Wykład 10

Implementacja języka SQL w systemach baz danych Oracle – wybrane komendy DDL dodatkowych, przydatnych mechanizmów serwera bazy danych jak sekwencje, wyzwalacze i indeksy.

(Wykład oparty na dokumentacji Oracle i materiałach portalu ORACLE-BASE)

I. Sekwencje

I. 1 Tworzenie sekwencji - polecenie CREATE SEQUENCE

CREATE SEQUENCE [schemat_użytkownika.]nazwa_sekwencji
zestaw_parametrów_sekwencji
;

Parametry sekwencji:

INCREMENT BY liczba_całkowita – definiuje o jaką liczbę będzie zmieniać się wartość sekwencji. Wartości dodatnie tego parametru implikują sekwencję rosnącą, a wartości ujemne – sekwencję malejącą wartość 0 tego parametru nie jest dozwolona. Gdy nie podamy tego parametru, wartość sekwencji będzie wzrastać domyślnie o 1.

START WITH liczba_całkowita – określa pierwszą wartość sekwencji, która powinna być mniejsza od maksymalnej wartości sekwencji rosnącej albo większa od minimalnej wartości sekwencji malejącej. Gdy nie podamy tego parametru, to domyślnie jest to minimalne wartość sekwencji rosnącej albo maksymalna wartość sekwencji malejącej.

MAXVALUE liczba_całkowita – określa maksymalną wartość sekwencji. Może być maksymalnie 28 cyfr znaczących. Musi być równa lub większa od wartości parametru START WITH i musi być większa od wartości parametru MINVALUE.

NOMAXVALUE – jest to parametr domyślny, oznaczający maksymalną wartość sekwencji rzędu 10^{27} dla sekwencji rosnącej albo -1 dla sekwencji malejącej.

MINVALUE liczba_całkowita – określa minimalną wartość sekwencji. Może być maksymalnie 28 cyfr znaczących Musi być równa lub mniejsza niż wartość parametru START WITH i musi być mniejsza od wartości parametru MAXVALUE.

NOMINVALUE – jest to parametr domyślny, oznaczający minimalną wartość sekwencji 1 dla sekwencji rosnącej albo wartość minimalną rzędu -10^{26} dla sekwencji malejącej.

CYCLE – określa sekwencję cykliczną, tzn. taką, która w sposób cykliczny generuje wartości

NOCYCLE – jest to parametr domyślny, oznaczający, że sekwencja rosnąca nie może generować wartości po osiągnięciu swojej wartości maksymalnej, a sekwencja malejąca nie może generować wartości po osiągnięciu swojej wartości minimalnej.

CACHE liczba_całkowita – określa prealokowaną w pamięci liczbę wartości sekwencji. Minimalnie jest to 2, a maksymalnie liczba rzędu 10^{27}. Jednak w praktyce formuła:

(CEIL(MAXVALUE – MINVALUE) / ABS(INCREMENT)

wyznacza maksymalną dozwoloną wartość dla parametru CACHE dla danej sekwencji.

NOCACHE – oznacza, że wartości sekwencji nie będą prealokowane w pamięci. Jeśli nie zdefiniujemy parametrów CACHE ani NOCACHE domyślnie będzie prealokowanych w pamięci 20 wartości sekwencji.

ORDER – gwarantuje, że wartości sekwencji są generowane w kolejności zgłoszonych żądań.

NOORDER – jest to parametr domyślny, oznaczający, że nie zostanie zagwarantowane, iż wartości sekwencji będą generowane w kolejności zgłoszonych żądań.

Przykłady definicji sekwencji:

```
CREATE SEQUENCE S0
START WITH 1
INCREMENT BY 1
NOMAXVALUE;

CREATE SEQUENCE S1
START WITH 0
INCREMENT BY 2
MINVALUE 0
NOMAXVALUE
NOCYCLE;

CREATE SEQUENCE S2
START WITH 0
INCREMENT BY -2
NOMINVALUE
NOCYCLE;
```

CREATE SEQUENCE S4;

Czy ostania sekwencja jest poprawnie zdefiniowana, a jeśli tak, to jakie będą jej parametry?

I. 2 Generowanie wartości sekwencji i sprawdzanie bieżącej wartości sekwencji – operatory `nextval` i `currval`

Nowo utworzona sekwencja nie ma bieżącej wartości! Należy ją dopiero wygenerować.

select S0.nextval from dual;

select S0.currval from dual;

Warto sprawdzić, jaki rezultat dadzą zapytania:

select S1.nextval, S1.currval from dual;

select S1.currval, S1.nextval from dual;

Można też wygenerować kilka wartości sekwencji, używając podklauzuli

CONNECT BY ,

w której podajemy warunek na liczbę wartości sekwencji, jakie mają zostać wygenerowane, używając parametru LEVEL.

Przykładowo:

```
select S2.nextval from dual
connect by level <= 10;
```

I. 3 Zmiana parametrów utworzonej sekwencji - polecenie ALTER SEQUENCE

```
ALTER SEQUENCE [schemat_użytkownika.]nazwa_sekwencji
zestaw_zmienianych_parametrów_sekwencji
;
```

I. 4 Zmiana nazwy sekwencji - polecenie RENAME

```
RENAME stara_nazwa_sekwencji TO nowa_nazwa_sekwencji;
```

I. 5 Usuwanie sekwencji - polecenie DROP SEQUENCE

```
DROP SEQUENCE [schemat_użytkownika.]nazwa_sekwencji;
```

II. Wyzwalacze

II. 1 Proste wyzwalacze DML

Tworzenie lub zmienianie istniejących prostych wyzwalaczy DML:

```
CREATE [OR REPLACE] TRIGGER [schemat_użytkownika.]nazwa_wyzwalacza
{BEFORE | AFTER} transakcja_DML [OR transakcja_DML …] ON tabela
[FOR EACH ROW]
[FOLLOWS [schemat_użytkownika.]nazwa_innego_wyzwalacza lub lista
innych wyzwalaczy]
[ENABLE | DISABLE]
[WHEN (warunek)]
[DECLARE …]
BEGIN ← oznacza rozpoczęcie kodu ciała wyzwalacza
….
[EXCEPTION …]
END; ← oznacza zakończenie kodu ciała wyzwalacza
/
```

FOR EACH ROW – oznacza wyzwalacz wierszowy, tzn. taki, który jest wykonywany natychmiast dla każdego wiersza. Ta własność jest dozwolona jedynie dla prostych wyzwalaczy DML

FOLLOWS … – umożliwia określenie kolejności wykonywania wyzwalaczy zdefiniowanych dla danej tabeli. Po FOLLOWS podajemy listę istniejących wyzwalaczy, które powinny być wykonane przed definiowanym wyzwalaczem.

ENALBE, DISABLE – parametry pozwalające definiować, czy utworzony wyzwalacz będzie włączony czy też wyłączony. Domyślnie wyzwalacze są włączane zaraz po ich utworzeniu.

WHEN (warunek) – klauzula umożliwiająca sformułowanie dodatkowego warunku, który będzie musiał być spełniony, aby operacje wyzwalacza zostały wykonane. W warunkach stosuje się kwalifikatory NEW i OLD bez dwukropka, omówione na następnej stronie w swoich podstawowych zastosowaniach, w których dwukropek jest wymagany :NEW i :OLD.

EXCEPTION – klauzula, w której można zdefiniować własną obsługę wyjątków/błędów.

DECLARE – klauzula pozwalająca definiować zmienne lokalne wyzwalacza

W klauzuli DECLARE nie deklarujemy kwalifikatorów :NEW i :OLD kolumny tabel, rozważanej w tym przypadku raczej jako pole tabeli, tzn. element wiersza tabeli, a formalnie element krotki relacji bazy danych.
:NEW.atrybut – nowa (wstawiana lub mająca zastąpić poprzednią) wartość atrybutu relacji w danej krotce
:OLD.atrybut – poprzednia wartość atrybutu relacji w danej krotce

Rodzaje transakacji_DML mogą być następujące:

DELETE
INSERT
UPDATE
UPDATE OF nazwa_kolumny[, nazwa_kolumny …]

Zastosowanie kwalifikatorów :NEW i :OLD w zależności od typu wyzwalacza:

- wyzwalacze wierszowe typu DELETE – tylko :OLD

- wyzwalacze wierszowe typu UPDATE – zarówno :NEW, jak i :OLD

- wyzwalacze wierszowe typu INSERT – tylko :NEW

BEFORE – wyzwalacz jest wykonywany przed wyzwalającą go transakcją DML. W przypadku wyzwalaczy wierszowych wyzwalacz ten jest wykonywany zanim zostanie zmieniony każdy wiersz, którego dotyczy transakcja DML wyzwalająca ten wyzwalacz.

Wyzwalacze wierszowe typu **BEFORE pozwalają na odczyt i zapis** wartości pól opatrzonych kwalifikatorami :NEW lub :OLD.

AFTER – wyzwalacz jest wykonywany po wyzwalającej go transakcji DML. W przypadku wyzwalaczy wierszowych wyzwalacz ten jest wykonywany po tym, gdy zostanie zmieniony każdy wiersz, którego dotyczy transakcja DML wyzwalająca ten wyzwalacz.

Wyzwalacze wierszowe typu **AFTER pozwalają na odczyt, ale nie zezwalają na zapis** wartości pól opatrzonych kwalifikatorami :NEW lub :OLD.

W celu wskazania, jaka część kodu wyzwalacza ma być wykonana dla danej transakcji DML w przypadku prostego wyzwalacza DML wyzwalanego przez kilka rodzajów transakcji, w kodzie ciała wyzwalacza możemy zastosować predefiniowane przypadki transakcji DML:

DELETING
INSERTING
UPDATING
UPDATING(‘Nazwa_kolumny’)

Przykłady:

- Wyzwalacz umożliwiający autoinkrementację wartości klucza głównego narzuconego na kolumnę PRACOWNICY.id_p, wykorzystujący wcześniej zdefiniowaną sekwencję S0.

```
CREATE SEQUENCE S0
START WITH 1
INCREMENT BY 1
NOMAXVALUE;

CREATE TRIGGER PRACOWNICY_S0
BEFORE INSERT ON PRACOWNICY
FOR EACH ROW
BEGIN
  SELECT S0.nextval
  INTO :NEW.id_p
  FROM DUAL;
END;
/
```

- Wyzwalacz zmieniający małe na wielkie litery wartości wstawianej lub modyfikowanej atrybutu PRACOWNICY.nazwisko

```
CREATE OR REPLACE TRIGGER PRACOWNICY_NAZWISKO
BEFORE INSERT OR UPDATE OF NAZWISKO ON PRACOWNICY
FOR EACH ROW
DECLARE
   w_nazwisko varchar2(30);
BEGIN
  w_nazwisko := upper(:new.nazwisko);
  :new.nazwisko := w_nazwisko;
END;
/
```

- Wyzwalacz archiwizujący w tabeli PRACOWNICY_ARCHIWUM oryginalny wiersz tabeli PRACOWNICY po jego modyfikacji komendą UPDATE włącznie z informację, kto i kiedy dokonał modyfikacji.

```
CREATE TABLE PRACOWNICY_ARCHIWUM
AS SELECT * from PRACOWNICY;

ALTER TABLE PRACOWNICY_ARCHIWUM
ADD (uzytkownik varchar2(40), data_archiwizacji date);

CREATE OR REPLACE TRIGGER PRACOWNICY_ARCHIWIZACJA
AFTER UPDATE ON PRACOWNICY
FOR EACH ROW
DECLARE
   operator varchar2(40);
   data date;
BEGIN
   select user into operator from dual;
   select sysdate into data from dual;
   INSERT INTO PRACOWNICY_ARCHIWUM
   (id_p,
    imie,
    nazwisko,
    pensja,
    id_bp,
    id_d,
    uzytkownik,
    data_archiwizacji)
    VALUES
   (:old.id_p,
    :old.imie,
    :old.nazwisko,
    :old.pensja,
    :old.id_bp,
    :old.id_d,
    operator,
    data);
END;
/
```

Sprawdźmy, jak zachowałby się ten wyzwalacz bez opcji FOR EACH ROW.
W tym celu wykonajmy update zmieniający pensję na 1000 wszystkich pracowników z działu 10. Sprawdźmy rezultaty wykonania wyzwalacza PRACOWNICY_ARCHIWIZACJA, a następnie zmieńmy jego definicję na przedstawioną poniżej w celu sprawdzenie, co stanie się, gdy wykonamy update zmieniający pensję na 5000 wszystkich pracowników z działu 10.

```
CREATE OR REPLACE TRIGGER PRACOWNICY_ARCHIWIZACJA
AFTER UPDATE ON PRACOWNICY
DECLARE
   operator varchar2(40);
   data date;
BEGIN
   select user into operator from dual;
   select sysdate into data from dual;
   INSERT INTO PRACOWNICY_ARCHIWUM
   (id_p,
    imie,
    nazwisko,
    pensja,
    id_bp,
    id_d,
    uzytkownik,
    data_archiwizacji)
    VALUES
   (:old.id_p,
    :old.imie,
    :old.nazwisko,
    :old.pensja,
    :old.id_bp,
    :old.id_d,
    operator,
    data);
END;
/
```

Ponieważ zdefiniowany powyżej wyzwalacz nie jest poprawny, zmodyfikujmy jego definicję w następujący sposób:

```
CREATE OR REPLACE TRIGGER PRACOWNICY_ARCHIWIZACJA
AFTER UPDATE ON PRACOWNICY
DECLARE
   operator varchar2(40);
   data date;
```

```
BEGIN
   select user into operator from dual;
   select sysdate into data from dual;
   INSERT INTO PRACOWNICY_ARCHIWUM
   (id_p,
    imie,
    nazwisko,
    pensja,
    id_bp,
    id_d,
    uzytkownik,
    data_archiwizacji)
    VALUES
   (id_p,
    imie,
    nazwisko,
    pensja,
    id_bp,
    id_d,
    operator,
    data);
END;
/
```

Kompilacja powyżej zdefiniowanego wyzwalacza kończy się z błędami, które nie zostały wyświetlone. W jaki sposób znaleźć opisy błędów kompilacji? Wystarczy zastosować komendę:

```
SHOW TRIGGER ERRORS;
```

Po analizie błędów kompilacji poprawiamy definicję wyzwalacza PRACOWNICY_ARCHIWIZACJA w przedstawiony poniżej sposób wyłącznie w celu sprawdzenie, czym skutkuje brak opcji FOR EACH ROW.

```
CREATE OR REPLACE TRIGGER PRACOWNICY_ARCHIWIZACJA
AFTER UPDATE ON PRACOWNICY
DECLARE
   operator varchar2(40);
   data date;
BEGIN
   select user into operator from dual;
   select sysdate into data from dual;
   INSERT INTO PRACOWNICY_ARCHIWUM
```

```
    (id_p,
     imie,
     nazwisko,
     pensja,
     id_bp,
     id_d,
     uzytkownik,
     data_archiwizacji)
     VALUES
    (0,
     null,
     null,
     null,
     null,
     null,
     operator,
     data);
END;
/
```

„Jednowierszowe” wyzwalacze DML, tzn. bez opcji FOR EACH ROW, są rzadko wykorzystywane.

Oprócz prostych używa się także złożonych wyzwalaczy DML (compound triggers).

II. 2 Wyzwalacze, które nie są związane z transakcjami DML

Wśród tych wyzwalaczy rozróżniamy wyzwalaczy związane z transakcjami DDL oraz różnymi innymi zdarzeniami w bazie danych

Tworzenie lub zmiana definicji wyzwalacza typy nie-DML

```
CREATE [OR REPLACE] TRIGGER trigger-name
{ BEFORE | AFTER } transakcja_zdarzenie [OR transakcja_zdarzenie]...
ON { [schemat_użytkownika.] SCHEMA | DATABASE }
[DECLARE ...]
BEGIN
--
[EXCEPTION ...]
END;
/
```

Przykład, wymagający ustawienia w SQL*Plus parametru SERVEROUTPUT:

```
SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER logowanie_firma
AFTER LOGON ON firma.SCHEMA
DECLARE
   operator varchar2(40);
   data date;
BEGIN
   select user into operator from dual;
   select sysdate into data from dual;
   dbms_output.enable;
   dbms_output.put_line('Witaj '||operator||' w dniu ' ||
to_char(data,'DD.MM.YYYY')|| ' o godzinie ' ||
to_char(data,'HH24:MI:SS'));
END;
/
```

Niestety, w zależności od wersji serwera baz danych oraz programu SQL*Plus i jego konfiguracji podane powyżej rozwiązanie nie daje oczekiwanego rezultatu. W takim przypadku należy dodać następujące komendy

```
SET SERVEROUTPUT ON;

EXEC NULL;
```

do jednego z plików konfiguracyjnych programu SQL*Plus, `glogin.sql`, dla którego ogólna ścieżka dostępu może zostać zapisana w następujący sposób:

ORACLE_HOME/sqlplus/admin/glogin.sql – w systemach linuksowych

ORACLE_HOME\sqlplus\admin\glogin.sql – w systemie Windows,

gdzie ORACLE_HOME oznacza ścieżkę dostępu do katalogu, w którym zostało zainstalowane oprogramowanie klienckie lub serwerowe bazy danych Oracle.

II. 3 Wyłączanie i wyłączanie wyzwalaczy

- Wyłączanie wyzwalacza

```
ALTER TRIGGER nazwa_wyzwalacza DISABLE;
```

- Włączanie wyzwalacz

```
ALTER TRIGGER nazwa_wyzwalacza ENABLE;
```

- Wyłączanie wszystkich wyzwalaczy dla danej tabeli

```
ALTER TABLE nazwa_tabeli DISABLE ALL TRIGGERS;
```

- Włączanie wyzwalacz

```
ALTER TABLE nazwa_tabeli ENABLE ALL TRIGGERS;
```

II. 4 Zmiana nazwy wyzwalacza

```
ALTER TRIGGER stara_nazwa_wyzwalacza RENAME TO
nowa_nazwa_wyzwalacza;
```

II. 5 Rekompilacja wyzwalacza

```
ALTER TRIGGER nazwa_wyzwalacza COMPILE;
```

II. 6 Usuwanie wyzwalacza

```
DROP TRIGGER nazwa_wyzwalacza;
```

II. 7 Związek wyzwalaczy z więzami, zapewniającymi integralność danych przechowywanych w bazach danych Oracle

Niekiedy należy zastosować wyzwalacze, aby zapewnić implementację zaprojektowanych związków między tabelami. To znaczy, aby zaprojektowane związki nie istniały tylko jako nasza intencja uwidoczniona w modelu logicznym, ale określona reguła integralności danych, faktycznie zapewniona przez serwer bazy danych. Przykładem takiej konstrukcji jest hierarchia encji oparta na jednym nadtypie z dwoma podtypami, przedstawiona w zrzutach ekranu zapisanych w pliku pracownicy_obywatelstwo.pdf, stanowiącym jeden z załączników do wykładu 6.

III. Indeksy

- Indeks jest dodatkową strukturą fizyczną w bazie danych, która umożliwia szybszy dostęp do danych.

- Indeksy są tworzone na atrybucie lub zbiorze atrybutów, które nazywa się wówczas atrybutami indeksowymi.

- Indeks jest uporządkowanym zbiorem rekordów (krotek) o stałej długości.

- Rekord (krotka) indeksu składa się z klucza jednoznacznie identyfikującego wiersz tabeli o danej podkrotce lub kilka wierszy tabeli o takiej samej podkrotce atrybutów indeksowych oraz ze wskaźnika do bloku danych krotki, w której wartości atrybutów indeksowych są jednoznaczenie reprezentowane przez klucz indeksu.

III. 1 Charakterystyka indeksów w bazach danych Oracle ze względu na więzy integralności (klucze główne, unikalne i obce)

- Indeksy unikalne

  Przykład komendy tworzącej indeks unikalny o nazwie własnej dept_indx na atrybucie mgr tabeli DEPT:

  CREATE UNIQUE INDEX DEPT_FKIDX ON DEPT (MGR);

- Indeksy nieunikalne

  CREATE INDEX EMP_FKIDX ON EMP (DEPTNO);

III. 2 Związek indeksów z więzami, zapewniającymi integralność danych przechowywanych w bazach danych Oracle oraz inne powody, dla których warto tworzyć indeksy

Indeksy są tworzone automatycznie dla kluczy głównych i unikalnych. W tym kontekście ciekawa jest analiza związku 1:1 między encjami EMP i DEPT, którą można przeprowadzić na podstawie zrzutów ekranu zapisanych w pliku scottBRSZD.pdf, stanowiącym jeden z załączników do wykładu 6.

Zastosowanie indeksów dla kluczy obcych jest obecnie opcjonalne w systemach baz danych Oracle, ale warto prześledzić analizę tego problemu, opublikowaną przez Burleson Consulting, skąd możemy poznać kilka istotnych wskazówek nt. indeksów i kluczy obcych.

- W bazach danych Oracle tworzenia indeksów dla kluczy obcych nie jest automatyczne.

- Chociaż w bazach danych Oracle mogą istnieć klucze obce bez indeksów, to obciążenie powodowane eksploracją danych za pomocą zapytań SQL określa, czy jest wymagany indeks dla klucza obcego. Brak indeksu dla klucza obcego będzie skutkował niepotrzebnym skanowaniem całej tabeli podrzędnej w poszukiwaniu wartości klucza obcego, które odpowiadają wymaganym przez zapytanie wartościom kluczom głównego lub unikalnego.

- Ogólna reguła, która określa, czy klucz obcy jest wymagany czy też nie, odwołuje się do tego, w jaki sposób tabela nadrzędna i podrzędna będą wykorzystywane w zapytaniach. Jeśli te tabele będą często uczestniczyć w złączeniach opartych na związku klucz obcy – jego klucz odniesienia, to należy zdefiniować indeks na takim kluczu obcym. Jeśli tego nie zrobimy, to złączenia oparte na tym związku klucz obcy – jego klucz odniesienia nie będą efektywne, ponieważ zwykle wiązać się będą z dodatkowo obciążającym sortowaniem, które będzie realizowane w celu uporządkowania danych przed złączeniem.

- Poza korzyściami, jakie dają indeksy na kluczach obcych w przypadku złączeń, indeksy na kluczach obcych są zalecane w przypadku różnych operacji DML:

  – Operacje DML w tabeli nadrzędnej, podczas których widoczna jest konieczność przeskanowania całej tabeli podrzędnej.

  – Operacje update/delete na kluczu odniesienia (głównym lub unikalnym) tabeli nadrzędnej.

Uwagi:

Jeśli klucz obcy został zdefiniowany z opcją „on delete cascade” i nie utworzono dla niego indeksu, to usunięcie wiersza tabeli nadrzędnej wiąże się z koniecznością pełnego przeskanowania tabeli podrzędnej w celu odszukania i znalezienia wierszy odpowiadających usuwanemu wierszowi nadrzędnemu. Podobnie skanowanie całej tabeli podrzędnej jest konieczne w przypadku klucza obcego utworzonego z opcją „on delete set null”.

Natomiast w przypadku klucza obcego utworzonego bez ww. opcji potrzebne jest pełnie skanowanie tabeli nadrzędnej, aby sprawdzić, czy istnieje w niej odpowiadający wiersz nadrzędny dla określonego wiersza w tabeli podrzędnej.

Rada:
Korzyści z utworzenia indeksów dla wszystkich kluczy obcych są większe niż ewentualne zmniejszenie wydajności serwera bazy związane z utrzymywaniem niepotrzebnych indeksów. Tę radę wzmacnia jeszcze kwestia blokad spowodowanych operacjami DML.

Warto przeanalizować zapytanie opublikowane Burleson Consulting, które znajduje klucze obce bez indeksów.

- Blokady tabelowe związane z operacjami DML a indeksy na kluczach obcych

Rozpatrzmy znane tabele EMP i DEPT z kluczem obcym w tabeli EMP narzuconym na atrybut DEPTNO, odnoszącym się do klucza głównego w tabeli DEPT.

Przykładowo w przypadku operacji usunięcie wierszy z tabeli DEPT z wartościami DEPTNO, które znajdują się w kluczu obcym w tabeli EMP w celu zachowania integralności danych serwer bazy danych musi zastosować pełną blokadę tabeli EMP, jeśli nie został utworzony indeks na kluczu obcym w tabeli EMP.

Gdyby został utworzony indeks na kluczu obcym w tabeli EMP, to podczas wspomnianej operacji wystarczyłaby blokada odpowiednich wierszy w tabeli podrzędnej, w których zostałyby znalezione wartości klucza obcego, odpowiadające wartości klucza głównego usuwanego wiersza z tabeli nadrzędnej DEPT.

Brak indeksów na kluczach obcych może spowodować nadmierną rywalizację o blokadę w tabeli podrzędnej podczas dużej aktywności DML w tabeli nadrzędnej.

III. 3 Indeksy złożone (kompozytowe) w bazach danych Oracle

Indeksy kompozytowe, zwane także połączeniowymi (konkatenowanymi) są indeksami utworzonymi na kilku kolumnach tabeli.

W dyskusji indeksów kompozytowych abstrahujemy od narzucania indeksów na klucze główne, unikalne i obce.

Kolejność atrybutów w definicji indeksów kompozytowych powinna być taka, aby miała ona największy sens dla zapytań. Atrybutu zastosowane w definicji indeksu kompozytowego nie muszą być atrybutami przylegającymi w definicji tabeli.

Indeksy kompozytowe mogą przyspieszać pozyskiwanie danych przez komendy SELECT, w których klauzula WHERE odnosi się do wszystkich lub istotnej części atrybutów, objętych indeksem kompozytowym. Dlatego kolejność atrybutów w definicji indeksów kompozytowych jest istotna.

Przykładowo, jeśli przypuszczamy, że atrybuty ENAME, JOB i SAL w tabeli EMP, będą często wykorzystywane w zapytaniach właśnie w takiej kolejności, to można utworzyć następujący indeks kompozytowy:

```
CREATE INDEX EMP_IDX0 ON EMP (ENAME, JOB, SAL);
```

Uwaga:

- Zapytania, które wykorzystują wszystkie indeksowane kolumny, a także zapytanie, które wykorzystują jedynie kolumnę ENAME albo kolumny ENAME i JOB są w stanie skorzystać z indeksu EMP_IDX0.

- Zapytania, które nie wykorzystują kolumny ENAME, nie skorzystają z indeksu EMP_IDX0.

Interesująca możliwość:

Indeksy kompozytowe mogą być oparte na tych samych atrybutach, jeśli ich permutacje zastosowane w definicjach indeksów są różne.

Przykładowo:

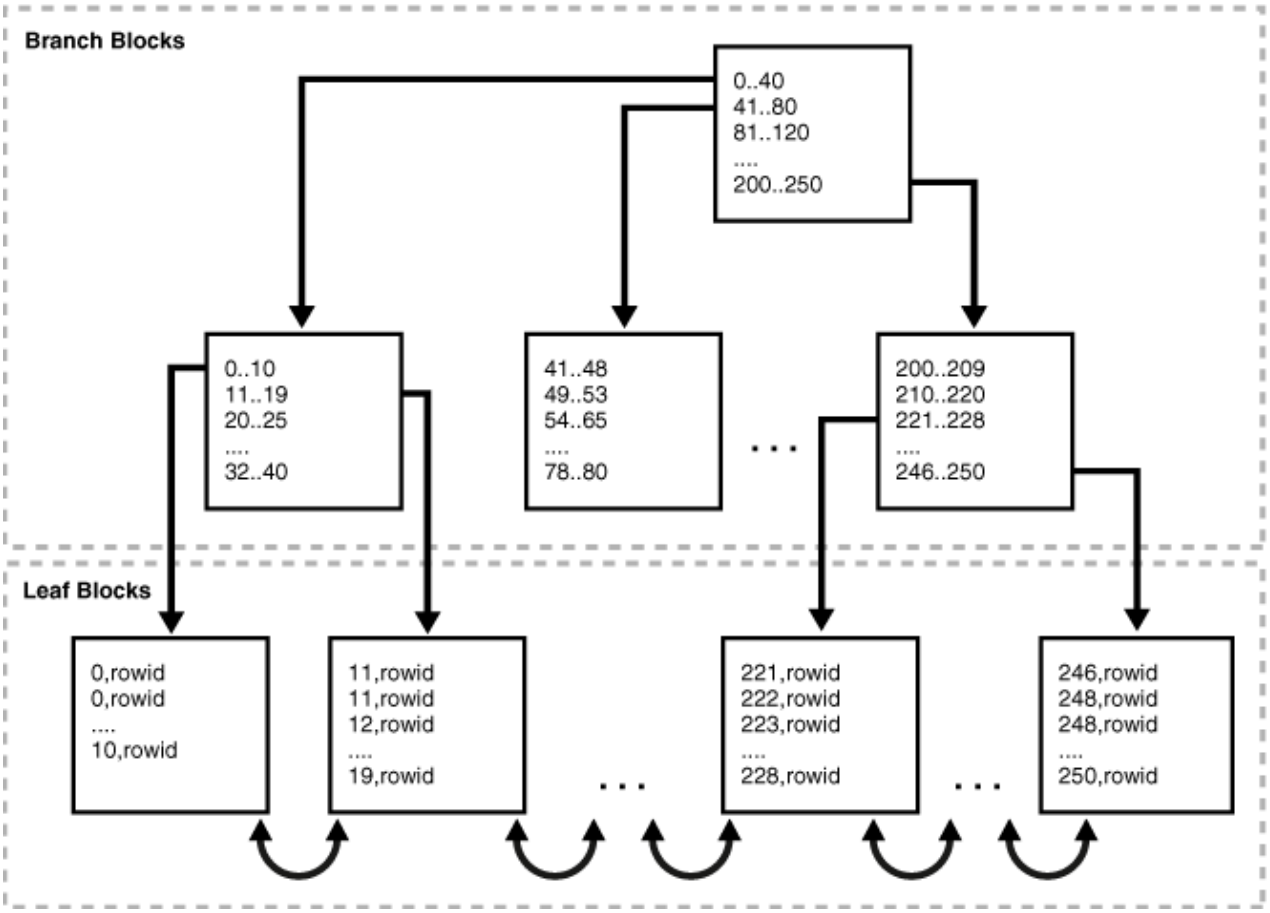
CREATE INDEX EMP_IDX1 ON EMP (ENAME, JOB);

CREATE INDEX EMP_IDX2 ON EMP (JOB, ENAME);

III. 4 Ogólne typy indeksów w bazach danych Oracle

- Indeksy w postaci B-drzew
  (B nie ma jednoznacznego odniesienia: Bayer (jeden z autorów), Boeing (firma, dla której pracował), zrównoważone (balanced), szerokie (broad), krzaczaste (bushy), ale nie mylić z drzewami binarnymi)

  Ten typ indeksów jest standardowym typem indeksów w bazach danych Oracle. Doskonale nadaje się dla kluczy głównych i unikalnych. Jeśli zastosujemy ten typ indeksu do indeksów kompozytowych, to będziemy mieli możliwość pozyskiwania danych posortowanych względem indeksowanych kolumn.



Rys. 3-1 zaczerpnięty z rozdziału „Indeksy i tabele zorganizowane wg indeksów" z instrukcji do Oracle Database 11.2.

Indeks w postaci B-drzewa posiada 2 rodzaje bloków:

- Bloki gałęzi (branch blocks), z których najwyższy poziom nazywa się korzeniem (root branch block). Bloki służą poszukiwaniu danych. Gałęzie zawierają dane, które wskazują niższego poziomu bloki indeksu. Przykładowo blok korzenia wskazuje niższego poziomu gałęzie, które następnie wskazują blok liścia, który zawiera wartości klucza indeksu reprezentowane w bloku gałęzi wskazującym ten blok liścia.

- Bloki liści (leaf blocks) przechowują w ramach pojedynczego wpisu indeksowaną wartości danych (tj. wartość klucza indeksu) wskazywaną przez blok gałęzi oraz odpowiadający jej nr wiersza tabeli podstawowej, dzięki czemu można dostać się poprzez zapisy w indeksie do odpowiedniego wiersza/wierzy danych tabeli podstawowej. Pojedynczy wpis do bloku liścia indeksu typu B-drzewo zawiera wartości krotki (key,rowid), a blok liścia jest posortowany według krotki (key,rowid). Blok liścia jest powiązany ze swoim prawym i lewym sąsiadem z zachowaniem wspomnianego sortowania według krotki (key,rowid).

Indeks w postaci B-drzewa jest zrównoważony, ponieważ wszystkie bloki liści znajdują się na tej samej głębokości. Z tego powodu, wydobycie dowolnego rekordu (krotki) z dowolnej pozycji w indeksie trwa w przybliżeniu tak samo.

Wysokość indeksu definiuje się przez liczbę bloków koniecznych do osiągnięcia bloku liścia z poziomu bloku korzenia. Przykładowo, indeks przedstawiony na Rys. 3-1 z instrukcji do Oracle Database 11.2 ma wysokość 3, gdzie 3 to poziom bloków liści.

Uwaga:

Indeksy na kolumnach typu tekstowego są oparte na binarnych wartościach znaków w określonym zestawie znaków bazy danych.

Rozróżnia się różne podtypy indeksów w postaci B-drzew:

– Indeksy, na których oparte są tabele zorganizowane według indeksów

  W tabelach zorganizowanych według indeksów dane są składowane zgodnie ze strukturą indeksu typu B-drzewa w przeciwieństwie do standardowych tabel, w których dane są składowane w postaci sterty (heap).

– Indeksy o odwróconym kluczu

  Na przykład składowanie klucza 701 zamiast 107. Umożliwia to rozłożenie wstawiania danych do indeksy na wiele bloków, co jest korzystne przy wielu takich samych operacjach wstawiania, jak ma to miejsce w Real Application Cluster.

– Indeksy klastrowe typu B-drzew

  Jest stosowany do indeksowania kluczy tabel klastrowych, gdzie przez tabelę klastrową należy rozumieć grupę tabel, które współdzielą te same bloki danych. W tym przypadku zamiast wskazywania wiersza, klucz indeksu wskazuje blok, który zawiera wiersze związane z kluczem klastrowym.

– Indeksy malejące i rosnące

Spośród tych ostatnich domyślne są indeksy rosnące (domyślny parametr ASC przy atrybucie indeksu w definicji indeksu). Natomiast indeksy malejące definiuje się za pomocą parametru DESC przy atrybucie indeksu w definicji indeksu. Wówczas indeks składuje wartości atrybutu w porządku malejącym. Możliwe są indeksy mieszane rosnąco-malejące w zależności od atrybutu.

Przykładowo:

```
CREATE INDEX EMP_IDX3 ON EMP (ENAME asc, DEPTNO desc);

CREATE INDEX EMP_IDX3 ON EMP (ENAME, DEPTNO desc);
```

- Indeksy, wykorzystujące mapy bitowe i mapy bitowe dla złączeń

  Te indeksy używają map bitowych do wskazywania wielu wierszy (krotek) w przeciwieństwie do indeksów typu B-drzewo, które wskazują pojedyncze wiersze (krotki). Indeks bitmapowy dla złączeń jest indeksem bitmapowym dla złączenia dwóch lub więcej tabel.

  Przykład:

```
CREATE BITMAP INDEX EMP_BM_IDX ON EMP (DEPT.DEPTNO)
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

- Indeksy oparte na funkcji

  Te indeksy dotyczą atrybutów używanych w wyrażeniach lub będących argumentami niektórych funkcji (np. upper).
  Zarówno indeksy typu B-drzew, jak i indeksy bitmapowe mogą być oparte na funkcji.

  Przykłada:

```
CREATE INDEX EMP_SMALL_BRUTTO_IDX
ON EMP (sal*100/119.64, sal);
```

- Indeksy w domenie aplikacji

  Te indeksy mogą być tworzone na potrzeby aplikacji opartych na bazie danych. Nie muszą mieć one standardowej struktury i mogą być składowane zarówno w tabelach baz danych Oracle, jak i w zewnętrznych plikach.

III. 6 Usunięcie indeksu

DROP INDEX nazwa_indeksu;

Uwagi:
W komendzie DROP INDEX … nie precyzujemy typu indeksu.
Nazwa indeksu w schemacie danego użytkownika musi być unikalna.

III. 7 Zmiana definicji indeksu

Nie istnieje komenda CREATE OR REPLACE INDEX …

Dlatego jednym ze sposobów jest usunięcie indeksu (DROP INDEX …) i utworzenie go (CREATE INDEX …) ze zmienioną definicją.

Jednak często wystarczy zmienić indeks własności indeksu za pomocą komendy ALTER INDEX …

III. 8 Wybrane możliwości komendy ALTER INDEX …

- Uczynienie indeksu nieużywalnym

ALTER INDEX nazwa_indeksu UNUSABLE;

Wówczas indeks ignorowany przez optymalizator zapytań oraz nie jest uwzględniany przez operacje DML.

Aby ponownie móc używać taki indeks, należy go przebudować:

ALTER INDEX nazwa_indeksu REBUILD;

Można też zastosować wygodniejszą opcję przebudowywania indeksu

ALTER INDEX nazwa_indeksu REBUILD ONLINE;

Ta druga opcja (REBUILD ONLINE) umożliwia aktualizowanie tabeli podczas przebudowywania indeksu dotyczącego tej tabeli.

Alternatywnie można indeks usunąć, a następnie ponownie utworzyć go.

- Uczynienie indeksu niewidzialnym

ALTER INDEX nazwa_indeksu INVISIBLE;

Indeks niewidzialny jest uwzględniany przez operacje DML w przeciwieństwie do indeksu nieużywalnego. Natomiast indeks niewidzialny jest ignorowany przez optymalizator zapytań, chyba że parametr serwera bazy danych OPTIMIZER_USE_INVISIBLE_INDEXES ma wartość TRUE.

Indeks niewidzialny łatwo uczynić ponownie widzialny za pomocą komendy:

ALTER INDEX nazwa_indeksu VISIBLE;

- Wyłączanie indeksów opartych na funkcji:

ALTER INDEX nazwa_indeksu_opartego_na_funkcji DISABLE;

- Włączanie indeksów opartych na funkcji:

ALTER INDEX nazwa_indeksu_opartego_na_funkcji ENABLE;

- Przebudowa indeksu do indeksu o odwróconym kluczu:

ALTER INDEX nazwa_indeksu REBUILD REVERSE;

- Przebudowa indeksu do indeksu o nieodwróconym kluczu:

ALTER INDEX nazwa_indeksu REBUILD NOREVERSE;

- Ponowna kompilacja indeksu

ALTER INDEX nazwa_indeksu COMPILE;

Ta operacje daje możliwość naprawy indeksu bez konieczności przebudowywania indeksu, który w wyniku zmiany swojej przestrzeni zarządzania, np. z zarządzania w domenie użytkownika do zarządzania w domenie systemowej, otrzymał status INVALID.