Unteranfrage erfordern.

In der Praxis werden solche komplexeren Constraints meist in der Anwendungslogik implementiert, was jedoch den Nachteil mit sich bringt, dass bei Anwendungsfehlern inkonsistente Daten in der Datenbank gespeichert werden können.

(l) Ermitteln Sie Entwicklung der Marktanteile für Länder l innerhalb ihres Kontinents k für alle verschiedenen Artikeltypen an. Die Entwicklung soll auf Jahresbasis analysiert werden und auf den Preisen der Bestellpositionen beruhen. Dabei sollen nur die bestellten Waren berücksichtigt werden, die von Lieferanten mit Sitz in k geliefert wurden und deren Kunde ebenfalls in k beheimatet ist. Wenn der umgesetzte Wert im vorherigen Jahr 0 sein sollte, geben sie NULL statt des prozentualen Anstiegs aus.

```
WITH marktanteile_jahre (kontinentnr, laendercode, typ, summe, jahr)
    AS (
    SELECT ko.name, ll.name, a.typ, SUM(bp.preis), EXTRACT(YEAR FROM b
        .datum)
    FROM kontinente ko, lieferanten l, laender ll, artikel a,
       bestellpositionen bp, bestellungen b, kunden ku, laender lk
    WHERE ko.kontinentennr = ll.kontinentennr AND
          ko.kontinentennr = lk.laendercode AND
          ku.laendercode = lk.laendercode AND
          l.laendercode = ll.laendercode AND
          a.artikelnr = bp.artikelnr AND
          bp.bestellnr = b.bestellnr AND
          b.kundennr = ku.kundennr AND
          bp.lieferantennr = l.lieferantennr
    GROUP BY ko.name, ll.name, a.typ, EXTRACT(YEAR FROM b.datum)
    ),
     kontinent_gesamt (kontinentnr, typ, summe, jahr) AS (
         SELECT kontinentnr, typ, SUM(summe), jahr
         FROM marktanteile_jahre
         GROUP BY kontinentnr, typ, jahr
     ),
     jahresentwicklung (kontinentnr, laendercode, typ , jahr, summe,
         vorjahressumme, prozentualeveraenderung, marktanteil,
         vorjahresmarktanteil) AS
```

```
        (SELECT m.kontinentnr, m.laendercode, m.typ, m.jahr, m.summe,
               COALESCE (v.summe, '0.0'::FLOAT8::NUMERIC),
               CASE WHEN (v.summe is NOT NULL AND v.summe > '0.0'::
                   FLOAT8::NUMERIC) THEN m.summe / v.summe ELSE NULL
                   END,
               CASE WHEN (k.summe is NOT NULL AND k.summe > '0.0'::
                   FLOAT8::NUMERIC) THEN m.summe / k.summe ELSE NULL
                   END,
               CASE WHEN (v.summe is NOT NULL AND v.summe > '0.0'::
                   FLOAT8::NUMERIC AND kv.summe is NOT NULL AND kv.
                   summe > '0.0'::FLOAT8::NUMERIC ) THEN v.summe / kv.
                   summe ELSE NULL END
         FROM marktanteile_jahre m
               JOIN kontinent_gesamt k ON k.kontinentnr = m.
                  kontinentnr AND k.jahr = m.jahr AND k.typ = m.typ
               LEFT OUTER JOIN marktanteile_jahre v ON m.typ = v.typ
                  AND m.laendercode = v.laendercode AND m.jahr = (v.
                  jahr+1)
               LEFT OUTER JOIN kontinent_gesamt kv ON kv.kontinentnr =
                   m.kontinentnr AND (kv.jahr + 1) = m.jahr AND kv.typ
                   = m.typ
         )
SELECT *
FROM jahresentwicklung
WHERE vorjahresmarktanteil is NOT NULL
ORDER BY marktanteil DESC;
```