PL/SQL

- prozedurale Erweiterung von SQL
- Oracle-Produkt, nicht genormt!
- nicht für Ein-/Ausgaben gedacht

Beispiel:

```
PROCEDURE ueberweisen( x NUMBER, y NUMBER, betrag NUMBER) IS
s konto.kontostand%TYPE; - - Deklarationsteil
BEGIN
SELECT kontostand INTO s FROM konto
WHERE kontonr = x FOR UPDATE OF kontostand;

IF s < betrag THEN
INSERT INTO abgelehnt VALUES (SYSDATE,x,y,betrag);
ELSE
s := s-betrag;
UPDATE konto
SET kontostand = s WHERE kontonr = x;
UPDATE konto
SET kontostand = kontostand+betrag WHERE kontonr = y;

INSERT INTO gebucht
VALUES (SYSDATE,x,y,betrag);
END IF;
COMMIT;
END;
```

PL/SQL

- block-orientiert
- Grundeinheiten: namenloser Block, Prozedur, Funktion
- Aufbau Block:

```
[ DECLARE ....]
BEGIN
..... - - Anweisungen
[ EXCEPTION ..... ]
END;
```

- Blöcke können in Anweisungen und Ausnahmebehandlungen geschachtelt auftreten

PL/SQL

Deklarationsteil:

- Variablen mit SQL-Datentyp (ohne Größenangabe!, nur eine pro Zeile):
  ```
  artikelnr NUMBER;
  gebdat DATE;
  ```

- Variablen mit vordefinierten Subtypen:
  ```
  preis REAL;
  m INT :=2;
  ```

- erlaubte SQL- und Subtypen: char, long, long raw, varchar, varchar2, date, number,
  string, character, boolean, float, int, integer, natural, decimal, dec, real, smallint,...

- Variablen vom Typ eines ex. Attributs bzw. einer ganzen Tabelle:
  ```
  angest angestellter%ROWTYPE;
  angnummer angestellter.angnr%TYPE;
  ```

- Konstanten:
  ```
  limit CONSTANT REAL := 500.00;
  ```

- Cursor-Variablen: Art Tupelzeiger, notwendig um alle Tupel einer (Ergebnis-)Relation abzulaufen
  ```
  CURSOR cname IS sfw-anfrage;
  ```

PL/SQL

Anweisungen: Zuweisungen, IFs, Schleifen, Anfragen, Updates, Prozedur- und Funktionsaufrufe, Cursor-Funktionen

- Zuweisungen (s. obiges Beispiel)
- IF:

```
IF a<b THEN.... - - Anweisungen
END IF;
```

```
IF a=c THEN .....
ELSIF ......
ELSE .....
END IF;
```

PL/SQL

Forts. Anweisungen:

- Schleifen:
  - LOOP:

    ```
    LOOP
    .....
    END LOOP;
    ```

    in Verbindung mit `EXIT;` bzw. `EXIT WHEN bed.;` zum Herausspringen aus der Schleife

  - ```
    WHILE a!=b LOOP
    .....
    END LOOP;
    ```

  - ```
    FOR zaehler IN unten..oben LOOP
    ....
    END LOOP;
    ```

    `zaehler` wird automatisch inkrementiert

  - GOTO:

    ```
    ....
    GOTO label1;
    ....
    <<label1>>
    ...
    ```

PL/SQL

Forts. Anweisungen:

- Anfragen:

  ```
  SELECT attr1,... [ INTO var1 ]
  FROM relation
  WHERE bedingung [ FOR UPDATE OF attr1];
  ```

  `FOR UPDATE`-Angabe bewirkt Sperren der Tabelle, `INTO..`-Angabe bei nur einem Ergebnistupel, welches sofort einer Variablen zugeordnet wird (bei mehreren Ergebnistupeln: Cursor benutzen)

- Updates:

  ```
  INSERT INTO relation VALUES (...);
  UPDATE relation SET ..... WHERE bedingung;
  DELETE FROM relation WHERE bedingung;
  ```

- Prozedur- und Funktionsaufrufe:
  procname(aktparam1,..);
  var1 := fktname(aktparam1,..);

PL/SQL

Forts. Anweisungen:

- Cursor-Funktionen:

```
DECLARE
CURSOR c1 IS SELECT * FROM artikel;
art artikel%ROWTYPE;
BEGIN
...
OPEN c1; - - jetzt erst wird die Anfrage ausgeführt
...
FETCH c1 INTO art; - - holt (nächstes) Ergebnistupel der Anfrage,
z.B. in LOOP
...
CLOSE c1; - - schließt Cursor, Anfrageergebnis nicht mehr
vorhanden
```

  Erneutes `OPEN` nach `CLOSE` stellt Anfrage neu

PL/SQL

Forts. Anweisungen:

- Beispiel:

```
CREATE PROCEDURE auftr_numerieren IS
i NUMBER(6);
auftr auftrag%ROWTYPE;
CURSOR c IS SELECT * FROM auftrag FOR UPDATE; - - mit Locking
BEGIN
i := 1; OPEN c;
LOOP
FETCH c INTO auftr;
EXIT WHEN c%NOTFOUND;
UPDATE auftrag SET auftrnr = i WHERE CURRENT OF c;
i := i + 1;
END LOOP;
CLOSE c;
END;
```

- Cursor-Attribute: c%FOUND, c%NOTFOUND, c%ISOPEN, (jeweils TRUE oder FALSE), c%ROWCOUNT (Anzahl Tupel)

PL/SQL

Exceptions

- um bei Fehlerfällen eigene Routinen zur Ausnahmebehandlung aufzurufen
- Bsp: liefert `SELECT ...INTO ... FROM..WHERE` kein oder mehr als ein Tupel: Fehler `NO_DATA_FOUND` bzw. `TOO_MANY_ROWS` wird ausgelöst
- vordefinierte Behandlungsroutinen für Fehler: zumeist Fehlermeldung, Programmabbruch
- Definition eigener Ausnahmebehandlung für vordefinierte Fehler:

```
...
EXCEPTION
WHEN NO_DATA_FOUND THEN
...- - eigene Anweisungen
WHEN TOO_MANY_ROWS THEN
... - - eigene Anweisungen
WHEN OTHERS THEN
... - - eigene Anweisungen
```

- vordefinierte Fehler:

  NO_DATA_FOUND, TOO_MANY_ROWS, LOGIN_DENIED, NOT_LOGGED_ON, CURSOR_ALREADY_OPEN, INVALID_CURSOR, INVALID_NUMBER, DUP_VAL_ON_INDEX, ROWTYPE_MISMATCH, STORAGE_ERROR, TIMEOUT_ON_RESOURCE, VALUE_ERROR, PROGRAM_ERROR, ZERO_DIVIDE

PL/SQL

Forts. Exceptions

- 'eigene' Fehler: Fehler deklarieren, auslösen, behandeln
- deklarieren:

```
DECLARE
my_except EXCEPTION;
...
```

- auslösen:

```
...
IF (a=0) RAISE my_except;
...
```

- behandeln:

```
WHEN my_except THEN anweisungen;
```

PL/SQL

Aufbau Funktion:

- ```
  FUNCTION fktname [ (par1,...)] RETURN typ IS
  [lokale Deklarationen ]
  BEGIN
  ....- - Anweisungen, darunter mind. eine der Art:
  RETURN wert;
  [EXCEPTION......]
  END;
  ```

- Aufbau der Parameter par1,...:

  parametername [**IN** | **OUT** | **IN OUT**] **typ** [**:= expr** | **DEFAULT expr**]
  voreingestellt: `IN`, **DEFAULT**-Angabe nur bei `IN` erlaubt

- Beispiel:

  ```
  FUNCTION erhoehe_gehalt (prozent REAL DEFAULT 1.5)
  RETURN REAL - - hoechstes Gehalt soll zurueckgegeben werden..
  IS
  hoechstes REAL; -- lokale Deklaration
  BEGIN
  UPDATE angestellter
  SET gehalt = gehalt + (prozent*gehalt/100);
  COMMIT; - - Transaktion beenden, Änderung dauerhaft machen
  SELECT max(gehalt) INTO hoechstes
  FROM angestellter;
  RETURN hoechstes;
  END;
  ```
  ```

PL/SQL

Aufbau Prozedur:

- ```
  PROCEDURE pname [(par1,...)] IS
  [lokale Deklarationen ]
  BEGIN
  .... - -Anweisungen
  [EXCEPTION .....]
  END;
  ```

- in SQL*Plus möglich: Prozeduren mittels `CREATE PROCEDURE.....` definieren, mit / Definition abschließen;
  Prozedur mit `EXECUTE pname(aktpar1,....)` ausführen;

- Prozedur neu definieren: `CREATE OR REPLACE PROCEDURE pname....`

- Prozeduren, Funktion löschen mit `DROP PROCEDURE pname` bzw. `DROP FUNCTION fktname`
  ```