# Einführung in PL/SQL

Günter Burgstaller

2022/12/01 10:07

# Inhaltsverzeichnis

**Inhalt**

# Einführung

PL/SQL (PL = *Procedural Language*) ist die prozedurale Spracherweiterung zu SQL von Oracle zur Programmierung des Oracle Datenbankservers. Die prozedurale Erweiterung *Transact-SQL* (T-SQL) ist die Entsprechung beim Microsoft SQL Server. Sie wird unter anderem zur Entwicklung von *gespeicherten Prozeduren*, *Packages* oder von *Triggern* verwendet.

Ein *Block PL/SQL Code* besteht aus drei Teilen:

- *Deklarationen*: Definiert und initialisiert Variablen und Cursor für den Block
- *Anweisungen*: Kommandos und Steuerlogik, die ausgeführt werden
- *Ausnahmebehandlung*: Behandlung von Laufzeitfehlern (*Exceptions*).

Eine *Stored Procedure* ist ein Beispiel für einen benannten Codeblock. Codeblöcke ohne Namen heißen *anonyme Codeblöcke*.

## PL/SQL Blockstruktur

Struktur eines PL/SQL Blocks:

```
declare
  <declarations section>
begin
  <executable commands>
exception
  <exception handling>
end;
```

Es kann sein, dass Sie ein abschließendes / benötigen, um den Block auszuführen. Das hängt vom verwendeten Clientwerkzeug ab, mit dem Sie sich zur Datenbank verbunden haben.

Einfachstes Beispiel:

```
begin
 null;
end;
/
```

## Konsolenausgabe

Zur Konsolenausgabe wird die gespeicherte Prozedur dbms_output.put_line (das Package heißt dbms_output) verwendet, anlog zu print beim *SQL Server*. Die Ausgabe wird jedoch nur sichtbar, wenn Sie zuvor set serveroutput on size unlimited aktivieren.

```
begin
  dbms_output.put_line('Hello Oracle.');
end;
/
```

Bearbeiten Sie Übung 1.

# Deklarationen

Der *Deklarationsabschnitt* beginnt mit dem Schlüsselwort declare, gefolgt von einer Liste von Variablen- bzw. Cursor-Definitionen. Folgender Block zur Kreisflächenberechnung zeigt die Möglichkeiten:

```
clear screen
declare
  pi constant number(9, 7) := 3.1415927;
  radius integer;
  area number(14, 2);
begin
  radius := 3;
  area := pi * power(radius, 2);
```

```
 insert into areas values(radius, area);
 commit;
end;
/
show errors
```

Zu beachten:

- Schlüsselwort constant für Variablen, deren Wert nicht mehr änderbar ist.
- Wertzuweisungen erfolgen über :=.
- Wertzuweisungen in der Deklaration auch über default möglich: pi constant number(9, 7) default 3.1415927

Bearbeiten Sie Übung 2.

# Cursordeklarationen

Cursordefinitionen erfolgen ebenfalls im Deklarationsabschnitt. Beispiel:

```
clear screen
declare
  pi constant number(9, 7) := 3.1415927;
  area number(14, 2);

 cursor radius_cursor is
   select * from radius_values;
  radius_row radius_cursor%rowtype;
begin
 open radius_cursor;
 fetch radius_cursor into radius_row;

  area := pi * power(radius_row.radius, 2);
 insert into areas values (radius_row.radius, area);

 close radius_cursor;
 commit;
end;
/
show errors
```

Bearbeiten Sie Übung 3.

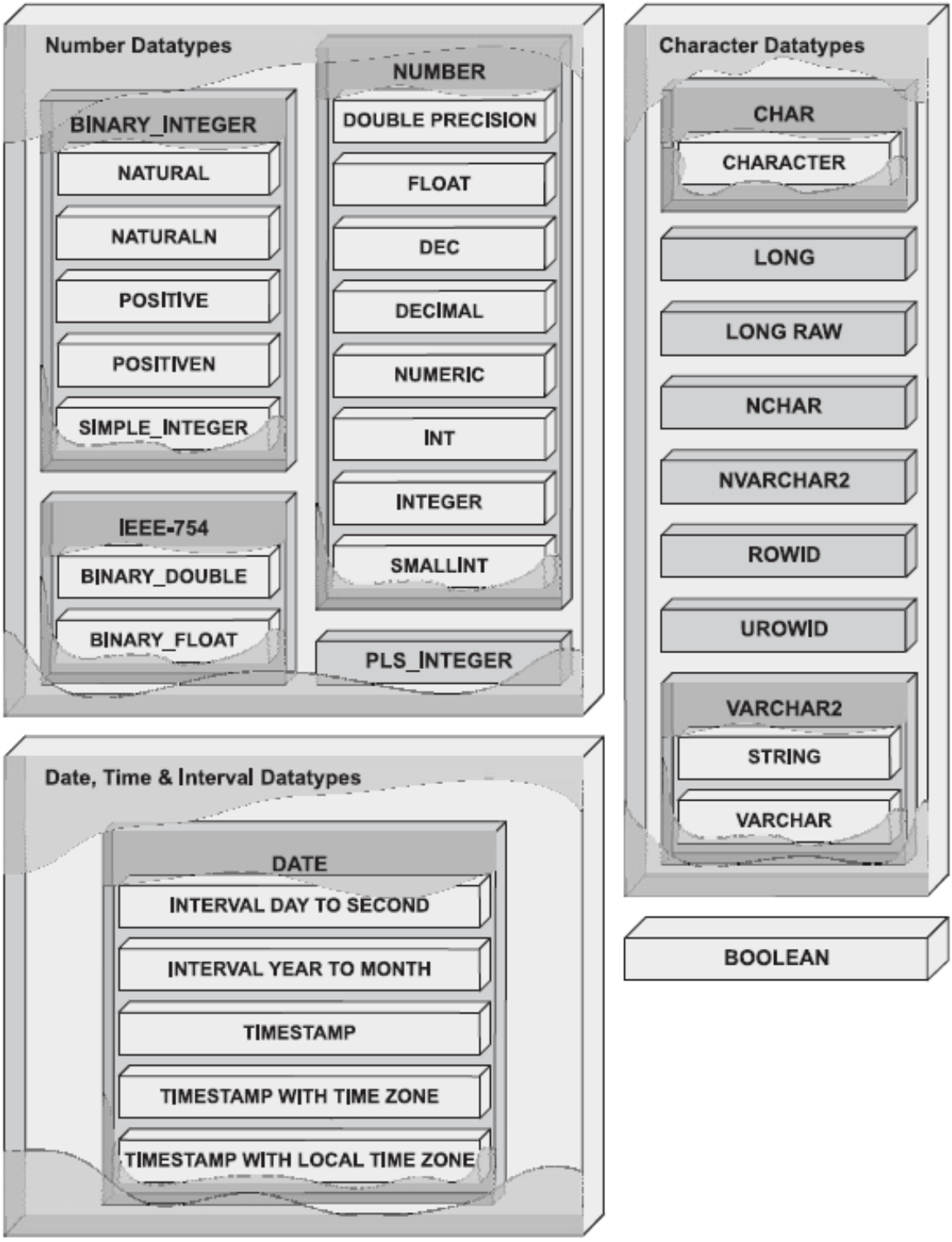
Die Bezeichner %rowtype und %type können dazu verwendet werden, um den Datentyp einer PL/SQL-Variable an die Datentypen von Tabellenspalten zu knüpfen. Das hat zwei Vorteile: man muss zum einen nicht nachsehen, wie eine Tabellenspalte definiert ist, zum anderen muss der PL/SQL-Code zumeist nicht geändert werden, wenn sich Tabellendefinitionen ändern.

Syntax %TYPE

# PL/SQL Datentypen

Neben einigen speziellen Datentypen unterstützt PL/SQL alle Datentypen, die der Oracle Datenbankserver auch in SQL erlaubt.

## Skalare Datentypen




Quelle: *Oracle Database 11g PL/SQL Programming, Oracle Press, p. 64*

## Sequences

Eine *Sequence* ist kein Datentyp, sondern ein *Datenbankobjekt*. Sie stellt einen automatischen Zähler zur Verfügung, der Werte liefert, die für eindeutige Felder, insbesonders also für Schlüsselfelder, verwendet werden können. Beispiel:

```
create sequence EmployeeIDSeq
start with 10 increment by 5
cache 100 nocycle;
insert into Employee
  (EmployeeID          , EmployeeName    )
values
  (EmployeeIDSeq.nextval, 'Edgar F. Codd')
;
insert into Employee
  (EmployeeID          , EmployeeName      )
values
  (EmployeeIDSeq.nextval, 'Raymond F. Boyce')
;
commit;
select *
 from Employee;
select EmployeeIDSeq.currval
 from dual;
```

Syntax CREATE SEQUENCE

Die Klausel CACHE 100 bewirkt, dass 100 Werte als Vorrat generiert werden. Das bringt Performance-Vorteile. Bei einem Absturz des Datenbankservers gehen jedoch alle bevorrateten Werte verloren. Der nächste Wert ist dann der nächst höhere nach dem höchsten Vorratswert.

Beim *SQL Server* werden automatische Zähler durch den Zusatz IDENTITY realisiert. Beispiel mit Beginnwert 10 und Schrittweite 5:

```
create table Employee
(
  EmployeeID int identity(10, 5),
  EmployeeName varchar(32)
);
```

Bearbeiten Sie Übung 17.

## Zusammengesetzte Datentypen

In PL/SQL können folgende zusammengesetzte Datentypen verwendet werden:

- Record
- Varray
- Nested Table
- Associative Array

Zusammengesetzte Datentypen werden in einem späteren Schwerpunktthema im Detail behandelt.

## Referenztypen

Referenztypen werden im Rahmen dieses Unterrichtsfaches nicht behandelt. Interessierte seien auf die Oracle PL/SQL-Handbücher verwiesen.

## Large Object Block (LOB) Typen

*Large Object Block (LOB)-Typen* dienen zur Speicherung von sehr großen Datenmengen innerhalb eines Datensatzes. Beispiel: eine Tabelle, die Buchinformationen wie Autor, Titel sowie den *gesamten Text* des Buches enthält. Oracle unterstützt folgende LOB-Typen:

| | |
|---|---|
| bfile | Zeigt auf eine externe Datei mit max. 4 GB |
| blob | Les- und schreibbare binäre Daten |
| clob | Les- und schreibbare Zeichendaten |
| nclob | Les- und schreibbare Unicode-Zeichendaten |

LOB-Typen sind in der Handhabung komplizierter als andere Datentypen. In der Regel muss das Package DBMS_LOB verwendet werden. Zu den LOB-Typen gibt es später ein eigenes Schwerpunktthema.

# Anweisungen

Der Anweisungsteil beginnt immer mit dem Schlüsselwort begin.

Im Anweisungsteil stehen einerseits Anweisungen, die deklarierte Variablen ändern:

```
begin
  radius := 3;
  area := pi * power(radius, 2);
...
end;
```

Weiters Befehle, die einen deklarierten Cursor verwenden:

```
begin
 open radius_cursor;
 fetch radius_cursor into radius_row;
  ...
 close radius_cursor;
end;
```

Sowie SQL-Kommandos:

```
begin
  ...
 insert into areas values(radius, area);
 commit;
end;
```

Natürlich können dort auch bedingte Anweisungen und Schleifen stehen.

## Bedingte Anweisungen

Der allgemeine Aufbau einer if-Anweisung ist folgendermaßen:

```
if sales > 50000 then
  bonus := 1500;
elsif sales > 35000 then
  bonus := 500;
else
  bonus := 100;
end if;
```

Wie in anderen Programmiersprachen können if-Anweisungen verschachtelt werden.

Bearbeiten Sie Übung 4.

## Schleifen

PL/SQL kennt drei Arten von Schleifen:

- *Einfache Schleifen* mit loop: werden solange wiederholt, bis eine exit oder exit when Anweisung erreicht wird.
- for Schleifen: eine festgelegte Anzahl an Wiederholungen wird durchlaufen.
- while Schleifen: Wiederholung bis eine Bedingung erfüllt ist.

## Einfache Schleifen

*Einfache Schleifen* verwenden das Schlüsselwort loop und sind *Endlosschleifen*. Sie werden durch eine exit oder exit when Klausel verlassen. Beispiel:

```
truncate table areas;
declare
  l_pi constant number(9, 7) default 3.1415927;
  l_radius integer(5);
  l_area number(14, 2);
begin
  l_radius := 3;

  loop
    l_area := l_pi * l_radius * l_radius;
   insert into areas values(l_radius, l_area);
    l_radius := l_radius + 1;

    exit when l_area > 100;
 end loop;
 commit;
end;
/
select *
 from areas;
```

Ergebnis:

```
RADIUS               AREA
---------------------- ----------------------
3                    28,27
4                    50,27
5                    78,54
6                    113,1
```

Bearbeiten Sie Übung 5.

Syntax LOOP (Oracle)

## Einfache Cursorschleifen

Cursor haben vier Attribute, die in Programmen verwendet werden können:

| Attribut | Bedeutung |
|---|---|
| %found | Ein nächster Datensatz ist verfügbar |
| %notfound | Keine Datensätze mehr vorhanden |
| %isopen | Der Cursor ist geöffnet |
| %rowcount | Anzahl der bisher gelesenen Datensätze |

Die Attribute nehmen den Wert true oder false an. Dadurch sind Ausdrücke wie exit when radius_cursor%notfound; möglich.

Bearbeiten Sie Übung 6.

## for Schleifen

Eine for Schleife wird eine festgelegte Anzahl an Wiederholungen durchlaufen und verwendet die Schlüsselwörter for ... loop ... end loop. Beispiel:

```
truncate table areas;
declare
  pi constant number(9, 7) default 3.1415927;
  radius integer(5);
  area number(14, 2);
```

```
begin
 for radius in 1..7 loop
   area := pi * power(radius, 2);
  insert into areas values(radius, area);
 end loop;
 commit;
end;
/
select *
 from areas;
```

| RADIUS | AREA |
|---|---|
| 1 | 3,14 |
| 2 | 12,57 |
| 3 | 28,27 |
| 4 | 50,27 |
| 5 | 78,54 |
| 6 | 113,1 |
| 7 | 153,94 |

Syntax FOR LOOP (Oracle)

Bearbeiten Sie Übung 7.

### for Cursorschleifen

Eine Cursor for Schleife durchläuft alle Ergebnisdatensätze einer Abfrage. Das *Öffnen* (open), Holen eines Datensatzes (fetch) und Schließen (close) des Cursors geschieht *automatisch (implizit)*.

```
truncate table areas;
declare
  pi constant number(9, 7) default 3.1415927;
  area number(14, 2);

 cursor radius_cursor is
   select *
    from radius_values;

begin
 for radius_row in radius_cursor loop
   area := pi * power(radius_row.radius, 2);
  insert into areas values(radius_row.radius, area);
 end loop;
 commit;
end;
/
select *
 from areas;
```

Beispielergebnis:

| RADIUS | AREA |
|---|---|
| 3 | 28,27 |
| 6 | 113,1 |
| 10 | 314,16 |
| 15 | 706,86 |
| 21 | 1385,44 |
| 28 | 2463,01 |
| 36 | 4071,5 |

Syntax Cursor FOR LOOP (Oracle)

Bearbeiten Sie Übung 8.

### while Schleifen

Eine while Schleife hat eine Eingangsbedingung. Die Schleife wird so lange durchlaufen, als die Eingangsbedingung erfüllt ist. Beispiel:

```
set serveroutput on size unlimited
declare
  random_value number;
  counter integer := 0;
begin
  random_value := dbms_random.value();

  while random_value < 0.8
  loop
    counter := counter + 1;
    random_value := dbms_random.value();
 end loop;
  dbms_output.put_line(counter);
end;
/
```

Syntax WHILE LOOP (Oracle)

Bearbeiten Sie Übung 9.

### Verwendung von continue und continue when

Der Befehl continue innerhalb einer Schleife beendet den aktuellen Durchlauf und springt zum Anfang der Schleife, d.h. startet die nächste Wiederholung. Beispiel:

```
truncate table areas;
declare
  pi constant number(9, 7) default 3.1415927;
  radius integer(5);
  area number(14, 2);
begin
  radius := 0;

  loop
    radius := radius + 1;

    if radius < 5 then
      continue;
    end if;

    area := pi * power(radius, 2);
   insert into areas values(radius, area);

    exit when area > 100;
 end loop;
 commit;
end;
/
select *
 from areas;
```

Ergebnis:

```
RADIUS               AREA
---------------------- ----------------------
```

5 78,54
6 113,1

Bei der Anweisung continue when kann die umschließende if Bedingung wegfallen.

Syntax CONTINUE (Oracle)

Bearbeiten Sie Übung 10.

# Bedingte Anweisungen mit case

Im Gegensatz zu T-SQL kann in PL/SQL das Schüsselwort case sowohl als Anweisung im Programmcode stehen als auch in Ausdrücken und SQL Befehlen verwendet werden.

## case als Ausdruck

Beispiel:

```
select short_signature
    , case short_signature
      when 'CY' then 'Cyrmon, Werner'
      when 'DL' then 'Dellinger, Franz'
      when 'EC' then 'Eberl, Christine'
      when 'EK' then 'Ecker, Gottfried'
      when 'ED' then 'Edelmoser, Karl'
      else 'Unbekannt'
    end "full_name"
 from initials
;
```

Bearbeiten Sie Übung 11.

Syntax CASE Ausdrücke

## case als Anweisung

Beachten Sie, dass die Anweisung mit end case statt nur end abgeschlossen werden muss. Die Teilanweisungen müssen mit ; abgeschlossen werden.
Bearbeiten Sie Übung 12.

Beispiel:

```
declare
  i integer;
begin
 for i in 1..7 loop
   case
     when i = 1 then dbms_output.put_line('Montag: 8-12 und 13-18 Uhr.');
     when i = 2 then dbms_output.put_line('Dienstag: 8-13 Uhr.');
     when i = 3 then dbms_output.put_line('Mittwoch: 9-12 Uhr und 14-20 Uhr.');
     when i = 4 then dbms_output.put_line('Donnerstag: 9-13 Uhr.');
     when i = 5 then dbms_output.put_line('Freitag: 10-15 Uhr.');
     when i = 6 then dbms_output.put_line('Samstag: 7-11 Uhr.');
     when i = 7 then dbms_output.put_line('Sonntag: geschlossen.');
   end case;
 end loop;
end;
/
```

Wenn kein else-Zweig angegeben ist, fügt PL/SQL automatisch else raise case_not_found; hinzu.
Bearbeiten Sie Übung 13.

Syntax CASE Anweisungen

# Ausnahmebehandlung

## Abfangen von Laufzeitfehlern

Bei Auftreten eines *Laufzeitfehlers (Exception)* wird die Kontrolle an den Ausnahmebehandlungsteil übergeben. PL/SQL stellt eine Reihe von Standardlaufzeitfehlern (Oracle) zur Verfügung. Weitere Laufzeitfehler können vom Programmierer definiert werden. Der Ausnahmebehandlungsteil beginnt mit dem Schlüsselwort exception.

Zur Wiederholung die Struktur eines PL/SQL Blocks:

```
declare
  <declarations section>
begin
  <executable commands>
exception
  <exception handling>
end;
```

Der Ausnahmebehandlungsteil ist nicht verpflichtend. Nicht behandelte Laufzeitfehler führen zum Abbruch der PL/SQL-Abarbeitung. DML-Vorgänge werden zurückgerollt.

Beispiel:

```
set serveroutput on size unlimited
declare
 result integer;
begin
 result := 2012 / 0;
exception
 when zero_divide then
   dbms_output.put_line('Black holes are where God divided by zero. (A. Einstein)');
end;
/
```

Beim Auftreten eines Laufzeitfehlers wird der Ausnahmebehandlungsteil nach einer passenden when-Anweisung durchsucht. Wird keine gefunden, wird der Laufzeitfehler an den umgebenden Block weitergereicht.

Mit der when others-Klausel können beliebige Laufzeitfehler abgefangen werden (Wildcard).
Bearbeiten Sie Übung 14.

## Laufzeitfehler auslösen

Laufzeitfehler können im PL/SQL-Code, wie bereits in vorhergegangen Beispielen ersichtlich, mit dem Befehl raise ausgelöst werden. Im Ausnahmebehandlungsteil kann der Laufzeitfehlername entfallen, um nach einer Fehlerbehandlung den Laufzeitfehler trotzdem an den umgebenden Codeblock weiterzureichen.

Syntax RAISE

## Benutzerdefinierte Laufzeitfehler

Der PL/SQL-Programmierer kann sich auch eigene Laufzeitfehler deklarieren. Beispiel:

```
declare
  power_failure exception;
begin
  raise power_failure;
exception
 when power_failure then
   dbms_output.put_line('*** Stromausfall ***');
   dbms_output.put_line('Systemabschaltung nach 5 Minuten Akku-Betrieb.');
end;
```

/

Syntax Exception-Deklaration

Bearbeiten Sie Übung 15.

### Verschachtelte Blöcke

Sobald die Kontrolle an den Ausnahmebehandlungsblock übergegangen ist, kann der Programmfluss *nicht mehr* an der Stelle fortgesetzt werden, an der der Laufzeitfehler aufgetreten ist. Wenn das gewünscht wird, muss die Operation, die zu einem Laufzeitfehler führt, mit einem eigenen PL/SQL-Block umgeben werden.

Bearbeiten Sie Übung 16.

# Literatur

- Oracle Database PL/SQL Language Reference 18c
- Feuerstein, Steven: "Oracle PL/SQL Programming

# Übungen

## Übung 1

1. Erstellen Sie einen Tablespace und ein Benutzerkonto mit Ihrer Matrikelnummer.
2. Verbinden Sie sich mit diesem Benutzerkonto zur Datenbank.
3. Führen Sie den Block *Hello Oracle* aus, ohne die Serverausgabe zu aktivieren.
4. Aktivieren Sie die Serverausgabe mit set serveroutput on size unlimited.
5. Führen Sie den Block nochmals aus. Worin besteht der Unterschied?

## Übung 2

1. Erstellen Sie die Tabelle areas mit geeigneten Datentypen.
2. Testen Sie den anonymen Block zur Kreisflächenberechnung.
3. Fragen Sie die Tabelle areas ab, um zu testen, ob der Block wie erwartet funktioniert hat.

## Übung 3

1. Erstellen Sie die Tabelle radius_values mit geeigneten Datentypen.
2. Fügen Sie einen Datensatz mit beliebigem Radius ein.
3. Testen Sie den anonymen Block zur Kreisflächenberechnung mit Cursor.
4. Fragen Sie die Tabelle areas ab, um zu testen, ob der Block wie erwartet funktioniert hat.
5. Erklären Sie die Funktion von %rowtype.
6. Ersetzen Sie den PL/SQL-Block durch ein INSERT kombiniert mit SELECT, das die gleiche Änderung bewirkt. Testen Sie das Kommando.
7. Ersetzen Sie den PL/SQL-Block und das Anlegen der Tabelle areas, indem Sie das Kommando create table areas_ as select... verwenden. Sehen Sie sich mit desc areas_ die Struktur der erzeugten Tabelle an.

## Übung 4

1. Erweitern Sie das Kreisberechnungsbeispiel (mit Cursor), indem Sie nur dann den Datensatz einfügen, wenn die Fläche größer als 30 ist.
2. Schreiben Sie zwei Testfälle, bei denen das einmal der Fall ist und einmal nicht.
3. Fragen Sie die Tabelle areas jeweils ab, um zu prüfen, ob der Block wie erwartet funktioniert hat.

## Übung 5

1. Testen Sie das Beispielprogramm.
2. Ändern Sie die Erhöhung des Radius auf Zehnerschritte. Setzen Sie den Startwert auf 10.
3. Der letzte Radiuswert in der Tabelle areas soll 1000 sein.

4. Testen Sie das geänderte Programm.

## Übung 6

1. Leeren Sie die Tabelle radius_values und fügen Sie folgende Datensätze ein: 3 6 10 15 21 28 36.
2. Ändern Sie den Block aus Übung 3, sodass alle Datensätze aus radius_values abgearbeitet werden.
3. Überprüfen Sie das Ergebnis in areas.
4. Ersetzen Sie den PL/SQL-Block durch eine SQL-Anweisung, die das Gleiche bewirkt (INSERT mit SELECT).

Erwartetes Ergebnis:

```
RADIUS               AREA
---------------------- ----------------------
3                    28,27
6                    113,1
10                    314,16
15                    706,86
21                    1385,44
28                    2463,01
36                    4071,5
```

## Übung T-SQL

1. Führen Sie folgenden funktionierenden T-SQL-Code im SQL Server aus, indem Sie die Variable @Pwd einmal mit topsecret, einmal mit TopSecret2120 und zuletzt mit Ihrer Matrikelnummer s99999999 belegen.

```
set nocount on
begin
  declare @Pwd varchar(32) = 'topsecret';
  declare @Length int = len(@Pwd);
  declare @Char char;
  declare @IsLongEnough bit = 1;
  declare @HasUpper bit = 0;
  declare @HasLower bit = 0;
  declare @HasDigit bit = 0;

   if @Length < 8
  begin
     set @IsLongEnough = 0;
  end;

   while @Length > 0 and (@HasUpper & @HasLower & @HasDigit = 0)
  begin
     set @Char = substring(@Pwd, @Length, 1);

      if @HasUpper != 1
     begin
        set @HasUpper = charindex(@Char
                          , 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' collate Latin1_General_CS_AS);
     end;

      if @HasLower != 1
     begin
        set @HasLower = charindex(@Char
                          , 'abcdefghijklmnopqrstuvwxyz' collate Latin1_General_CS_AS);
     end;

      if @HasDigit != 1
     begin
        set @HasDigit = charindex(@Char, '0123456789');
```

```
    end;

    set @Length = @Length - 1;
  end;

  select @IsLongEnough & @HasUpper & @HasLower & @HasDigit "IsSecure?";
end;
go
```

2. Übersetzen Sie den T-SQL Block nach PL/SQL.

3. Testen Sie den PL/SQL-Block und zeigen Sie, dass er das gleich Ergebnis wie der T-SQL-Block liefert, indem Sie ihn mit den gleichen Werten (topsecret, TopSecret2120, Matrikelnummer s99999999) aufrufen.

Lösungsvorschlag

# Übung 7

1. Testen Sie das Beispielprogramm.
2. Ändern Sie das Beispielprogramm, sodass radius und area nicht in die Tabelle areas eingefügt werden, sondern auf der Konsole ausgegeben werden.
3. Sehen Sie sich die Syntax FOR LOOP an.
4. Ändern Sie das Beispielprogramm, sodass die Radiuswerte absteigend durchlaufen und ausgegeben werden.

Erwartetes Ergebnis:

7 153,94
6 113,1
5 78,54
4 50,27
3 28,27
2 12,57
1 3,14
Zeichenketten werden bei Oracle mit dem Operator || verknüpft.

# Übung 8

1. Stellen Sie sicher, dass die Tabelle radius_values einige Datensätze enthält.
2. Testen Sie das Beispielprogramm.
3. Sehen Sie sich die Syntax Cursor FOR LOOP an. Aus dem Syntaxdiagramm geht hervor, dass die *Deklaration eines Cursornamens* auch noch entfallen kann.
4. Ändern Sie das Beispielprogramm, indem Sie einen *anonymen Cursor* verwenden, also ohne explizite Cursordeklaration.
5. Ersetzen Sie den PL/SQL-Block durch ein INSERT kombiniert mit SELECT, das die gleiche Änderung bewirkt. Testen Sie das Kommando.
6. Ersetzen Sie den PL/SQL-Block und das Anlegen der Tabelle areas, indem Sie das Kommando create table areas_ as select... verwenden. Sehen Sie sich mit desc areas_ die Struktur der erzeugten Tabelle an.

# Übung 9

## Übung 9a

1. Sehen Sie sich die Syntax WHILE LOOP an.
2. Ändern Sie das Beispielprogramm von Übung 5, indem Sie eine while Schleife verwenden.

## Übung 9b

1. Testen Sie das Beispielprogramm mit der Zufallszahlenbedingung.
2. Ändern Sie es so, sodass es eingebettet in eine while Schleife 99 Mal ausgeführt wird.
3. Die zufallsbedingte Zahl an Wiederholungen soll jeweils durch ein Leerzeichen getrennt ausgegeben werden. Verwenden Sie put für eine Ausgabe ohne Zeilenumbruch statt put_line. Achtung: Sie benötigen ein abschließendes put_line(''), damit die Ausgabe von put sichtbar wird.

Beispielergebnis:

1 1 4 3 0 0 3 10 5 2 1 4 3 1 1 5 4 ... 26 5 7 7 7 6 6 5 1 5 1 0 3 1 5 1 3 0 6 1 3 9 4

## Übung: Sieb des Eratosthenes

1. Lesen Sie den Wikipedia-Artikel über das Sieb des Eratosthenes.
2. Erstellen Sie eine Tabelle primes mit dem Feld v_value integer PK.
3. Befüllen Sie die Tabelle mit den Werten 2 bis 1000.
4. Erstellen Sie einen PL/SQL-Block, der nach dem Sieb des Eratosthenes alle Zahlen aus der Tabelle primes entfernt, die keine Primzahlen sind.
5. Testen Sie den Block und bestimmen Sie die Anzahl der in primes enthaltenen Primzahlen.
6. Schreiben Sie eine Abfrage, die ermittelt, wieviele davon *palindromische Primzahlen* sind.
7. Ermitteln Sie mit set timing on und set timing off, wie lange ihr Programm für die Zahlenmengen 1000, 10.000, 100.000 und 1.000.000 benötigt. Bestimmen Sie jeweils die Anzahl der Primzahlen sowie der palindromischen Primzahlen. Geben Sie die fünf größten Primzahlen der Menge aus.

Die Ausgabe eines Durchlaufs soll, hier am Beispiel von 100.000, wie folgt aussehen:

```
Prime numbers <= 100.000
------------------------
Elapsed: 00:00:01.71
# primes <= 10^5
----------------
          9592
# palindromic primes <= 10^5
----------------------------
                     113
5 largest primes
----------------
         99991
         99989
         99971
         99961
         99929
```

Lösungsvorschlag

## Übung 10

1. Testen Sie den Beispielblock.
2. Lesen Sie die Syntax CONTINUE durch.
3. Ändern Sie den Beispielblock um, sodass die if Anweisung durch ein continue when ersetzt wird.
4. Testen Sie den geänderten Block.

## Übung 11

1. Sehen Sie sich das Codebeispiel im Text an.
2. Erstellen Sie eine geeignete Tabelle initials (mit Primärschlüssel), sodass das Codebeispiel ohne Änderung fehlerfrei ausführbar ist.
3. Befüllen Sie die Tabelle initials mit geeigneten Werten, sodass alle Fälle getestet werden.

## Übung 12

Gegeben ist folgender fehlerhafter PL/SQL Programmcode:

```
declare
  grade integer;
begin
  grade := 3;
```

```
 case grade
   when 1 then dbms_output.put_line('Sehr gut')
   when 2 then dbms_output.put_line('Gut')
   when 3 then dbms_output.put_line('Befriedigend')
   when 4 then dbms_output.put_line('Genügend')
   when 5 then dbms_output.put_line('Nicht genügend')
 end;
end;
/
```

Beheben Sie die enthaltenen Fehler und führen Sie den korrigierten Code aus.

## Übung 13

1. Probieren Sie das Codebeispiel im Text aus.
2. Ändern Sie das Beispiel, sodass die Schleife bis 8 läuft.
3. Was passiert, wenn i den Wert 8 enthält? Warum?
4. Ändern Sie das Beispiel, sodass bei ungültigen Werten kein Laufzeitfehler auftritt.
5. Ändern Sie das Beispiel, sodass CASE als *Ausdruck* und nicht als *Anweisung* verwendet wird.

## Übung 14

1. Testen Sie das vorangegangene Codebeispiel im Text *ohne* Ausnahmebehandlungsteil.
2. Testen Sie das Codebeispiel mit Ausnahmebehandlungsteil.
3. Führen Sie eine DML-Operation ohne commit durch. Führen Sie anschließend den PL/SQL-Block nochmals aus. Ist die Änderung noch vorhanden?
4. Ersetzen Sie den Null-Divisor durch einen gültigen Wert.
5. Führen Sie den Test mit der DML-Operation nochmals durch.
6. Lösen Sie mit raise program_error im Anweisungsteil einen Laufzeitfehler aus.
7. Erweitern Sie den Ausnahmenbehandlungsteil, sodass alle übrigen Laufzeitfehler abgefangen werden mit der Meldung 'Errare humanum est.'.
8. Testen Sie Ihre Erweiterung.

## Übung 15

1. Testen Sie das Codebeispiel im Text.
2. Erweitern Sie das Beispiel um einen weiteren benutzerdefinierten Laufzeitfehler ups_low_batteries.
3. Behandeln Sie den Fehler durch Ausgabe der Meldung:

*** USV Batterien schwach ***
System wird heruntergefahren ...

## Übung 16

1. In nachfolgendem Code wird mit raise program_error; eine Fehlersituation simuliert, bei der bei jedem Schleifendurchlauf ein Laufzeitfehler auftritt. Probieren Sie das Codebeispiel aus:

```
clear screen
set serveroutput on size unlimited
set feedback off
declare
  i integer;
begin
 for i in 1..5
  loop
    dbms_output.put_line(i);
    raise program_error;
 end loop;
end;
/
```

2. Ändern Sie das Codebeispiel, sodass die Schleife weiterläuft und bei jedem Laufzeitfehler die Meldung Error logged, operation continues ... ausgegeben wird.

# Übung 17

1. Erstellen Sie mit dem im *Oracle SQL Developer* integrierten *Data Modeler* eine Tabelle Employee mit folgenden Feldern:

   - EmployeeID integer PK
   - EmployeeName varchar(32)

2. Generieren Sie das relationale Modell und den DDL-Code.

3. Führen Sie den DDL-Code in Ihrer Datenbank aus.

4. Erstellen Sie die Sequence EmployeeIDSeq wie im Beispiel von Abschnitt Sequences.

5. Führen Sie die INSERT-Operationen aus. Welchen aktuellen Wert hat nun die Sequenz?

6. Stoppen Sie die Datenbank durch shutdown abort und starten Sie sie wieder.

7. Fragen Sie den nächsten Wert der Sequenz ab. Was ist ungewöhnlich an dem Wert? Erklären Sie die Ursache.