Skúška DS2

Výnimky

Syntax

```
BEGIN
  ...
EXCEPTION
  WHEN exception_name THEN
    ...
END
```

Definovaná užívateľom

```
DECLARE
  too_many_records EXCEPTION
BEGIN
  START TRANSACTION
  RAISE too_many_records
  COMMIT
EXCEPTION
  WHEN too_many_records THEN
    DBMS_OUTPUT.PUT_LINE("Too many...")
    ROLLBACK
  WHEN others THEN
    ROLLBACK
END
```

Procedúry

Anonymné

```
DECLARE
  ...
BEGIN
  ...
END
```

Pomenované

```
CREATE OR REPLACE PROCEDURE
  GetName(p_login VARCHAR2)
AS
  ...
BEGIN
  DMBS_OUTPUT.PUT_LINE('TST001')
END
```

Funkcie

```
CREATE OR REPLACE FUNCTION
  GetName(p_login VARCHAR2)
RETURN VARCHAR2
AS
  ...
BEGIN
  RETURN 'TST001'
END
```

Vstupné a výstupné parametre

```
CREATE OR REPLACE PROCEDURE
  GetName(
    p_login IN VARCHAR,
    p_output OUT VARCHAR
  )
AS
  ...
BEGIN
  SELECT name INTO p_output FROM Student
    WHERE login = p_login
END
```

<– Použitie

```
DECLARE
  v_name VARCHAR2
BEGIN
  GetName('TST001', v_name)
  DBMS_OUTPUT.PUT_LINE(v_name)
END
```

Triggery

```
CREATE OR REPLACE TRIGGER OnLoginChange
  AFTER #(BEFORE/AFTER/INSTEAD OF)
    INSERT OR #(INSERT/UPDATE/DELETE)
    UPDATE OF login
  ON Student FOR EACH ROW
BEGIN
  CASE
    WHEN INSERTING THEN ... #premenné :OLD, :NEW
    WHEN UPDATING('login') THEN ...
  END CASE
END
```

Máme aj compound(zložené) triggery. Dokážu pracovať v určitých časových bodov počas vykonávania triggeru (BEFORE STATEMENT, BEFORE EACH ROW, AFTER EACH ROW, AFTER STATEMENT)

Kurzory

Pomocné premenné na prechádzanie výsledkov vykonaného dotazu.

- **Implicitné**: vytvárajú sa automaticky po vykonaní dotazu INSERT/UPDATE/DELETE
- **Explicitné**: definujeme v procedúre ako premennú, často sa používa pri prechádzaní výsledkov SELECTu

Explicitný

```
DECLARE
  CURSOR c_students
    IS SELECT * FROM Student
  v_record Student%ROWTYPE
BEGIN
  OPEN c_students
  LOOP
    FETCH c_students INTO v_record
    EXIT WHEN c_students%NOTFOUND
  END LOOP
  CLOSE c_students
END
```

Implicitný

```
DECLARE
BEGIN
  FOR student IN (SELECT * FROM Student)
  LOOP
  END LOOP
END
```

Balíky

Zoskupujú objekty (funkcie, procedury, premenné, výnimky...) do jedného priestoru.

- globálne premenné a výnimky mimo procedúru/funkciu
- verejné a súkromné funkcie a procedúry
- vlastný menný priestor s ľubovoľnými menami objektov
- jednoduchšia orientácia

Balíky majú špecifikáciu a telo:

```
#špecifikácia
CREATE OR REPLACE PACKAGE pkg AS
  verejné premenné, typy, výnimky, procedury
END

#telo
CREATE OR REPLACE PACKAGE BODY pkg IS
  privátne objekty...
END
```

Hromadné SQL operácie

- **BULK COLLECT**: Naviazanie výstupnej kolekcie s PL/SQL procesorom, hormadne priradí niekoľko záznamov do kolekcie na jeden krát.

```
Studs Student%ROWTYPE
SELECT * BULK COLLECT INTO Studs FROM Student
```

- **FORALL**: Naviazanie vstupnej kolekcie s SQL procesorom, umožňuje hromadné vykonanie DML operácií z kolekcie.

```
FORALL i IN Studs.FIRST..Studs.LAST
  UPDATE ....
```

Statické PL/SQL

SQL príkazy, ktoré dokážeme volať priamo v PL/SQL `(SELECT, INSERT, UPDATE, DELETE, MERGE LOCK TABLE, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION)`

- premenné sú implicitne viazané

Dynamické PL/SQL

Volanie akýchkoľvek SQL príkazov počas behu aplikácie.

- nemožno overiť správnosť syntaxu
- vstavujeme sa riziku SQL injection.
- spúšťame pomocou `EXECUTE IMMEDIATE`
- používame len prípade, že nám nestačí Statické PL/SQL
- premenné nie sú implicitne viazané (musíme použiť USING)

Viazané premenné

Umožňujú využiť už vytvorený plán vykonávania dotazu aj s odlišnými vstupnými parametrami.

ADO.NET C# a JDBC Java

Databázové konektory pre programovacie jazyky, ktoré:

- podporujú viazané premenné (`PreparedStatement`)

```
Connection conn = DriverManager.getConnection(...);
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM Student WHERE vsp
>= ?");
stmt.setDouble(1, 88);
ResultSet res = stmt.executeQuery();
```

SQL Injection

Útok, kde SQL dotaz s neošetreným (nesanitovaným) vstupom dokáže kompromitovať databázu `(WHERE user_id = 1 OR 1=1)`.

SQL Injection zabránime:

- Používaním statického SQL.
- Používaním viazaných premenných (PDO, PreparedStatements).
- Riadením prístupových práv, aby sa utočník z kompromitovanej časti databázy nerozšíril na ďalšie.

Objektovo orientované databázové systémy

Prvky OOP sú dnes implementované vo všetkých známych DBMS.

Prínos OOP v DBMS:

- vlastné dátové typy

```
CREATE TYPE Address AS OBJECT (street VARCHAR2(50), city VARCHAR2(30))
```

- uložené procedúry, triggery

```
CREATE OR REPLACE PROCEDURE GetName(p_login VARCHAR2)
```

- kolekcie (polia, nested tabuľky, asociatívne polia)

```
DECLARE
  TYPE StudentList IS TABLE OF Student
  students StudentList
```

- dedičnosť

```
CREATE TYPE CompanyAddress UNDER Address (ico VARCHAR2(12))
```

- ukazatele, referencie, dereferencie

```
CREATE TYPE AddressType (address_type_id INT, address_type VARCHAR2)
CREATE TYPE CompanyAddress UNDER Address (ico VARCHAR2(12), type ref AddressType)
```

Dátové typy

- dáta (atribúty)
- operácie (metódy)
  - členské, statické, konštruktor

Transakcie

Logická, atomická jednotka práce s databázou, ktorá obsahuje viac databázových operácií (`SELECT, INSERT, UPDATE, DELETE`)

- `COMMIT` - úspešné uloženie transakcie
- `ROLLBACK` - vrátenie aktualizácií do pôvodného stavu pri neúspešnej transakcii

Úlohou transakcie je previesť korektný stav databáze do ďalšieho korektného stavu, medzi nimi môže byť databáza aj v nekorektnom stave (v strede transakcie)

`COMMIT` zavádza potvrdzovací bod v čase, kedy je DB v korektnom stave, `ROLLBACK` vracia DB do stavu, v ktorom bola pri vykonávaní `BEGIN TRANSACTION`

```
BEGIN TRANSACTION;

try {
  UPDATE Account 345 { balance -= 100; }
  UPDATE Account 999 { balance += 100; } #chyba v tomto riadku => výsledok neuloží
  COMMIT;
} catch (SqlException) {
  ROLLBACK;
}
```

Chyby v DBMS

- **Lokálne chyby**: chyba v dotaze
- **Globálne chyby**
  - systémové chyby (soft crash)
  - chyby disku (hard crash)

Log súbor

Súbor ukladá informácie o vsetkých vykonaných operáciach dopredne (pred vykonaním sa uloží do logu).

- DBMS sa dokáže zotaviť pomocou údajov z logu `UNDO/REDO` operáciami transakcií
- pri `ROLLBACK` sa podľa logu vieme vrátiť na pôvodné hodnoty
- do logu zapisujeme sekvenčne (rýchlejšie), do DB sa zapisuje náhodným prístupom na disk

ACID

- **A** - Atomicity (v transakcii sú prevedený všetky príkazy transakcie alebo žiaden)
- **C** - Correctness (transakcia prevídza korektný stav na iný korektný stav DB)
- **I** - Isolation (transakcie sú navzájom izolované, zmeny sa prepíšu až po `COMMIT`)
- **D** - Durability (po potvdení transakcie sú zmeny v DB natrvalo, aj pri globálnej chybe)

Zotavenie databázy

Pri globálnej chybe je databáza v korektnom stave, sú avšak stratané aktualizácie v hlavnej pamäti (v logu sú).

Počas zotavenia sa vykonávajú operácie:

- **UNDO**: stav transakcie prerušenej chybou nepoznáme, zmena musí byť zrušená
- **REDO**: transakcia bola úspešne ukončená (`COMMIT`), zmeny ale neboli prenesené z **LOGU do DB**, transakcia musí byť prepracovaná

Techniky zotavenia DBMS

- Odložené aktualizácie (NO-UNDO/REDO)
- Okamžité aktualizácie (UNDO/NO-REDO)
- Kombinované aktualizácie (UNDO/REDO)

Odložené aktualizácie (NO-UNDO/REDO)

- aktualizácie sú počas transakcie ukladané do pamäte
- po potvrdení transakcie je uložená do LOGU a potom do DB
  - **NO-UNDO** - nie je potrebné UNDO, DB nebola aktualizovaná pred `COMMIT`
  - **REDO** - vykoná sa v prípade, že nové zmeny boli zapísané do LOGU, ale nie do DB
- Vyšší výkon, ale hrozí pretečenie pamäte

Okamžité aktualizácie

- po každej aktualizácii sú do logu ukladané pôvodné hodnoty
- po potvrdení transakcie je do logu zapísaný `COMMIT` záznam
  - **UNDO** - ak transakcia nebola potvrdená, bude pri zotavení zrušená (v logu sú staré hodnoty, v DB nové)
  - **NO-REDO** - nové hodnoty sú už v DB po aktualizácii
- Nižší výkon (počet diskových operácií)

Kombinovaná technika (UNDO/REDO)

V praxi sa používa UNDO/REDO algoritmus.

- **Zápis aktualizácií**
  - nové ohdnoty sú zapísané do logu po `COMMIT`
  - v intervaloch sa vytvárajú **KONTROLNÉ BODY**, v tomto bode sa zapíšu aktualizácie transakcií do DB:
    - **aktuálne vykonávané transakcie**, ktorých pôvodné hodnoty sú uložené do logu pre UNDO
    - **potvrdené transakcie**, ktoré neboli uložené v poslednom **KB**
    - zápis **záznamu** KB do logu
- **Zotavenie**
  - Vytvoríme 2 zoznamy transakcií **UNDO** a **REDO**
  - Do **UNDO** vložíme transakcie, ktoré neboli potvrdené (`COMMIT`) pred posledným **KB**. **REDO** je prázdny.
  - DBS prechádza záznamy v LOGU od posledného **KB**:
    - ak je v transakcii *T* nájdený v logu `COMMIT`, *T* sa presunie z **UNDO –> REDO**
  - DBS prechádza LOG spätne a ruší aktualizácie z **UNDO** (zapisuje pôvodné dáta z logu)
  - DBS prechádza LOG dopredne a prepracováva transakcie z **REDO** uložené v logu
  - DB je **korektnom stave**

Zotavenie pri chybe disku

- Chyby disku sa riešia redundantnými diskmi (RAID), ak je aj tak DB poškodená:
  - môžeme načítať dáta z DUMPu zo zálohy
  - prechádza sa log a prepracujú sa potvrdené transakcie po čase zálohy

Záchranné body (SAVEPOINTs)

Umožňujú rozdeliť transakciu na menšie časti, v prípade `ROLLBACK` sa vraciame na posledný **ZB**, nerušíme celú transakciu. **ZB** nie je ekvivalent `COMMIT`.

```
SAVEPOINT <name>
ROLLBACK TO <name>
RELEASE <name>
```

- Po ukončení transakcie sa všetky **ZB** zrušia

Úroveň izolácie START TRANSACTION

- READ UNCOMMITED
- READ COMMITED
- REPEATABLE READ
- SERIALIZABLE

Problémy paralelného behu

Problém nastáva, keď 2 transakcie sú čítať/zapisovať rovnaký záznam. Existujú konflikty:

- RW, WR, WW

Problémy:

- strata aktualizácie
- nepotvrdená závislosť
- nekonzistentná analýza
- neopakovateľné čítanie
- výskyt fantómov

Strata aktualizácie

- transakcia A prepíše dáta transakcie B

A	B
READ t	-
-	READ t
WRITE t	-
-	WRITE t

Nepotvrdená závislosť

- transakcia A, prečíta nepotvrdené dáta zapísané transakciou B

A	B
-	WRITE t
READ t	-
-	ROLLBACK

Nekonzistentná analýza

- transakcia A opakovane číta dáta, ktoré upravuje aj transakcia B

ACC1 { money: 10 }, ACC2 { money: 20 }

A (SUM=0)	B
SUM += ACC1.MONEY	-
-	ACC1.MONEY = 30
-	ACC2.MONEY = 10
-	COMMIT
SUM += ACC2.MONEY	

Neopakovateľné čítanie

- transakcia A prečíta 2x rovnakú tabuľku, modifikovanú transakciou B, pričom 2x prečíta rozdielne dáta

A	B
READ t	-
-	UPDATE t
-	COMMIT
READ t	-

Výskyt fantómov

- transakcia A prečíta 2x rovnakú tabuľku, modifikovanú transakciou B, pričom 2x prečíta rôzny počet riadkov

A	B
READ t	-
-	INSERT t
-	COMMIT
READ t	-

Techniky riadenia paralelného behu

- **správa verzií**
  - optimistický prístup, predpokladáme, že transakcie sa nazvájom ovplyvňovať nebudú
  - vytvára kópie dát a sleduje, ktorá verzia je viditeľná pre ostatné transakcie
- **zamykanie**
  - pesimistický prístup, spravuje prístup k 1 kópii dát pomocou zámkov

Zamykanie

Ak transakcia A chce čítať alebo zapisovať objekt v DB, tak požiada o zámok na tento objekt.

Typy zámkov

- **zdieľaný (S)**: požadovaný pred čítaním záznamu
- **výlučný (X)**: požadovaný pred aktualizáciou záznamu (insert ,update, delete)

Zámky sa prideľujú automaticky bez explicitného žiadania užívateľom.

- **A drží výlučný zámok (X) na záznam t:**
  - akýkoľvek paralelný zámok nie je umožnený
- **A drží zdieľaný zámok (S) na záznam t:**
  - požiadavka na paralelný, výlučný zámok (X) nie je umožnená
  - požiadavka na paralelný, zdieľaný zámok (S) je umožnená

Ak požadovaný zámok nemôže byť pridelený, transakcia prejde do čakacieho stavu (wait state). DBS radí čakajúci transakcie do fronty, aby transakcie nezostali v tomto stave navždy

Prísne, dvojfázové zamkynanie

- **Fáza 1**: požadovanie zámku
- **Fáza 2**: uvoľnovanie zámkov:
  - ak je nejaký zámok uvoľnený, nie je možnie požiadať ďalší zámok
  - zámky sú automaticky uvoľnené na konci transakcie

Uväznutie
Pri vzájomnom čakaní na zámky môže dôjsť k uväznutiu.

Riešenie uväznutia:
- **detekcia uväznutia**:
  - nastavenie časových limitov (po prekročení časového limitu `ROLLBACK`)
  - detekcia cyklu v grafe Wait-For (jedna a uväznutých transakcií sa `ROLLBACK`ne)
- **prevencia uväznutia**:
  - nastavenie časových razítok:
    - každá transakcia dostane časové razítko (začiatok transakcie)
    - ak transakcia A požaduje zámok na zamknutý záznam transakciou B:
      - **Wait-Die**: ak A je staršia ako B, A čaká; ak A je mladšia ako B, zruší sa a spustí znova
      - **Wound-Wait**: ak A je staršia ako B, zruší sa a spustí znova; ak A je mladšia ako B, čaká
    - vysoký počet operácií `ROLLBACK`

Úroveň izolácie
- Vyššia úrovňa izolácie značí nižšiu priepustnosť. Nižšia úroveň značí vyššiu priepustnosť.
- **Úrovne**:
  - READ UNCOMMITED
    - T môže čítať údaje, ktoré neboli potvrdené inou transakciou (neexistuje v ORACLE)
  - READ COMMITED
    - každý dotaz spustený transakciou vidí len dáta potvrdené pred začiatkom dotazu
  - REPEATABLE READ
  - SERIALIZABLE
    - každý dotaz spustený transakciou vidí len dáta potvrdené pred začiatkom transakcie
    - **vhodná pre**:
      - veľké databázy, krátke transakcie s málo záznamami
      - nízka pravdepodobnosť na paralelné transakcie, ktoré modifikujú rovnaké záznamy
      - dlhé transakcie, ktoré hlavne čítajú
  - READ-ONLY
    - ako serializable, akurát transakcie nesmie aktualizovať dáta
    - **vhodná pre**: generáciu zostáv s konzisteným obsaom
- Úroveň SERIALIZABLE značí maximálnu izoláciu od ostatných paralelných transakcií

Tabuľka typu halda

Jedná sa o stránkované pole, záznamy sú uložené v stránkach/blokoch o veľkosti najčastejšie 8kB

- záznamy nie sú usporiadané
- ukladá sa na koniec O(1) alebo na prvé voľné miesto
- pri mazaní sa záznamy značia, nemžú sa fyzicky
- kvôli ne-mazaniu sa počet blokov haldy neznižuje
- **zložitosti**:
  - vkladanie O(1)..O(n)
  - hľadanie O(n)
  - mazanie (n)
  - rozsah O(n)

Index typu B-strom

- neobsahuje celé záznamy, ale indexované zoradené atribúty (kľúč)
- obsahuje ROWID, ktoré odkazuje do haldy na kompletný záznam tabuľky
- uzol B-stromu {Kľúč, ROWID}
- jednoducho stránkovateľný
- vyvážený
- Všetky **zložitosti** O(logn)

Zložený index

- dátová štruktúra, ktorá umožňuje rýchle vyhľadávanie v taubľke
- index je implementovaný B-stromom, pričom jeden uzol má {Kľúč, ROWID}
- Kľuč je obsahuje indexované zoradené atribúty
- **vhodný pre dotazy**:
  - bodové, rozsahové dotazy, ktoré odpovedajú lexikografickému usporiadaniu
- **nevhodný pre dotazy**:
  - ktoré nevyužívajú index => prechádza sa haldou

Netriviálna funkcia

CreateResponse

Signatúra netriviálnej funkcie **CreateResponse(p_text, p_id_ad, p_id_company, p_start_at, p_end_at)**.

- Spustí transakciu

```
start transaction.
```

- Zistí počet kreditov firmy, ktorá v zastúpení odpovedá na inzerát.

```
v_credits = select credits from Company where id_company = p_id_company;
```

- Pokiaľ počet kreditov je menší ako 10, vráti 0

```
if v_credits < 10 {
  return 0;
}
```

- Vloží odpoveď na inzerát

```
insert into Ad_Response(text, id_ad, id_company, created_at)
values (p_text, p_id_ad, p_id_company, NOW());
```

- Odčíta firme čiastku 10 kreditov, ktorá odpovedá cene zahájenia komunikácie

```
update Company set credits = v_credits where id_company = p_id_company;
```

- Skontroluje či `p_start_at` a `p_end_at` nie sú null a `p_start_at` < `p_end_at`, pokiaľ platí, vloží záznam do **Ad_Inteview**

```
if p_start_at != null and p_end_at != null and p_start < p_end_at {
  insert into Ad_Interview(id_ad_response, start_at, end_at)
  values (last_insert_id(), p_start_at, p_end_at);
}
```

(last_insert_id() v tomto prípade odpovedá ID poslednej vloženej odpovede.)

- Ukončíme transakciu a vrátime 1

```
commit;
return 1;
```