Balíčky

Balíčky

- **Balíček (package)** je databázový objekt, který seskupuje/zapouzdřuje logicky související typy, položky a podprogramy jazyka PL/SQL.
- Při odkazování se na jména objektů v balíčku je název balíčku předřazován jménům objektů – je jmenným prostorem (v jednom schématu pak lze mít např. dvě stejně pojmenované funkce, ovšem v různých balíčcích).
- Balíček se obvykle skládá ze dvou částí:
  - **Deklarační část (specification)** – popis rozhraní balíčku – deklarace typů, proměnných, konstant, výjimek, deklarace (hlavičky) kurzorů, procedur a funkcí.
  - **Implementační část (body)** – nepovinná část (nemusí být vždy potřeba) – plně definuje kurzory, procedury, funkce (implementuje specifikaci).

Proč používat balíčky?

- **Modularita** – zpřehledňuje, usnadňuje, urychluje vývoj.
- **Snadnější návrh aplikací** – balíčky mají rozhraní (specifikaci), specifikace je kompilovatelná i bez těla balíčku. Následně hned může být vytvářen a kompilován externí kód používající balíček, který bude záviset na deklarační části balíčku, tělo balíčku může být implementováno až později.
- **Ukrývání nepodstatných implementačních detailů** – usnadňuje údržbu a vylepšování kódu, chrání integritu balíčku. Na aplikační úrovni se změny nemusí provádět, pokud se nemění rozhraní balíčku.

Proč používat balíčky?

- **Přidaná hodnota** – veřejné proměnné deklarované v balíčku a kurzory přetrvávají po dobu session databázového uživatele (jsou automaticky v session scope), mohou být snadno sdíleny všemi podprogramy (procedurami, funkcemi). V proměnných lze takto uchovávat data mezi transakcemi, bez nutnosti jejich ukládání v databázi.
- **Výkon** – při prvním použití je celý balíček načten do paměti, při dalších voláních už nemusí být používán přístup na disk. Použití balíčků zabraňuje zbytečné rekompilaci kódu, který závisí na podprogramech v balíčku – pokud je provedena jen interní změna v implementaci balíčku, volající kód zůstává závislý pouze na nezměněné deklaraci v deklarační části balíčku a nemusí být rekompilován.

Vytvoření balíčku

- Syntaxe pro vytvoření balíčku (deklarační a impl. části):

```
CREATE [OR REPLACE] PACKAGE <název balíčku> AS
  FUNCTION ... hlavička až po RETURN <typ> včetně;
  PROCEDURE ... hlavička až po seznam parametrů včetně;
  -- atd. (deklarace procedur a funkcí, kurzorů aj.)
END <název balíčku>;

CREATE [OR REPLACE] PACKAGE BODY <název balíčku> AS
  -- definice celých funkcí, procedur, kurzorů
  FUNCTION ... BEGIN ... celá implementace ... END;
  PROCEDURE ... BEGIN ... celá implementace ... END;
  -- ...
END <název balíčku>;
```

- Spouštění funkcí/procedur: Před název funkce/procedury se předřazuje jméno balíčku: JMENO_BALICKU.JMENO_FUNKCE(...);

Soukromé a veřejné položky balíčku

- Všechny objekty uvedené v deklarační části balíčku jsou veřejně dostupné vně balíčku (**public**).
- Veřejné proměnné a kurzory jsou platné po dobu platnosti session uživatele – lze je použít pro uchování hodnot sdílených více transakcemi v session. Při ukončení session jsou hodnoty zneplatněny.
- Objekty, které mají být pouze interní pro daný balíček (**private**, navenek neviditelné), se deklarují jen v těle balíčku, v deklarační části se neuvádějí:
  - Např. soukromé proměnné sdílené více procedurami.
- Rozhraní obecně, tj. i deklarační část balíčků, by měla být přehledná, jednoduchá, dobře definovaná. Čím méně veřejných položek, tím lépe se bude balíček udržovat a modifikovat.

Kurzory v balíčku

- Také kurzory lze jako znovupoužitelné jednotky deklarovat v balíčku. Kurzor pak může být použit různými bloky PL/SQL kódu (v balíčku, mimo balíček). Související SELECT se nemusí v kódu nikde zbytečně opakovat.

- Syntaxe deklarace kurzorů v balíčku (nově použití RETURN):

```
CREATE [OR REPLACE] PACKAGE <jmeno_balicku> AS
  CURSOR <jmeno_kurzoru> [(<parametry...>)]
    RETURN <typ_zaznam>;
  ...
END <jmeno_balicku>;

CREATE [OR REPLACE] PACKAGE BODY <jmeno_balicku> AS
  CURSOR <jmeno_balicku> [(<parametry...>)]
    RETURN <typ_zaznam> IS
    SELECT ...;
  ...
END <jmeno_balicku>;
```

Kurzory v balíčku

- Za klíčovým slovem RECORD v deklaraci kurzoru je uveden datový typ záznam (název typu), což může být uživatelem deklarovaný typ nebo např. <jmeno_tabulky>%ROWTYPE;
- Jeden kurzor (a související SELECT) může být využíván řadou aplikací (a snadno udržován, dokumentován).
- Uživateli kurzoru „zabaleného“ do balíčku stačí znát:
  - Jaké sloupce kurzor vybírá (typ záznamu),
  - jak jsou vracené řádky řazeny (v tomto smyslu by měly být kurzory okomentovány v deklarační části balíčku).

PRAGMA SERIALLY_REUSABLE

- Ve výchozím nastavení je balíček při použití načten do operační paměti a zůstane tam po celou dobu session.
  - To je problematické u aplikací, u nichž je session dlouho otevřená.
  - Také je problém pokud používáte proměnné v balíčku (protože se ve výchozím nastavení znovu-používají).
- Tyto problémy se dají lehce vyřešit přidáním PRAGMA SERIALLY_REUSABLE k definici balíčku.

Přetěžování podprogramů v balíčku

- Jeden balíček může deklarovat (a definovat) více procedur/funkcí se stejným názvem.
- Tyto procedury/funkce se však musí lišit alespoň typem (nebo počtem) parametrů, aby bylo při volání zřejmé, o jakou máme zájem.

Zásady psaní vlastních balíčků

- Balíčky by měly být psány co nejobecněji, aby byly znovupoužitelné.
- Zbytečně by neměla být duplikována funkcionalita poskytovaná standardními balíčky Oracle.
- Rozhraní (specifikace) balíčků by měla být přehledná, malá, bez duplicit, definovaná ještě před psaním těla balíčku. Měla by obsahovat pouze takové typy, podprogramy apd., které musí být viditelné pro uživatele balíčku (uživatelé by neměli mít možnost se upínat na nepodstatné implementační detaily balíčku, aby se dal balíček snadno udržovat/modifikovat).

Řetězec závislostí DB objektů

- Objekty v databázi na sobě mohou záviset – např. procedura používá ve svém kódu název pohledu, který je definován nad určitou tabulkou → **řetězec závislostí**.
- Zjištění řetězce závislostí objektů, které vlastní přihlášený uživatel:

```
SELECT name, type, referenced_name, referenced_type
FROM user_dependencies
WHERE referenced_owner = USER
ORDER BY name;
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE
T_UCASTNIK_LETU_MAX_CEST	TRIGGER	LET	TABLE
T_ZAMESTNANEC_AKTIVITA	TRIGGER	ZAMESTNANEC	TABLE
T_ZAMESTNANEC_AKTIVITA_UPD	TRIGGER	ZAMESTNANEC	TABLE
T_ZAMESTNANEC_INT_PRACE	TRIGGER	ZAMESTNANEC	TABLE
T_ZAMESTNANEC_INT_PRACE_UPD	TRIGGER	ZAMESTNANEC	TABLE
T_ZAMESTNANEC_SEQ	TRIGGER	ZAMESTNANEC_SEQ	SEQUENCE
T_ZAMESTNANEC_SEQ	TRIGGER	DUAL	NON-EXISTENT
T_ZAMESTNANEC_SEQ	TRIGGER	ZAMESTNANEC	TABLE
T_ZAMESTNANEC_SEQ_UPD	TRIGGER	ZAMESTNANEC	TABLE
T_ZAMESTNANI_SEQ	TRIGGER	ZAMESTNANI_SEQ	SEQUENCE

Platnost DB objektů

- Zjištění, zda je databázový objekt platný (zkompilovaný, připravený k použití):

```
SELECT object_name, object_type, status FROM user_objects;
```

OBJECT_NAME	OBJECT_TYPE	STATUS
ZVYSENI_PLATU	PROCEDURE	VALID
ZASTAVKA	TABLE	VALID
ZAMESTNANI_SEQ	SEQUENCE	VALID
ZAMESTNANI	TABLE	VALID
ZAMESTNANEC_SEQ	SEQUENCE	VALID
ZAMESTNANEC	TABLE	VALID
UCASTNIK_LETU	TABLE	VALID

- Zjištění řetězce závislostí DB objektů včetně platnosti objektů:

```
SELECT ud.name, ud.type, uo.status,
ud.referenced_name, ud.referenced_type
FROM user_dependencies ud, user_objects uo
WHERE ud.name = uo.object_name AND ud.type = uo.object_type
AND ud.referenced_owner = USER
ORDER BY name DESC;
```

Porušení řetězce závislostí

- Např. při změně struktury tabulky (ALTER TABLE) se zneplatní pohledy definované nad touto tabulkou, procedury využívající tyto pohledy (všechny závislé objekty získají status INVALID).
- Předtím, než budou moci být neplatné objekty znovu použity, musí se znovu zkompilovat. Databázový systém provede kompilaci automaticky při jejich prvním použití, často je ale potřeba opětovně kompilovat dlouhou řadu objektů, což je náročné na výkon.

Balíčky redukují závislosti

- Balíčky redukují množství objektů, které je potřeba rekompilovat, když se změní kód nějaké procedury, funkce v balíčku:
  - Vnější kód (mimo balíček) používající proceduru/funkci z balíčku je **závislý jen na deklaraci** procedury/funkce ze specifikace (rozhraní) balíčku (tj. vlastně na hlavičce procedury/funkce).
  - Pokud se nezmění rozhraní procedury/funkce, závislý vnější kód se při změně těla procedury/funkce v implementační části balíčku nezneplatní a nemusí se zbytečně rekompilovat (**balíčky navyšují výkon**).
  - Specifikace a tělo balíčku jsou dva samostatné databázové objekty. Tělo balíčku závisí na specifikaci balíčku (pokud se změní specifikace, tělo se rekompiluje).

Přístupová práva k balíčkům a podprogramům

- Uživatelům lze přiřazovat právo kompilovat a spouštět procedury, funkce – **privilege EXECUTE**.

```
GRANT EXECUTE ON <název procedury/funkce>
  TO <jméno uživatele>;
```

```
Např. GRANT EXECUTE ON moje_funkce TO public;
  přiřadí právo EXECUTE všem uživatelům.
```

- Následně lze právo EXECUTE uživatelům odnímat:

```
REVOKE EXECUTE ON <název procedury/funkce>
  FROM <jméno uživatele>;
```

- Přiřazení privilegia EXECUTE k danému balíčku umožní příslušným uživatelům spouštět všechny procedury a funkce v balíčku, vidět deklarační část (specifikaci) balíčku a přistupovat k proměnným, typům ve specifikaci balíčku:

```
GRANT EXECUTE ON <název balíčku> TO <jméno uživatele>;
```

Přístupová práva k balíčkům a podprogramům

- EXECUTE nelze přiřadit zvlášť pro deklarační část a zvlášť pro tělo balíčku (privilegium se týká vždy deklarační části a nelze jej žádným způsobem aplikovat na tělo balíčku).
- EXECUTE ještě nedává právo vidět kód procedur/funkcí, nebo jej dokonce měnit.
- Právo číst a modifikovat PL/SQL kód má vlastník kódu a administrátor systému (DBA). Aby mohl kód číst někdo další, musí mít práva k určitým tabulkám v datovém slovníku.
- Neexistuje žádné právo, které by uživateli umožňovalo modifikovat konkrétní balíčky z jiného schématu (tj. balíčky, které nevlastní). Modifikovat je může pouze vlastník a administrátor, který má privilegium CREATE ANY PROCEDURE. Poskytnutí tohoto privilegia jinému uživateli je velmi nebezpečné – uživatel by mohl měnit kód uživatelů, kteří mohou mít vyšší práva, a provádět tak činnosti, ke kterým není normálně oprávněn.