Oracle
DBA & Developer Days 2011

日本オラクル、今年最大の技術トレーニングイベント

2011年11月9日(水)～11月11日(金) シェラトン都ホテル東京

ORACLE

ここからはじめよう Oracle PL/SQL入門

日本オラクル株式会社 オラクルダイレクト テクニカルサービスグループ
マスタープリンシパルセールスコンサルタント 宇多津 真彦

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

Agenda

- PL/SQLとは
- プログラムを作成してみる
- 例外処理
- ストアド・プログラム
- パッケージ機能でライブラリをつくる
- トランザクション制御

PL/SQLとは

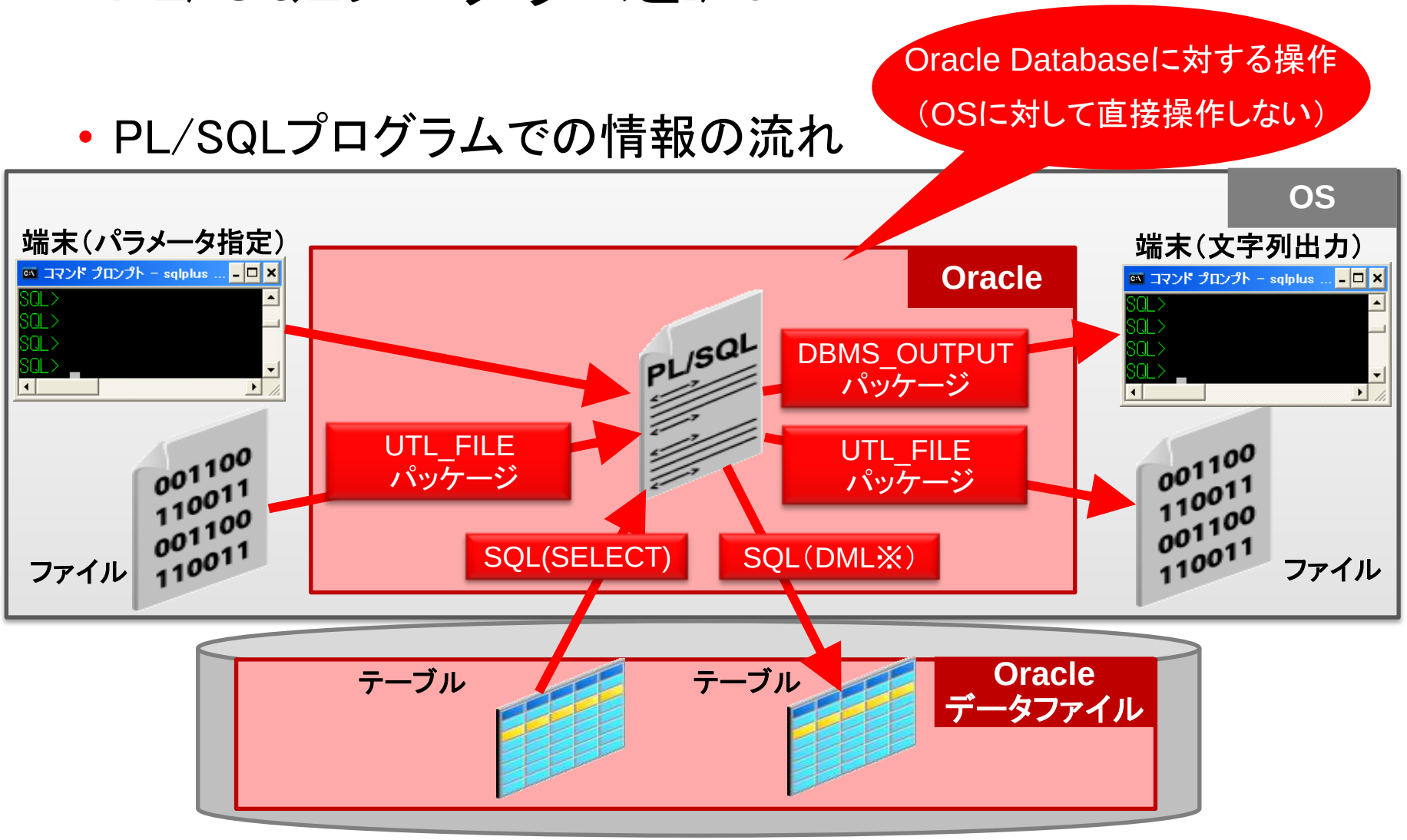
- PL/SQLは、SQLの手続き型拡張機能としてオラクル社が提供する言語です
- SQLはデータの集合を操作するための言語ですが、実際の業務処理(= ビジネス・ロジック)では手続き型の処理が必要な場合があります
- Oracle Databaseの内部でSQLと緊密な連携をおこなうことで効率的に業務処理を実行できます

PL/SQLを使うメリット

- SQL言語と緊密に統合されており、手続き型の処理を記述可能
  - SQLで取得した「データの塊（集合）」を、PL/SQLにより業務上の要望に応じて加工（集合演算とは異なる処理）
  - PL/SQLでストアド・ファンクションを作成することで、SQL操作の中に複雑な業務ロジックを組み込むことができる
  - 複数のトランザクションを統合し、一連の業務処理をまとめて記述可能（バッチ処理を記述）
- Oracle Databaseのデータ操作をコンパクトに記述できる
  - 例外処理、カプセル化、データ隠ぺいおよびオブジェクト指向のデータ等、プログラミングに必要充分な機能を持つ
- Oracle Database上で稼動するので、プラットフォームによるプログラミングの差異が無い（= 高い移植性）

PL/SQLプログラムとI/O

- PL/SQLプログラムでの情報の流れ



※ DML = INSERT / UPDATE / DELETE 等

Agenda

- PL/SQLとは
- プログラムを作成してみる
- 例外処理
- ストアド・プログラム
- パッケージ機能でライブラリをつくる
- トランザクション制御

HELLO WORLD プログラム

- SQL*Plus上に文字列を出力する

HELLO WORLD プログラム

- SQL*Plus上に文字列を出力する

```
DECLARE
  str VARCHAR2(30);
BEGIN
  str := 'HELLO WORLD';

  DBMS_OUTPUT.PUT_LINE(str);
END;
```

ファイル: put_line.sql

```
% sqlplus scott/tiger
SQL> @put_line
  8  /
PL/SQLプロシージャが正常に完了しました。
SQL> set serveroutput on
SQL> @put_line
  8  /
HELLO WORLD

PL/SQLプロシージャが正常に完了しました。

SQL>
```

ファイル内容をバッファに読み込む

バッファの内容を実行する

期待通りに表示されない

SQL*Plusの設定

SQL文を実行する
（DML：INSERT/UPDATE/DELETE など）

- そのままDML文を記述する

```
DECLARE
BEGIN
  INSERT INTO emp (
    empno,
    ename
  )VALUES(
    1000,
    'Ichiro'
  );
END;
```

ファイル：insert_emp.sql

```
% sqlplus scott/tiger
SQL> @insert_emp
 12  /

PL/SQLプロシージャが正常に完了しました。

SQL> SELECT empno,ename FROM emp
  2   WHERE empno = 1000;

     EMPNO ENAME
---------- ----------
      1000 Ichiro
SQL>
```

SQL文を実行する
SELECT：基本構文とよくある失敗（その1）

- テーブル emp の件数を取得したい

```
DECLARE
BEGIN
   SELECT count(*)
     FROM emp;
END;
```

ファイル：select_cnt_01.sql

```
% sqlplus scott/tiger
SQL> @select_cnt_01
  6  /
  SELECT count(*)
  *
行3でエラーが発生しました。:
ORA-06550: 行3、列3:
PLS-00428: INTO句はこのSELECT文に入ります。
```

- PL/SQLでは、SELECT文はSQLの内部で利用するか、INTO句を利用して一時的に変数に格納します

```
DECLARE
   cnt NUMBER(4);
BEGIN
   SELECT count(*) INTO cnt
     FROM emp;
   DBMS_OUTPUT.PUT_LINE(cnt);
END;
```

ファイル：select_cnt_02.sql

```
% sqlplus scott/tiger
SQL> set serveroutput on
SQL> @select_cnt_02
  8  /
12        ← 件数:12件
PL/SQLプロシージャが正常に完了しました。
SQL>
```

SQL文を実行する
SELECT：基本構文とよくある失敗（その2）

- テーブル emp（複数行存在）のカラム内容を取得したい

```
DECLARE
  str VARCHAR2(10);
BEGIN
  SELECT ename INTO str FROM emp;
  DBMS_OUTPUT.PUT_LINE(str);
END;
```

ファイル：select_ename01.sql

```
% sqlplus scott/tiger
SQL> @select_col_a01
  7  /
DECLARE
*
行1でエラーが発生しました。:
ORA-01422: 完全フェッチがリクエストよりも多くの行を戻しました
ORA-06512: 行4
```

```
DECLARE
  CURSOR c1 IS
    SELECT ename FROM emp;
  str VARCHAR2(10);        カーソル定義
BEGIN
                  カーソルを OPEN
  OPEN c1;
  LOOP
    FETCH c1 INTO str;  一件取り出し
    EXIT WHEN C1%NOTFOUND;
      ↑データが無くなればループを脱出
    DBMS_OUTPUT.PUT_LINE(str);
  END LOOP;
  CLOSE c1;  カーソルを CLOSE
END;
```

ファイル：select_ename02.sql

レコード単位での処理

- テーブルのデータを行毎に処理したい場合、レコードを利用

```
DECLARE
  CURSOR c1 IS
    SELECT empno,ename FROM emp;
  TYPE emp_rec_type IS RECORD (
    empno NUMBER(4),
    ename VARCHAR2(10)
  );
  emp_rec emp_rec_type;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO emp_rec;
    EXIT WHEN C1%NOTFOUND;

    DBMS_OUTPUT.PUT(TO_CHAR(emp_rec.empno));
    DBMS_OUTPUT.PUT_LINE(' '||emp_rec.ename);
  END LOOP;
  CLOSE c1;
END;
```

レコードの定義

個々のメンバーを「フィールド」とよぶ

←定義したレコード型を元にレコード変数を定義

<レコード変数>.<フィールド名>でアクセス可能

効果的な変数定義（%TYPE）

- %TYPEをつかう
  - テーブル定義が変更されてもソースは修正しなくともよい

```
DECLARE
  CURSOR c1 IS
    SELECT empno,ename FROM emp;
  TYPE emp_rec_type IS RECORD (
    empno NUMBER(4),
    ename VARCHAR2(10)
  );
  emp_rec emp_rec_type;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO emp_rec;
    EXIT WHEN C1%NOTFOUND;

    DBMS_OUTPUT.PUT(TO_CHAR(emp_rec.empno));
    DBMS_OUTPUT.PUT_LINE(' '||emp_rec.ename);
  END LOOP;
  CLOSE c1;
END;
```

```
DECLARE
  CURSOR c1 IS
    SELECT empno,ename FROM emp;
  TYPE emp_rec_type IS RECORD (
    empno emp.empno%TYPE,
    ename emp.ename%TYPE
  );
  emp_rec emp_rec_type;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO emp_rec;
    EXIT WHEN C1%NOTFOUND;

    DBMS_OUTPUT.PUT(TO_CHAR(emp_rec.empno));
    DBMS_OUTPUT.PUT_LINE(' '||emp_rec.ename);
  END LOOP;
  CLOSE c1;
END;
```

emp テーブルの empno カラムと同じ型定義

効果的な変数定義（%ROWTYPE）

- %ROWTYPEをつかうとレコード変数定義が簡単になります

```
DECLARE
  CURSOR c1 IS
    SELECT empno,ename FROM emp;
  TYPE emp_rec_type IS RECORD (
    empno emp.empno%TYPE,
    ename emp.ename%TYPE
  );
  emp_rec emp_rec_type;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO emp_rec;
    EXIT WHEN C1%NOTFOUND;

    DBMS_OUTPUT.PUT(TO_CHAR(emp_rec.empno));
    DBMS_OUTPUT.PUT_LINE(' '||emp_rec.ename);
  END LOOP;
  CLOSE c1;
END;
```

```
CURSOR c1 IS
  SELECT empno,ename FROM emp;
emp_rec c1%ROWTYPE;
```

カーソル定義をもとに
レコード変数を定義

```
CURSOR c1 IS
  SELECT empno,ename FROM emp;
emp_rec emp%ROWTYPE;
```

テーブル定義をもとに
レコード変数を定義

参考）レコードを利用した挿入と更新

- 列リストを指定するかわりにレコードを指定できます

```
DECLARE
  emp_rec emp%ROWTYPE;
BEGIN
  emp_rec.empno := 1000;
  emp_rec.ename := 'Ichiro';
  INSERT INTO emp (
    empno, ename
  )VALUES(
    emp_rec.empno, emp_rec.ename
  );
END;
```

通常のINSERT構文

```
DECLARE
  emp_rec emp%ROWTYPE;
BEGIN
  emp_rec.empno := 1000;
  emp_rec.ename := 'Ichiro';
  INSERT INTO emp VALUES emp_rec;
END;
```

INSERTの拡張構文

```
DECLARE
  emp_rec emp%ROWTYPE;
BEGIN
  emp_rec.empno := 1000;
  emp_rec.ename := 'Ichiro01';
  emp_rec.deptno := 10;
UPDATE emp
   SET ename = emp_rec.ename,
       deptno = emp_rec.deptno
 WHERE empno = emp_rec.empno;
END;
```

通常のUPDATE構文

```
DECLARE
  emp_rec emp%ROWTYPE;
BEGIN
  emp_rec.empno := 1000;
  emp_rec.ename := 'Ichiro01';
  emp_rec.deptno := 10;
 UPDATE emp
    SET ROW = emp_rec
  WHERE empno = emp_rec.empno;
END;
```

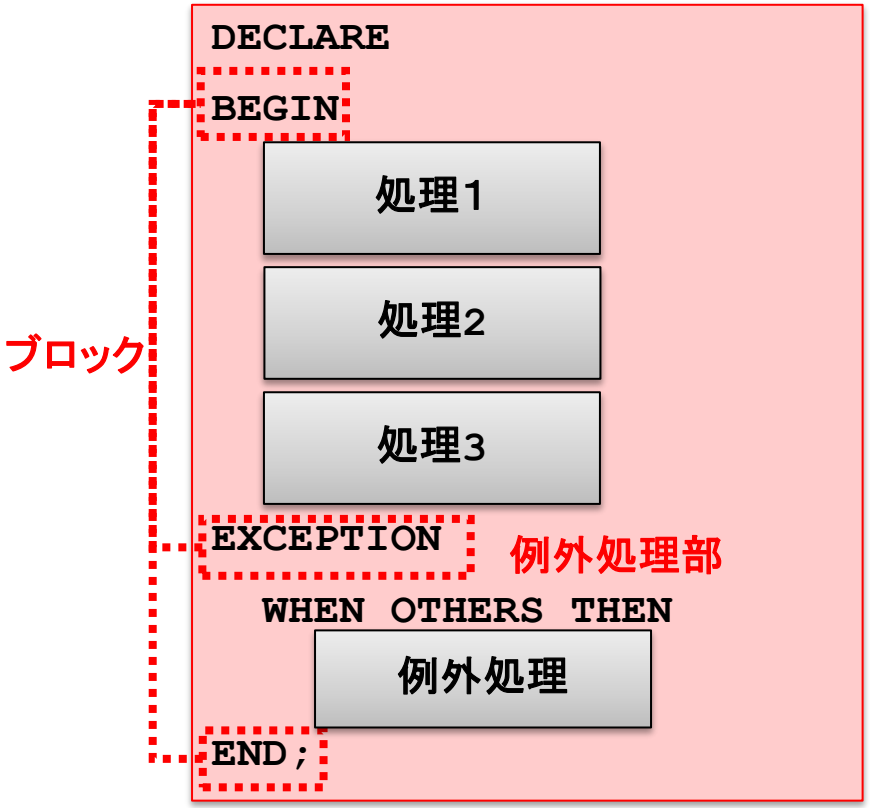
UPDATEの拡張構文

Agenda

- PL/SQLとは
- プログラムを作成してみる
- 例外処理
- ストアド・プログラム
- パッケージ機能でライブラリをつくる
- トランザクション制御

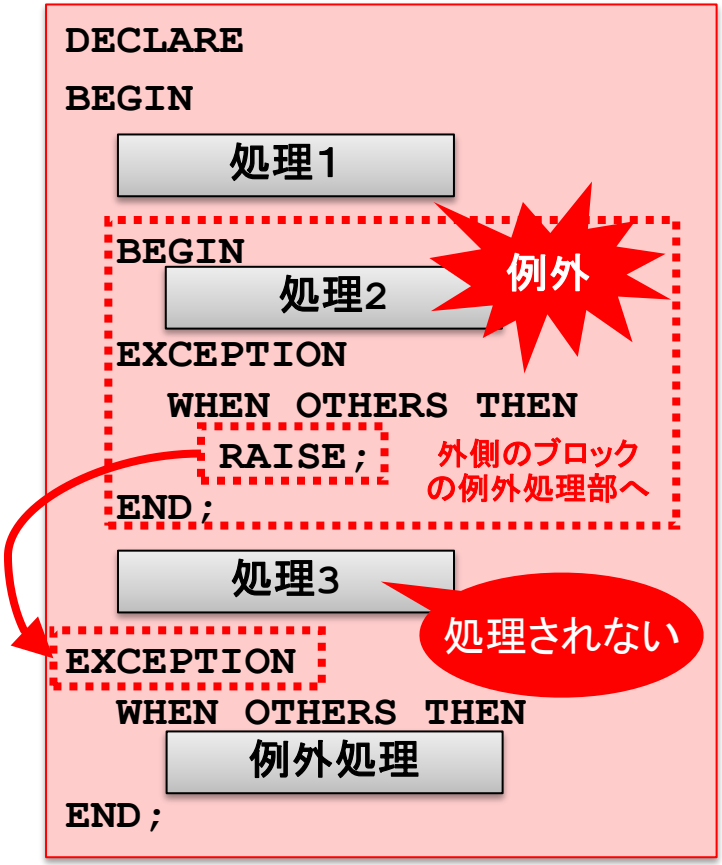
例外処理とは

- 例外（PL/SQL実行時エラー）が発生してもプログラムで処理を継続できるようにすることができます



ブロックの入れ子と例外処理

- 入れ子のブロックできめ細かな例外処理が可能です



特定の例外を捕捉する

- 特定の例外を捕捉し、例外に応じた処理をおこないます

```
DECLARE
BEGIN
  INSERT INTO emp (empno,ename)
   VALUES (1000,'Ichiro');
  BEGIN
    INSERT INTO emp (empno,ename)
     VALUES (1000,'Ichiro 2');
  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      UPDATE emp
         SET ename = 'Ichiro 2'
       WHERE empno = 1000;
    WHEN OTHERS THEN
      NULL;
  END;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error');
END;
```

キー重複エラーを捕捉

ファイル：ins_upd_emp01.sql

```
SQL> @ins_upd_emp
 20  /

PL/SQLプロシージャが正常に完了しました。

SQL> SELECT empno,ename FROM emp
  2   WHERE empno = 1000;

     EMPNO ENAME
---------- ----------
      1000 Ichiro 2

SQL>
```

よく利用される例外は事前定義例外として提供されています

捕捉した例外情報を表示

- SQLERRM、SQLCODEを利用してエラー情報を取得します

```
DECLARE
  -- emp表のempnoカラムはNUMBER(4)
BEGIN

  INSERT INTO emp (
    empno,
    ename
  ) VALUES (
    10000,
    'Ichiro*10'
  );

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE( SQLERRM );
    DBMS_OUTPUT.PUT_LINE( SQLCODE );
END;
```

ファイル：insert_emp_err01.sql

```
SQL> @insert_emp_err01

 18   /
ORA-01438: この列に許容される指定精度より
大きな値です
-1438

PL/SQLプロシージャが正常に完了しました。


SQL>
```

ユーザ独自の例外定義

- 例外を定義して、例外処理部で独自にハンドリングできます
  - RAISE_APPLICATION_ERRORを利用して例外を発生させます

```
DECLARE
  usr_excp EXCEPTION;
  usr_excp_num NUMBER := -20001;
  usr_excp_str VARCHAR2(30) := 'User Exception';
  PRAGMA EXCEPTION_INIT(usr_excp, -20001);
BEGIN

  RAISE_APPLICATION_ERROR(
    usr_excp_num,      ←エラー・コードを指定
    usr_excp_str
  );

EXCEPTION
  WHEN usr_excp THEN
    DBMS_OUTPUT.PUT_LINE( SQLERRM );
    DBMS_OUTPUT.PUT_LINE( SQLCODE );
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error');
END;
```

ファイル：raise_appl_err01.sql

```
SQL> @raise_appl_err01
 20  /

ORA-20001: User Exception
-20001
```

メッセージ取得可能

```
PL/SQLプロシージャが正常に完了しました。

SQL>
```

ユーザが利用できるエラー番号の範囲：
-20000から-20999 まで

エラー・メッセージは2048Byte以内

参考）ユーザ定義例外のRAISE

- RAISEではエラーメッセージを取得できません

```
DECLARE
  usr_excp EXCEPTION;
  usr_excp_num NUMBER := -20001;
  usr_excp_str VARCHAR2(30) := 'User Exception';
  PRAGMA EXCEPTION_INIT(usr_excp, -20001);
BEGIN

  RAISE usr_excp;

EXCEPTION
  WHEN usr_excp THEN
    DBMS_OUTPUT.PUT_LINE( SQLERRM );
    DBMS_OUTPUT.PUT_LINE( SQLCODE );
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error');
END;
```

ファイル：raise_appl_err02.sql

```
SQL> @raise_appl_err02
 17  /
ORA-20001: メッセージが取得できない
-20001

PL/SQLプロシージャが正常に完了しました。

SQL>
```

RAISEにユーザ定義例外を指定するのみ

エラーメッセージが取得できない為、
エラーハンドリングを検討する必要が
あります

Agenda

- PL/SQLとは
- プログラムを作成してみる
- 例外処理
- ストアド・プログラム
- パッケージ機能でライブラリをつくる
- トランザクション制御

ストアド・プログラム

- ストアド・プロシージャ
  - PL/SQLで記述した一連の処理手続きに名前をつけ、Oracle Databaseに格納したもの

```
DECLARE
BEGIN
…
END;
```

```
CREATE PROCEDURE proc IS
BEGIN
…
END;
```

プロシージャ作成

- ストアド・ファンクション
  - PL/SQLで記述した、なんらかの値を返却する処理手続きに名前をつけ、Oracle Databaseに格納したもの

```
DECLARE
BEGIN
  IF cnt > 20 THEN
    …
  END IF;
END;
```

```
DECLARE
BEGIN
  IF func(cnt,20) THEN

  END IF;
END;
```

```
CREATE FUNCTION func
 (p1 NUMBER, p2 NUMBER)
 RETURN BOOLEAN IS
BEGIN
 IF p1 > p2 THEN
   RETURN TRUE;
 END IF;
 RETURN FALSE;
END;
```

ファンクション作成

プロシージャとファンクションの使い分け

- プロシージャ
  - バッチ処理などの一連の処理手続きはプロシージャにて実装
    - プロシージャ内部でINSERT/UPDATE/DELETEを実施
  - エラーを例外にて捕捉でき、必要に応じて処理を取り消す（ROLLBACK）ことができる
- ファンクション
  - パラメータにて渡された値の判定、変換処理、値の抽出をおこなう
    - 一般的にファンクション内部ではINSERT/UPDATE/DELETEを実施しない
  - エラーは例外で捕捉するものの例外処理は行なわず、エラーが発生したことを示す値を返却することが多い
  - SQL文中で利用することが可能（一部例外あり）

ファンクションをつくる

- 値の抽出、値の変換などをおこなう例

```
CREATE OR REPLACE FUNCTION
  get_sales_tax(date_in IN DATE)
  RETURN NUMBER
IS
  tax sales_tax.tax%TYPE;
BEGIN
  SELECT tax INTO tax
    FROM sales_tax
   WHERE FROM_DATE <= date_in
     AND TO_DATE   >= date_in;
  RETURN tax;
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
```

例外発生時はNULLを戻す

テーブルより特定情報を取得

```
CREATE OR REPLACE FUNCTION
  get_quarter(date_in IN DATE)
  RETURN NUMBER
IS
  mm_str VARCHAR2(2);
BEGIN
  mm_str := TO_CHAR(date_in,'MM');
  CASE mm_str
   WHEN '01' THEN RETURN 3; WHEN '02' THEN RETURN 3;
   WHEN '03' THEN RETURN 4; WHEN '04' THEN RETURN 4;
   WHEN '05' THEN RETURN 4; WHEN '06' THEN RETURN 1;
   WHEN '07' THEN RETURN 1; WHEN '08' THEN RETURN 1;
   WHEN '09' THEN RETURN 2; WHEN '10' THEN RETURN 2;
   WHEN '11' THEN RETURN 2; WHEN '12' THEN RETURN 3;
   ELSE           RETURN NULL;
  END CASE;
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
```

例外発生時はNULLを戻す

値の変換

参考）内部でSELECTをおこなうファンクション

- ファンクション内部でSELECTを実施しており、データ量によってはパフォーマンスが悪くなることがあります

```
SELECT … FROM  tbl1
 WHERE  col1 > get_price(item_id, sysdate-60) ;
```

```
CREATE FUNCTION get_price (item_id_in IN NUMBER, dt_in IN DATE)
  RETURN NUMBER
IS
  ret_num NUMBER;
BEGIN
  SELECT price  INTO ret_num FROM price_list
   WHERE item_id = item_id_in
     AND from_dt <= TRUNC(dt_in) AND to_dt  >   TRUNC(dt_in) ;
    RETURN ret_num;
END get_price;
```

対処案1）ファンクションの利用をやめ、結合処理に作り直す

対処案2）PL/SQLファンクションの結果キャッシュを利用する

参考）内部でSELECTをおこなうファンクション

PL/SQLファンクションの結果キャッシュ（11g R1～、Enterprise Edition）

- PL/SQLファンクションの結果をSGAにキャッシュし、複数のセッションで利用できます
  - ファンクションおよびパラメータの値を組にして結果をキャッシュ
  - システムで必要なメモリが足りなくなると古いものから破棄
  - ファンクション内で参照している表が変更されるとキャッシュは破棄

```
CREATE OR REPLACE FUNCTION get_price(item_id_in IN NUMBER, dt_in IN DATE)
RETURN NUMBER  RESULT_CACHE   RELIES_ON ( price_list )
IS
  ret_num NUMBER;
BEGIN
  SELECT price INTO ret_num FROM price_list
   WHERE item_id =  item_id_in
     AND from_dt <= TRUNC(dt_in)
     AND to_dt   >  TRUNC(dt_in) ;
  RETURN ret_num;
END get_price;
```

11g R2**からは**
RELIES_ON
記述は不要

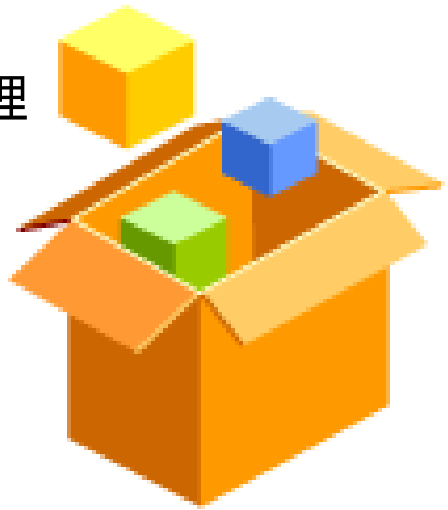
Agenda

- PL/SQLとは
- プログラムを作成してみる
- 例外処理
- ストアド・プログラム
- パッケージ機能でライブラリをつくる
- トランザクション制御

パッケージ機能でライブラリをつくる

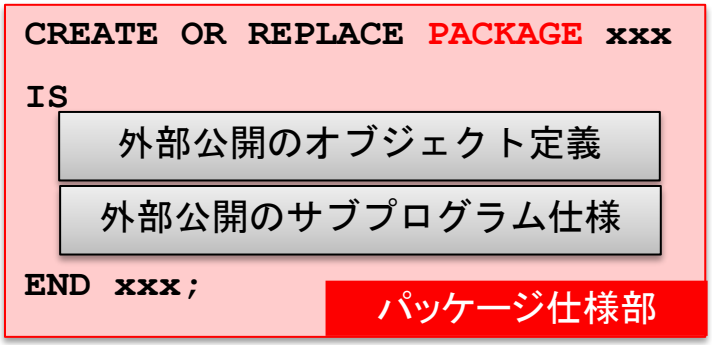
- パッケージとは
  - 論理的に関連するPL/SQLの型、変数およびサブプログラムをまとめる為のスキーマ・オブジェクト

- パッケージの利点
  - モジュール性
    - アプリケーション共通で利用するものを一元管理
    - 情報の隠蔽
  - サブプログラムのオーバーロード機能
  - オブジェクトの永続性

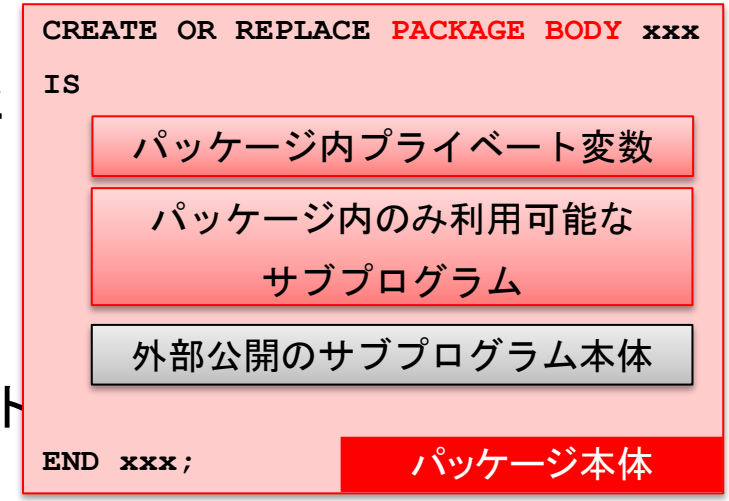


パッケージのモジュール性

- アプリケーション共通で利用するものを一元管理できます
  - パッケージ仕様部に記述します
    - 定数（アプリ利用のマジックナンバー）
    - 例外、カーソル、型（レコード）



- 他のアプリケーションから実行されたくないサブプログラムなどを隠蔽できます
  - パッケージ本体のみに記述します
  - パッケージ内のみ利用可能なプライベート変数も定義できます



パッケージをつくる

```
CREATE OR REPLACE PACKAGE pkg IS
  PROCEDURE p    (p_in IN VARCHAR2);
  FUNCTION  f_n (p_in IN NUMBER)
    RETURN NUMBER;
  FUNCTION  f_c (p_in IN VARCHAR2)
    RETURN VARCHAR2;
END;
```

ファイル（パッケージ仕様部）: pkg01.pks

```
CREATE OR REPLACE PACKAGE BODY pkg IS
  PROCEDURE p(p_in IN VARCHAR2)IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_in);
  END p;
  FUNCTION f_n (p_in IN NUMBER)
    RETURN NUMBER IS
  BEGIN
    RETURN p_in;
  END f_n;
  FUNCTION f_c (p_in IN VARCHAR2)
    RETURN VARCHAR2 IS
  BEGIN
    RETURN p_in;
  END f_c;
END;
```

ファイル（パッケージ本体）: pkg01.pkb

```
SQL> @pkg01.pks
  8  /

パッケージが作成されました。

SQL> @pkg01.pkb
 17  /

パッケージ本体が作成されました。

SQL> execute pkg.p('test')
test

PL/SQLプロシージャが正常に完了しました。

SQL> select pkg.f_n(100) from dual;

PKG.F_N(100)
------------
         100

SQL> select pkg.f_c('test') from dual;

PKG.F_C('TEST')
--------------------------------
test
```

プロシージャをSQL*Plusで実行する際はexecute にて指定

オーバーロード

同じ名前（引数の数、順序、型は異なる）のサブプログラムを定義

```
CREATE OR REPLACE PACKAGE pkg IS
  PROCEDURE p (p_in IN VARCHAR2);
  FUNCTION  p (p_in IN NUMBER)
    RETURN NUMBER;
  FUNCTION  p (p_in IN VARCHAR2)
    RETURN VARCHAR2;
END;
```

ファイル（パッケージ仕様部）：pkg01.pks

```
CREATE OR REPLACE PACKAGE BODY pkg IS
  PROCEDURE p(p_in IN VARCHAR2)IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_in);
  END p;
  FUNCTION p(p_in IN NUMBER)
    RETURN NUMBER IS
  BEGIN
    RETURN p_in;
  END p;
  FUNCTION p(p_in IN VARCHAR2)
    RETURN VARCHAR2 IS
  BEGIN
    RETURN p_in;
  END p;
END;
```

ファイル（パッケージ本体）：pkg01.pkb

```
SQL> @pkg02.pks
  8  /

パッケージが作成されました。

SQL> @pkg02.pkb
 17  /

パッケージ本体が作成されました。

SQL> execute pkg.p('test')
test

PL/SQLプロシージャが正常に完了しました。

SQL> select pkg.p(100) from dual;

PKG.P(100)
------------
         100

SQL> select pkg.p('test') from dual;

PKG.P('TEST')
--------------------------------
test
```

適切に選択される

パッケージオブジェクトの永続性

- パッケージで定義されている変数の変更は、セッションが切れるまで有効です
  - セッションが同じであれば、
  - 複数のトランザクション（後述）間で「グローバル変数」のように利用可能

例）

- 同一セッションで「アプリ基準日付」を使いまわしたい
  - テーブルへのアクセスは、セッション開始後の初回実行時のみ

```
CREATE OR REPLACE PACKAGE BODY base_date
IS
  /* プライベート変数 */
  p_base_date DATE := NULL;
  FUNCTION get_base_date
    RETURN DATE
  IS
  BEGIN
    IF p_base_date IS NOT NULL THEN
      dbms_output.put_line('no select');
      RETURN p_base_date;
    END IF;
    SELECT app_date INTO p_base_date
      FROM CONTROL_MST WHERE id = 2;
    RETURN p_base_date;
  EXCEPTION
    WHEN OTHERS THEN
      RETURN NULL;
  END get_base_date;
END base_date;
```

アプリ基準日付の取得

参考）パッケージの初期化部

- パッケージ本体に初期化部を記述することもできます

```
CREATE OR REPLACE PACKAGE BODY base_date
IS
  /* プライベート変数 */
  p_base_date DATE := NULL;
  FUNCTION get_base_date
    RETURN DATE
  IS
  BEGIN
    IF p_base_date IS NOT NULL THEN
      dbms_output.put_line('no select');
      RETURN p_base_date;
    END IF;
    SELECT app_date INTO p_base_date
      FROM CONTROL_MST WHERE id = 2;
    RETURN p_base_date;
  EXCEPTION
    WHEN OTHERS THEN
      RETURN NULL;
  END get_base_date;
END base_date;
```

```
CREATE OR REPLACE PACKAGE BODY base_date
IS
  /* プライベート変数 */
  p_base_date DATE := NULL;
  FUNCTION get_base_date
    RETURN DATE
  IS
  BEGIN
    RETURN p_base_date;
  END get_base_date;

BEGIN
  SELECT app_date INTO p_base_date
    FROM CONTROL_MST WHERE id = 2;
EXCEPTION
  WHEN OTHERS THEN
    p_base_date := NULL;
END base_date;
```

初期化部

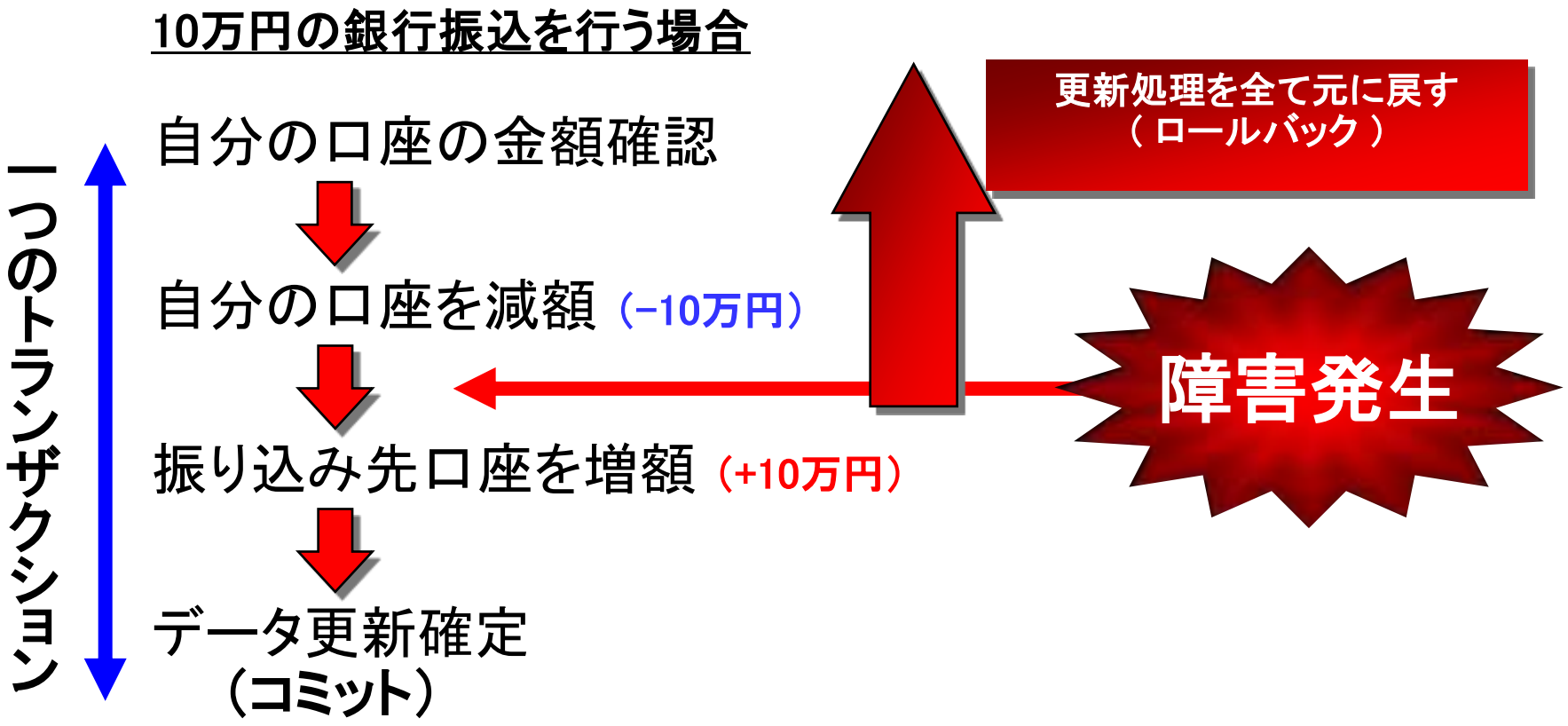
パッケージのインスタンス化時に1回だけ実行

Aganda

- PL/SQLとは
- プログラムを作成してみる
- 例外処理
- ストアド・プログラム
- パッケージ機能でライブラリをつくる
- トランザクション制御

トランザクション管理

- トランザクション ＝ 分割できない一連の情報処理の単位



トランザクションの制御

- Oracle DatabaseではDML文やDDL文が最初に実行された時からトランザクションが開始される（宣言は不要）

```
BEGIN
    処理1
    処理2
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
END;
```

- 通常、
  処理部の最後にCOMMIT
  例外処理部分にROLLBACK

- トランザクションとして制御したい単位でストアド・プロシージャ（後述）に分割します

トランザクションの制御（バッチ的な処理）

- エラーが発生した時に、再実行可能な単位でトランザクションを制御（コミット/ロールバック）する

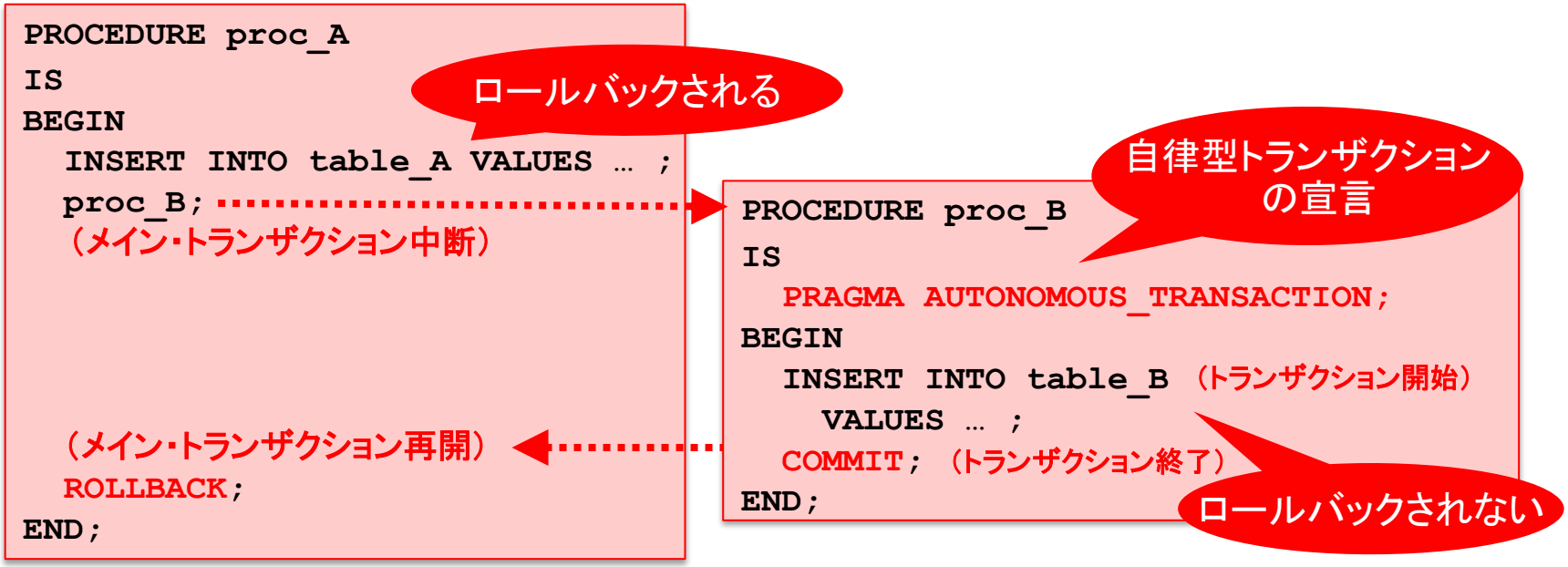
再実行可能な単位をどこに設定するのか？

- すべての処理が正常終了するまでコミットしない
  - エラーが発生するとすべてをロールバックし、エラー原因を対処し、再実行
- 処理単位毎にコミットする
  - 処理がどこまで正常に終了しているかを記録しておく必要がある
  - 再実行時には正常終了部分の処理はスキップする処理を組み込む

PL/SQLで作成するジョブの単位を適切に検討する

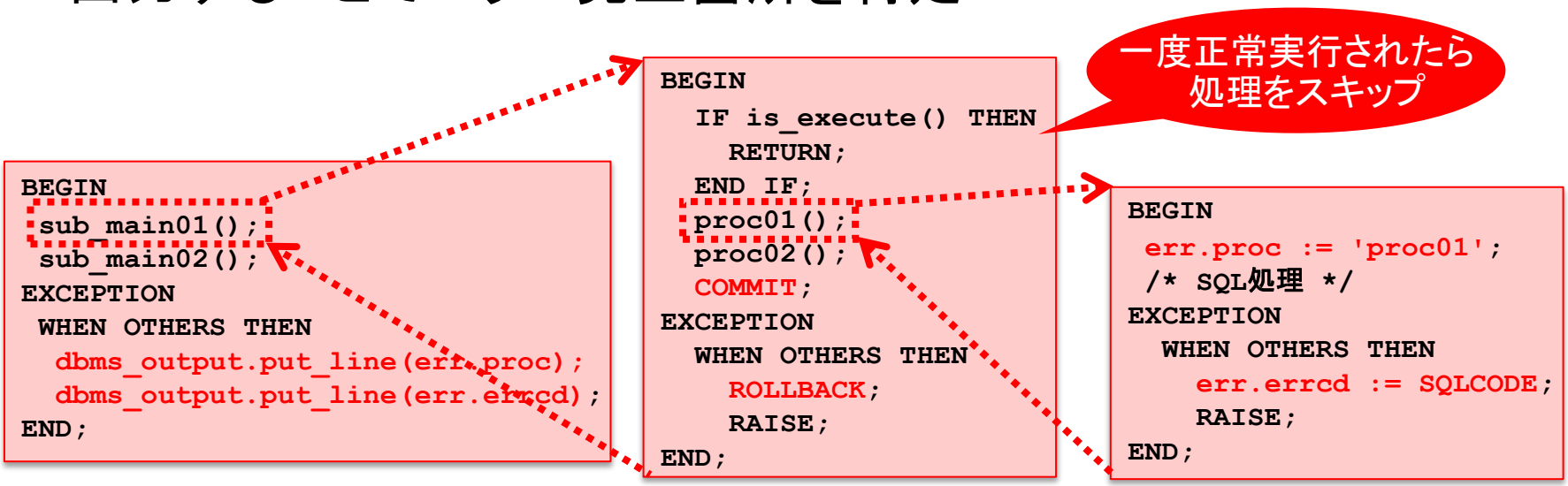
自律型トランザクション

- 実行中のトランザクションとは別の独立したトランザクションのサブプログラムを呼び出すことができます
  - メイントランザクションがロールバックしてもログなどを取得できます
  - 自律型トランザクション側でコミットした情報を別のトランザクションからすぐに確認できます



参考）バッチプログラムを作成するときのTips

- 複数のSQLを単一のブロックにいれない
  - 必ず例外処理部でエラー処理をおこなう
- ブロック毎に名前をつけて、パッケージ変数に名前をセット
- エラーが発生するとパッケージ変数にエラー情報をセット
- 最上位ブロックの例外処理部では上記のパッケージ変数を出力することでエラー発生箇所を特定



まとめ

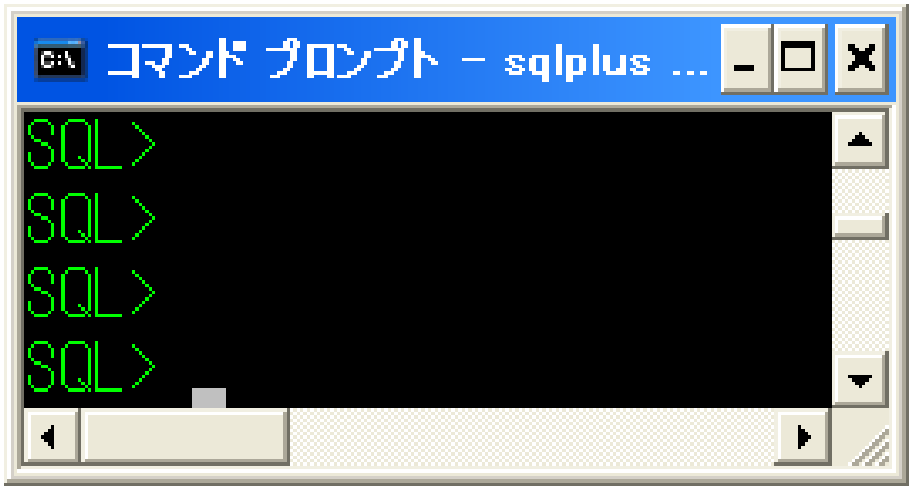
- PL/SQLにより Oracle Database の内部で効率的に業務処理を実行できます
- ストアド・プログラムを使い業務ロジックを作成することで、Oracle Database 利用者は業務ロジックを共用することができます
- 例外処理、トランザクション処理、パッケージ機能の特性など、PL/SQLを利用する上で知っておくべきことを解説しました

Appendix

- よくつかうSQL*Plusの機能
  - SQL*Plusとは
  - SQLスクリプトの実行
  - SQLスクリプト内の文字列置換
  - SQL*Plusの表示機能
  - 表示内容のファイルへの保存
  - SQL*Plusバインド変数の使用と値の表示
  - SQLスクリプト終了時にOSにエラーを返却

SQL*Plusとは

- SQL*Plusは、Oracle Databaseへのコマンドライン・ユーザ・インターフェースであり、以下のようなことができます
  - Oracle Databaseの管理
  - 表定義およびオブジェクト定義の検証
  - Oracle Databaseへのバッチ処理スクリプトの開発および実行
  - 問い合わせ結果の書式設定、計算の実行



SQLスクリプトの実行

- 定型処理は、SQL*Plusで実行可能なSQLやPL/SQLをテキストファイルにひとつにまとめたものをSQLスクリプトとして実行します。
  - SQL*Plusコマンドラインにて指定

```
% sqlplus username/[password] @<ファイル名[.拡張子]> <パラメータ> …
```

  - SQL*PlusでOracle Databaseに接続した後に指定

```
% sqlplus username/[password]
SQL> start <ファイル名[.拡張子]> <パラメータ> …
SQL> @<ファイル名[.拡張子]> <パラメータ> …
```

※ パラメータを利用する際には、SQLスクリプトファイル中にて &1 および &2 のような置換変数（後述）を利用します

SQLスクリプト内の文字列置換

- 置換変数：任意の変数の前に'&'もしくは'&&'をつけたもの

```
select '&pswd01' from dual;

select '&&pswd02' from dual;
```

ファイル：replace01.sql

- 単一&置換変数はスクリプト中で出現する度に入力を促す
- 二重&置換変数は、スクリプト中の初回は入力を促すが、後続の変数では初回と同じ値を利用する（何回も問い合わせてこない）

```
SQL> @replace01
pswd01に値を入力してください: test1
旧    1: select '&pswd01' from dual
新    1: select 'test1' from dual

'TEST
-----
test1

pswd02に値を入力してください: test2
旧    1: select '&&pswd02' from dual
新    1: select 'test2' from dual

'TEST
-----
test2
```

1回目実行

```
SQL> @replace01
pswd01に値を入力してください: test-001
旧    1: select '&pswd01' from dual
新    1: select 'test-001' from dual

'TEST-00
--------
test-001

旧    1: select '&&pswd02' from dual
新    1: select 'test2' from dual

'TEST
-----
test2
```

入力を促さない

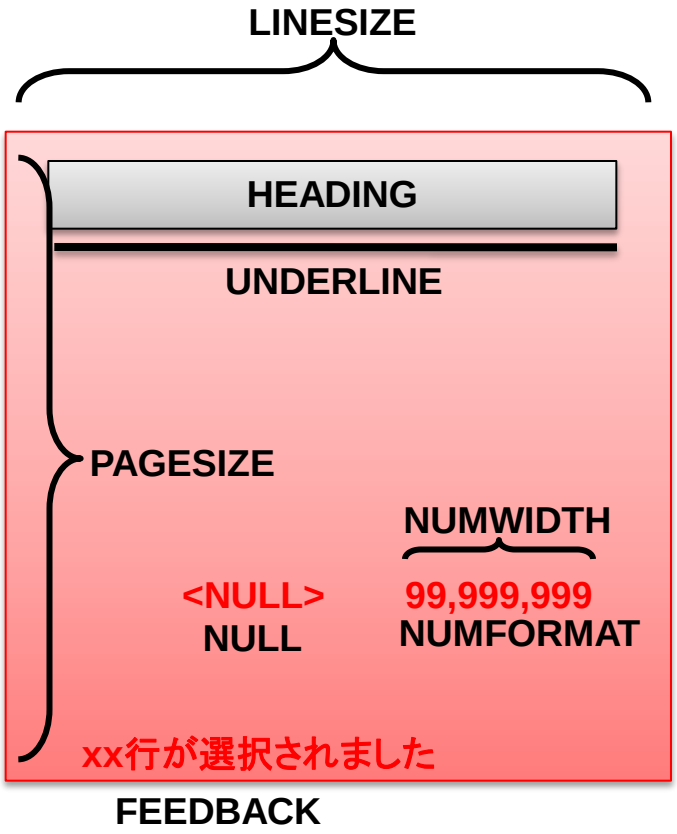
（1回目に続けて）

2回目実行

SQL*Plusの表示機能（その1）

- SQL*Plusの代表的な表示設定

ページ設定	SET PAGES[IZE] {14 | n}
1行の長さ	SET LIN[ESIZE] {80 | n}
レポートの列ヘッダ情報の出力有無	SET HEA[DING] {ON | OFF}
レポートの列ヘッダーのアンダーラインで利用する文字	SET UND[ERLINE] {- | c | ON | OFF}
数値のフォーマット	SET NUMF[ORMAT] format
数値のデフォルト桁	SET NUM[WIDTH] {10 | n}
NULL値を表す文字列	SET NULL text
画面出力の制御	SET TERM[OUT] {ON | OFF}
端末出力の行末スペース削除	SET TRIM[OUT] {ON | OFF}
ファイル出力時の行末スペース削除	SET TRIMS[POOL] {ON | OFF}
実行結果の表示	SET FEED[BACK] {6 | n | ON | OFF}
列セパレータの指定	SET COLSEP { | text}



SQL*Plusの表示機能（その2）

- 列の書式設定
  - COLUMNコマンドで設定
  - 列の書式設定
    - 数値型　column <カラム名> format <書式>　SQL> column sal format 99,999,999
    - 文字列型　column <カラム名> format a<長さ>　SQL> column ename format a10
    - 日付型：　必要に応じてSQL文上でTO_CHAR等を利用して文字列型に

```
SQL> select * from emp where empno = 7900;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- -------- ---------- ----------
    DEPTNO
----------
      7900 JAMES      CLERK            7698 81-12-03        950
        30

SQL> column empno format 9999
SQL> column mgr   format 9999
SQL> column sal   format 000,009
SQL> select * from emp where empno = 7900;
EMPNO ENAME      JOB          MGR HIREDATE     SAL       COMM     DEPTNO
----- ---------- --------- ----- -------- -------- ---------- ----------
 7900 JAMES      CLERK      7698 81-12-03 000,950                     30
```

読みにくい

数値型のデフォルトの桁長は10桁

表示内容のファイルへの保存

- SPOOLコマンドを利用して実行結果を保存する

```
SQL> spool c:¥temp¥emp.txt
SQL> select ename from emp
  2   where empno = 7900;

ENAME
----------
JAMES

SQL> spool off
SQL> exit
D:¥>
D:¥> type c:¥temp¥emp.txt
SQL> select ename from emp
  2   where empno = 7900;

ENAME
----------
JAMES

SQL> spool off
D:¥>
```

- 作成されるファイルはLINESIZEまでスペース詰されているので、不要時は SET TRIMSPOOL ON を設定

```
D:¥> type emp7900.sql
select ename from emp
 where empno = 7900;

D:¥> sqlplus scott/tiger
  (略)
SQL> set termout off
SQL> spool c:¥temp¥emp7900.txt
SQL> @emp7900
SQL> spool off
SQL> exit
  (略)
D:¥> type c:¥temp¥emp7900.txt
SQL> @emp7900

ENAME
----------
JAMES

SQL> spool off
D:¥>
```

画面に表示されない

- SQLスクリプトが作成済みで、出力内容を画面に表示したくない場合は SET TERMOUT OFF を設定する

SQL*Plusバインド変数の使用と値の表示

- バインド変数は、SQL*Plusで作成し、PL/SQLおよびSQLで参照することのできる変数

•VARIABLEで設定し、表示はPRINTでおこないます

•複数のスクリプト間で変数の受け渡しができます

```
VARIABLE ret_val NUMBER
BEGIN
 :ret_val := 0;
END;
/
@val01.sql
PRINT ret_val
@val01.sql
PRINT ret_val
```

ファイル：val00.sql

```
BEGIN
  :ret_val := :ret_val + 4;
END;
/
```

ファイル：val01.sql

```
SQL> @val00

PL/SQLプロシージャが正常に完了しました。

PL/SQLプロシージャが正常に完了しました。
   RET_VAL
----------
         4

PL/SQLプロシージャが正常に完了しました。
   RET_VAL
----------
         8

SQL>
```

SQLスクリプト終了時にOSにエラーを返却

- EXITおよびWHENEVER SQLERROR を利用する

```
WHENEVER SQLERROR EXIT SQL.SQLCODE ROLLBACK
SELECT * FROM &tname
 WHERE empno = 9999;
EXIT 0
```

ファイル：exit01.sql

```
D:¥> sqlplus -s scott/tiger
@exit01
tnameに値を入力してください： emp
旧 1: SELECT * FROM &tname
新 1: SELECT * FROM emp

レコードが選択されませんでした。

D:¥> echo %errorlevel%
0
```

```
D:¥> sqlplus -s scott/tiger
@exit01
tnameに値を入力してください： em
旧 1: SELECT * FROM &tname
新 1: SELECT * FROM em
SELECT * FROM em
              *
行1でエラーが発生しました。:
ORA-00942: 表またはビューが存在しません。

D:¥> echo %errorlevel%
942
```

存在しないテーブル名

※ SQL.SQLCODEはUNIX系OSでは終了コードが1byteしかないので、
256で割った余りを返します（KROWN#10025 ）

OTNセミナーオンデマンド

コンテンツに対する
ご意見・ご感想を是非お寄せください。

OTNオンデマンド　感想

http://blogs.oracle.com/oracle4engineer/entry/otn_ondemand_questionnaire

上記に簡単なアンケート入力フォームをご用意しております。

セミナー講師/資料作成者にフィードバックし、
コンテンツのより一層の改善に役立てさせていただきます。

是非ご協力をよろしくお願いいたします。

OTNセミナーオンデマンド

日本オラクルのエンジニアが作成したセミナー資料・動画ダウンロードサイト

ORACLE TECHNOLOGY NETWORK
OTNセミナー オンデマンド

掲載コンテンツカテゴリ(一部抜粋)
Database 基礎
Database 現場テクニック
Database スペシャリストが語る
Java
WebLogic Server/アプリケーション・グリッド
EPM/BI 技術情報
サーバー
ストレージ

100以上のコンテンツをログイン不要でダウンロードし放題

データベースからハードウェアまで充実のラインナップ

毎月、旬なトピックの新作コンテンツが続々登場

例えばこんな使い方

- 製品概要を効率的につかむ
- 基礎を体系的に学ぶ/学ばせる
- 時間や場所を選ばず（オンデマンド）に受講
- スマートフォンで通勤中にも受講可能

毎月チェック！

コンテンツ一覧 はこちら
http://www.oracle.com/technetwork/jp/ondemand/index.html

新作＆おすすめコンテンツ情報 はこちら
http://oracletech.jp/seminar/recommended/000073.html

OTNオンデマンド

ORACLE

オラクルエンジニア通信

オラクル製品に関わるエンジニアの方のための技術情報サイト

技術資料 — インストールガイド・設定チュートリアルetc. 欲しい資料への最短ルート

アクセスランキング — 他のエンジニアは何を見ているのか？人気資料のランキングは毎月更新

特集テーマ Pick UP — 性能管理やチューニングなど月間テーマを掘り下げて詳細にご説明

技術コラム — SQLスクリプト、索引メンテナンスetc. 当たり前の運用/機能が見違える!?

http://blogs.oracle.com/oracle4engineer/

オラクルエンジニア通信

oracletech.jp

ITエンジニアの皆様に向けて旬な情報を楽しくお届け

製品/技術情報
Oracle Databaseっていくら？オプション機能も見積れる簡単ツールが大活躍

セミナー
基礎から最新技術までお勧めセミナーで自分にあった学習方法が見つかる

スキルアップ
ORACLE MASTER！試験頻出分野の模擬問題と解説を好評連載中

Viva! Developer
全国で活躍しているエンジニアにスポットライト。きらりと輝くスキルと視点を盗もう

http://oracletech.jp/

oracletech

あなたにいちばん近いオラクル

Oracle Direct

まずはお問合せください

Oracle Direct

システムの検討・構築から運用まで、ITプロジェクト全般の相談窓口としてご支援いたします。
ステム構成やライセンス/購入方法などお気軽にお問い合わせ下さい。

Web問い合わせフォーム

専用お問い合わせフォームにてご相談内容を承ります。
http://www.oracle.co.jp/inq_pl/INQUIRY/quest?rid=28

※フォームの入力にはログインが必要となります。
※こちらから詳細確認のお電話を差し上げる場合がありますので
ご登録の連絡先が最新のものになっているかご確認下さい。

フリーダイヤル

0120－155－096

※月曜～金曜
9:00～12:00、13:00～18:00
（祝日および年末年始除く）

ORACLE


ORACLE®