Sprawozdanie z projektu “Baza danych do projektu IAESTE CaseWeek”

Przemysław Poljański
Maciej Kędzielski
grupa śr. 10-12

Omówić zagadnienia: procedury składowane, przyjmowanie i zwracanie parametru
Procedury składowane (nazywane też osadzonymi lub po prostu procedurami) są w języku SQL odpowiednikiem funkcji w pojęciu większości popularnych języków programowania. Umożliwiają one stworzenie zapytania zapisanego w bazie danych SQL, mogącego przyjmować i zwracać pewne parametry. Zapytanie to jest dostępne w bardzo prostej formie dla każdego użytkownika bazy danych oraz przenoszone z każdym backupem. Przyjmowanie parametru polega na wprowadzaniu do procedury danych z zewnątrz, które przy każdym jej wywołaniu mogą być inne i powodować inny wynik wykonania procedury. Zwracanie parametru jest po prostu przekazaniem wyniku działania procedury (funkcji) po jej zakończeniu w postaci, która może następnie być dalej przetwarzana w programie.

Omówić zagadnienia teoretyczne realizowane w punktach g)-l) etapu 3
CURSOR - jest to obiekt służący do przechowywania danych wybranych przy pomocy polecenia SELECT w celu ich dalszego wykorzystania w procedurach. Dane zawarte w kursorze można przypisać do zmiennych przy pomocy komendy FETCH. Kursor przed użyciem należy zadeklarować i otworzyć, a po użyciu zamknąć.
FOR - jest to komenda tworząca pętlę z wbudowanym licznikiem oraz przypisanymi do niego wartościami dla kolejnych wykonań lub warunkiem wykonania. Ciało pętli znajduje się pomiędzy komendami LOOP i END LOOP. Zawarte wewnątrz instrukcje wykonują się cyklicznie, przy czym w każdym kolejnym obejściu pętli wartość licznika jest inna i z reguły wpływa na wynik.
COMMIT/ROLLBACK - są to polecania kończące transakcję. COMMIT powoduje zatwierdzenie zmian wywołanych wewnątrz transakcji, a ROLLBACK ich wycofanie i powrót do zapisanego stanu, który zwykle jest stanem z początku wykonywania transakcji.
EXCEPTION - deklarowany przez użytkownika wyjątek. Może zostać wywołany poleceniem RAISE. Wywołanie wyjątku przerywa działanie programu i powoduje przejście do bloku obsługi wyjątku. RAISE może wywoływać także wyjątki wbudowane w język SQL, na przykład ZERO_DIVIDE.
WHILE - jest to komenda tworząca pętlę z warunkiem wykonania. Wykonywać się ona będzie cyklicznie, dopóki warunek wykonania będzie miał wartość TRUE lub wywołana zostanie komenda BREAK. Wykonywaniem pętli można zarządzać również przy użyciu komendy CONTINUE, która przerywa obecne wykonanie pętli i powoduje przejście do kolejnego.
INSERT - komenda umożliwiająca dodanie do bazy danych nowego rekordu. Wymaga podania nazw kolumn, do których będą wprowadzane dane oraz odpowiednich danych.
UPDATE - komenda pozwalająca na zmianę wartości konkretnych pól dla wszystkich rekordów spełniających dany warunek.
DELETE - komenda powodująca usunięcie rekordów spełniających dany warunek.
BEFORE/AFTER - komendy służące do obsługi zdarzeń (TRIGGER). Zdarzenie jest wykryciem przez program pojawienia się konkretnej komendy lub zmiany stanu. Użycie BEFORE lub AFTER decyduje czy procedura obsługi zdarzenia wykona się przed czy po faktycznym wykonaniu zdarzenia.
PACKAGE - pozwala na zgrupowanie kilku procedur w celu wygodniejszego dostępu do nich. Procedury wchodzące w skład obiektu PACKAGE są dostępne w sposób charakterystyczny dla metod klas w programowaniu obiektowym - po kropce (np. nazwa_paczki.funkcja)

Przemyślenie, w jaki sposób można zabezpieczyć utworzone tabele w środowisku wielodostępnym (np. serwer Web) przed jednoczesną edycją przez wiele osób (tematyka PessimisticLock/OptimisticLock)

Pessimistic lock - w momencie uzyskania dostępu do zasobu przez jednego użytkownika, zostaje on zablokowany dla wszystkich innych. Uniemożliwia to powstawanie błędów spowodowanych równoczesną edycją zasobów, ale znacznie zmniejsza dostępność zasobu - może on być zablokowany przez długi czas przez użytkownika, który jedynie odczytuje dane.
Optimistic lock - stan zasobu w momencie uzyskania do niego dostępu jest zapamiętywany i sprawdzany w momencie zakończenia transakcji obejmującej ten zasób. Jeśli podczas transakcji dane nie zmieniły się, jest ona zatwierdzona, a w przeciwnym wypadku odrzucana. Pozwala to na równoczesny dostęp do zasobów wielu użytkownikom, ale może skutkować koniecznością powtarzania transakcji lub niespójnością danych.

Charakterystyka bazy danych i praktycznego celu jej istnienia

Stworzona w projekcie baza danych opisuje strukturę organizacyjną cyklu warsztatów inżynierskich CaseWeek organizowanych przez organizację studencką IAESTE na wielu uczelniach w Polsce. Celem ideowym jej powstania było przyłączenie do internetowego systemu zarządzania wydarzeniem w skali kraju. Zawarte w bazie tabele reprezentują trzy główne elementy wydarzenia (warsztaty, uczestników oraz organizatorów) oraz szczegóły z nimi związane (np. miejsce warsztatu, dane prelegenta, uczelnię). Relacje przedstawione na diagramie ERD reprezentują rzeczywiste relacje zachodzące w wydarzeniu. Na przykład prelegent prowadzący konkretny warsztat jest związany z konkretną firmą lub uczelnią, a z tymi z kolei kontaktuje się konkretny komitet lokalny IAESTE.

Tabelka dla EXPLAIN_PLAN:

Explain plan pozwala wyświetlić plan wykonywania danego zapytania przez kompilator SQL w celu sprawdzenia czy na pewno jest optymalny.

Dla zapytania:

```
EXPLAIN PLAN FOR SELECT * FROM STUDENTS stud
JOIN PERSONS pers ON pers.P_ID = stud.STUD_ID
JOIN ATTENDS att ON att.STUD_ID = stud.STUD_ID
JOIN WORKSHOPS ws ON ws.WS_ID = att.WS_ID;
SELECT * from TABLE(dbms_xplan.display);
```

Otrzymano następującą tabelę:

Plan hash value: 3209688994

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |   618 |   367K|    15  (14)| 00:00:01 |
|*  1 |  HASH JOIN         |           |   618 |   367K|    15  (14)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| PERSONS   |   412 | 64272 |     5   (0)| 00:00:01 |
|*  3 |   HASH JOIN        |           |   618 |   273K|     9  (12)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| STUDENTS |   206 | 52942 |     3   (0)| 00:00:01 |
|*  5 |    HASH JOIN       |           |   618 |   118K|     6  (17)| 00:00:01 |
|   6 |     TABLE ACCESS FULL| WORKSHOPS|     5 |   850 |     2   (0)| 00:00:01 |
|   7 |     TABLE ACCESS FULL| ATTENDS  |   618 | 16068 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Predicate Information (identified by operation id):

--

1 - access("PERS"."P_ID"="STUD"."STUD_ID")
3 - access("ATT"."STUD_ID"="STUD"."STUD_ID")
5 - access("WS"."WS_ID"="ATT"."WS_ID")

Note

- dynamic sampling used for this statement (level=2)

Pisemne wyjaśnienie polecenia SET OPTIMIZER_MODE
Polecenie SET OPTIMIZER_MODE służy do instruowania bazy danych na temat sposobu optymalizacji pobierania danych. Posiada on trzy tryby: *first_rows*, *first_rows_n* oraz *all_rows.* Pierwszy z nich optymalizuje pobieranie danych przy założeniu, że pobrane będzie kilka wierszy. Drugi pozwala na podanie spodziewanej ich liczby i na tym opiera optymalizację. Trzeci zakłada pobieranie wielkich ilości danych i powinien być dla takich przypadków stosowany. *All_rows* jest domyślnym trybem OPTIMIZER_MODE.

Omówić bardzo ogólnie istotę zapytań równoległych
Serwery SQL posiadają możliwość równoległej obsługi zapytań do bazy danych. Aby to zrealizować, dla analizowanych zapytań tworzone są obiekty “exchange operator”, które umożliwiają stworzenie planu obsługi zapytań. Taki plan może być obsługiwany przez kilka wątków. Dzięki stworzeniu planu zapewniona jest kontrola dostępu do danych oraz ich przepływu.