



POZNAN UNIVERSITY OF TECHNOLOGY

Partycjonowanie

Robert Wrembel

Politechnika Poznańska

Instytut Informatyki

Robert.Wrembel@cs.put.poznan.pl

www.cs.put.poznan.pl/rwrembel



Partycjonowanie tabel (1)

➤ Podział tabeli na mniejsze fragmenty

- **operacje dostępu** do dysków mogą być wykonywane **równolegle**;
- jest równoważone **obciążenie dysków**;
- **polecenia SQL** adresujące różne partycje mogą być wykonywane **równolegle**;
- polecenia SQL mogą **adresować konkretną partycję** eliminując w ten sposób konieczność przeszukiwania całej tabeli;
- wzrasta **bezpieczeństwo** danych w przypadku awarii sprzętu - awaria np. jednego dysku uniemożliwia dostęp tylko do partycji na tym dysku, natomiast partycje znajdujące się na nieuszkodzonych dyskach są nadal dostępne;
- wzrasta **szybkość odtwarzania danych po awarii** - odtwarzaniu podlegają tylko uszkodzone partycje, a nie cała tabela

Partycjonowanie tabel (2)

- Wybór partycji do której trafi rekord jest realizowany na podstawie wartości jednego lub kilku wybranych atrybutów tabeli – tzw. atrybutów partycjonujących

<u>A1</u>	A2	A3	A4	A5	A6

<u>A1</u>	A2	A3	A4	A5	A6

horizontal

vertical

mixed

<u>A1</u>	A2	A3

<u>A1</u>	A4	A5	A6

<u>A1</u>	A2	A3

<u>A1</u>	A2	A3

Kryteria poprawności partycjonowania (1)

⇒ Kompletność (ang. **completeness**)

- jeżeli tabela **T** została podzielona na partycje **TP₁**, **TP₂**, ..., **TP_n**, to każdy rekord z **T** lub jego fragment musi się znaleźć w jednej z partycji **TP₁**, **TP₂**, ..., lub **TP_n**
- kryterium to gwarantuje, że na skutek partycjonowania żadne dane z tabeli pierwotnej **T** nie zostaną utracone

Kryteria poprawności partycjonowania (2)

⇒ Rekonstrukcja (ang. **reconstruction**)

- musi istnieć możliwość zrekonstruowania pierwotnej tabeli **T** ze wszystkich jej partycji
- rekonstrukcja nie może doprowadzić ani do utraty danych, ani do powstania danych nadmiarowych

⇒ Rozłączność (ang. **disjointness**)

- jeżeli tabela **T** została podzielona na partycje **TP₁**, **TP₂**, ..., **TP_n**, to każdy rekord z **T** lub jego fragment nie może się znaleźć w więcej niż jednej partycji
- kryterium to gwarantuje, że na skutek partycjonowania w bazie danych nie pojawią się dane nadmiarowe
- wyjątkiem od tej reguły jest partycjonowanie pionowe, w którym wartości atrybutów stanowiących klucz podstawowy tabeli występują w każdej partycji

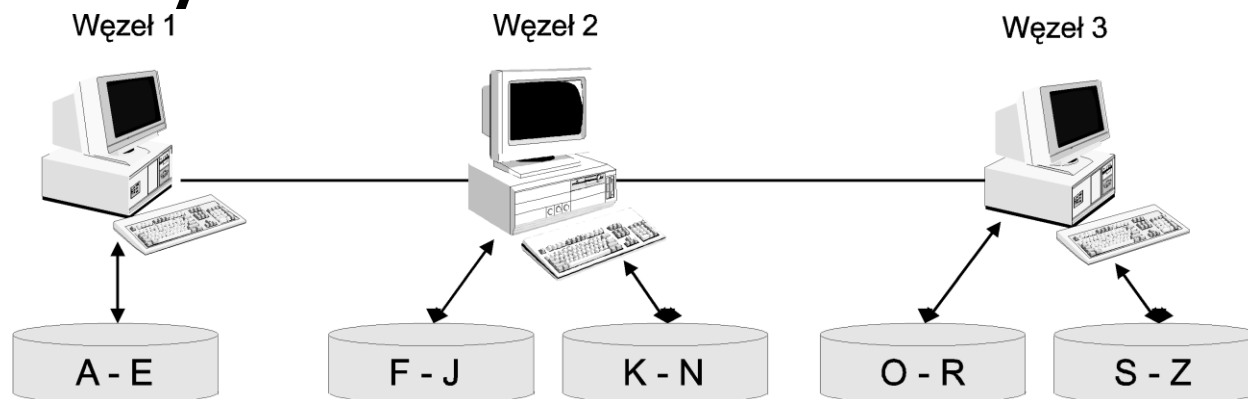
Algorytmy partycjonowania (1)

⇒ Round-Robin

- umożliwia równomierne rozpraszanie danych w węzłach sieci
- dane są rozpraszane w sposób przypadkowy, więc odnalezienie żądanych informacji wymaga przeszukania wszystkich węzłów

⇒ Bazujący na wartości

- rozmieszczenie danych w sieci zależy od wartości samych danych



Algorytmy partycjonowania (2)

⇒ Haszowy

- dane są umieszczane w węzłach zgodnie z wartością systemowej funkcji haszowej
- argumentem wejściowym tej funkcji jest wartość atrybutu, a jej wynikiem — adres węzła, w którym zostanie umieszczony rekord
- w celu odnalezienia żądanych informacji SZBD wykorzystuje tę samą funkcję haszową, która została wykorzystana do rozproszenia danych
- zaletą tej metody jest możliwość automatycznego umieszczania w tym samym węźle rekordów pochodzących z różnych, powiązanych z sobą tabel

Algorytmy partycjonowania (3)

⇒ Hybrydowy

- umożliwia dwustopniowe rozpraszanie danych
- w kroku pierwszym, dane są umieszczane w poszczególnych węzłach za pomocą haszowania
- w kroku drugim, dane są umieszczane na poszczególnych dyskach danego węzła, za pomocą techniki partycjonowania bazującego na wartości

Oracle – partycjonowanie zakresowe (1)

- Dla każdej partycji określa się zakres wartości atrybutu partycjonującego
 - do danej partycji trafiają więc rekordy ze ściśle określonego dla tej partycji zakresu

```
create table klienci1
(klient_id number(10), imie varchar2(25),
nazwisko varchar2(25), kod_miasta varchar2(6))
partition by range(kod_miasta)
(partition p_klienci_C values less than ('D') tablespace dane1,
partition p_klienci_F values less than ('G') tablespace dane2,
partition p_klienci_M values less than ('N') tablespace dane3);
```

- Rekord jest wstawiany do odpowiedniej partycji na podstawie wartości atrybutu partycjonującego
- Jeżeli wartość tego atrybutu dla wstawianego rekordu nie pasuje do zakresu żadnej partycji, wówczas system zgłasza błąd

Oracle – partycjonowanie zakresowe (2)

➔ Nieograniczony rozmiar ostatniej partycji

```
create table klienci2
(klient_id number(10),
 imie varchar2(25),
 nazwisko varchar2(25),
 kod_miasta varchar2(6))
partition by range(kod_miasta)
(partition p_klienci_C values less than ('D') tablespace dane1,
 partition p_klienci_F values less than ('G') tablespace dane2,
 partition p_klienci_M values less than ('N') tablespace dane3,
 partition p_klienci_MAX values less than (maxvalue)
 tablespace dane4);
```

- ➔ Słowo kluczowe **maxvalue** w ostatniej klauzuli **partition** wskazuje, że maksymalny zakres tej partycji jest nieograniczony. Tak zdefiniowana partycja będzie przechowywała również rekordy z pustymi wartościami atrybutów partycjonujących, tj. **kod_miasta** w naszym przykładzie



Oracle – partycjonowanie listowe (1)

- ➔ Dla każdej partycji określa się zbiór wartości atrybutu partycjonującego za pomocą **klauzuli partition by list ()**

```
create table bilety1
(nr_biletu varchar2(15),
cena number(6,2),
klient_id number(10),
klasa varchar2(12))
partition by list (klasa)
(partition p_ekonomiczna values ('ekonomiczna')
tablespace dane1,
partition p_wyzsze values ('business', 'pierwsza')
tablespace dane2);
```

Oracle – partycjonowanie listowe (2)

- Jeżeli wartość atrybutu partycjonującego dla wstawianego rekordu nie pasuje do wartości żadnej partycji, wówczas system zgłasza błąd
- Definicję tabeli partycjonowanej można rozszerzyć o partycję umożliwiającą przechowywanie wszystkich innych wartości
 - do definiowania takiej partycji wykorzystuje się słowo kluczowe **default**

```
create table bilety2
(nr_biletu varchar2(15), cena number(6,2),
 klient_id number(10), klasa varchar2(12))
partition by list (klasa)
(partition p_ekonomiczna values ('ekonomiczna')
 tablespace dane1,
 partition p_business values ('business', 'pierwsza')
 tablespace dane2,
 partition p_inne values (default) tablespace dane3);
```



Oracle – partycjonowanie haszowe (1)

- ⇒ Tabela **klienci_part_hash1** jest dzielona na pięć partycji
- każda z nich jest umieszczona w domyślnej przestrzeni tabel użytkownika
 - nazwy partycji są nadawane przez system według formatu **SYS_Pnnn**

```
create table klienci_part_hash1
(klient_id number(10),
 imie varchar2(25),
 nazwisko varchar2(25),
 kod_miasta varchar2(6))
partition by hash(kod_miasta)
partitions 5;
```

Oracle – partycjonowanie haszowe (2)

- ➔ Tabela **klienci_part_hash2** zostanie podzielona na pięć partycji umieszczonych w jawnie wyspecyfikowanych przestrzeniach tabel **dane1**, **dane2** i **dane3**
- liczba partycji jest większa niż liczba przestrzeni tabel, więc partycje są umieszczane w kolejnych przestrzeniach tabel za pomocą algorytmu round-robin
 - nazwy partycji są nadawane przez system

```
create table klienci_part_hash2
(klient_id number(10),
 imie varchar2(25),
 nazwisko varchar2(25),
 kod_miasta varchar2(6))
partition by hash(kod_miasta)
partitions 5
store in (dane1, dane2, dane3);
```

Oracle – partycjonowanie haszowe (3)

- ➔ Tabela **klienci_part_hash3** zostanie podzielona na trzy partycje umieszczone kolejno w przestrzeniach tabel **dane1**, **dane2** i **dane3**

```
create table klienci_part_hash3
(klient_id number(10),
 imie varchar2(25),
 nazwisko varchar2(25),
 kod_miasta varchar2(6))
partition by hash(kod_miasta)
(partition p_klienci_1 tablespace dane1,
 partition p_klienci_2 tablespace dane2,
 partition p_klienci_3 tablespace dane3);
```



Oracle – partycjonowanie hybrydowe (1)

- Partycje uzyskane za pomocą partycjonowania zakresowego mogą być dalej dzielone albo za pomocą partycjonowania haszowego albo listowego
- tabela **sprzedaz1**, jest najpierw dzielona na trzy główne partycje zakresowe zgodnie z wartością atrybutu **rok** (klauzula **partition by range (rok)**)
 - partycje te mają nazwy **p_1990**, **p_2000**, **p_2010** i są umieszczane odpowiednio w przestrzeniach tabel **dane1**, **dane2**, **dane3**
 - następnie, każda z tych partycji jest dzielona na dwie podpartycje haszowe, zgodnie z wartością atrybutu **sklep_id** (klauzule **subpartition by hash (sklep_id) subpartitions 2**)

```
create table sprzedaz1
(id_sprzedazy number(10), kwota number (6,2), klient_id number(10),
sklep_id number(10), rok number(4))
partition by range (rok)
subpartition by hash (sklep_id)
subpartitions 2
(partition p_1990 values less than (1991) tablespace dane1,
partition p_2000 values less than (2001) tablespace dane2,
partition p_2010 values less than (2011) tablespace dane3);
```



Oracle – partycjonowanie hybrydowe (2)

➤ Każda z partycji głównych może być niezależnie dzielona na podpartycje

- tabela **sprzedaz2** jest dzielona na trzy główne partycje zakresowe
- następnie każda z tych partycji jest niezależnie dzielona na podpartycje haszowe: **p_1990** i **p_2000** — na 2 podpartycje (klauzule **subpartitions 2**), a **p_2010** — na cztery podpartycje o jawnie podanych nazwach **sp_1**, **sp_2**, **sp_3**, **sp_4**

```
create table sprzedaz2
(id_sprzedazy number(10), kwota number (6,2), klient_id number(10),
sklep_id number(10), rok number(4))
partition by range (rok)
subpartition by hash (sklep_id)
(partition p_1990 values less than (1991) tablespace dane1
subpartitions 2,
partition p_2000 values less than (2001) tablespace dane2
subpartitions 2,
partition p_2010 values less than (2011) tablespace dane3
(subpartition sp_1, subpartition sp_2,
subpartition sp_3, subpartition sp_4));
```



Virtual column partitioning

⇒ Od Oracle11g

⇒ Kolumna wirtualna ⇒ wyliczana

```
create table Products_Virt1
(...
  SellPrice number(6,2) ,
  Tax number(4,2) ,
  Gross as (SellPrice*Tax))
PARTITION by range(Gross)
(partition Prod1 values less than (1000) ,
 partition Prod2 values less than (2000));
```

```
select *
from Products_Virt1
where SellPrice*Tax<1000;
```

System partitioning

- ➔ **Od Oracle11g**
- ➔ **Brak ograniczeń zbioru wartości w partycji ➔ dowolny rekord może się znaleźć w dowolnej partycji**
- ➔ **Rozmieszczaniem rekordów w partycjach steruje aplikacja ➔ Insert jawnie wskazuje partycje**

```
create table Customers_Sys
(CustomerID varchar2(10)
        CONSTRAINT pk_Customers PRIMARY KEY,
...)
PARTITION by SYSTEM
(partition Cust_Europe,
 partition Cust_America,
 partition Cust_Asia);
```

Reference partitioning

⇒ **Od Oracle11g**

⇒ **Stosowane do partycjonowania tabeli z kluczem obcym wg schematu partycjonowania tabeli z kluczem podstawowym, np. Sprzedaż → Klienci**

```
create table Customer_List_Country
(CustomerID varchar2(10) PRIMARY KEY,
Town varchar2(20),
Postal_code varchar2(10),
Country varchar2(20)
)
PARTITION by LIST(Country)
(partition part_UK values ('UK'),
partition part_USA values ('USA'));
```

must be NOT NULL and non deferrable
constraint

```
create table Sales_List_Cust
(ProductID varchar2(8),
CustomerID varchar2(10) not null constraint CustID_FK
references Customer_List_Country(CustomerID),
...)
PARTITION by REFERENCE (CustID_FK);
```

Reference partitioning

- **Uwaga: poprzednia definicja klucza obcego może powodować błąd:**
 - **ORA-14655: reference partitioning constraint not found**
- **Rozwiązanie: zdefiniować FK na poziomie tabeli**

```
create table Sales_List_Cust
(ProductID varchar2(8) ,
 CustomerID varchar2(10) not null,
 ...,
 constraint CustID_FK foreign key (CustomerID)
      references Customer_List_Country(CustomerID) )
PARTITION by REFERENCE (CustID_FK) ;
```

- **Uwaga: pominięcie NOT NULL dla CustomerID kończy się komunikatem**
 - **ORA-14652: reference partitioning foreign key is not supported**

Dynamic partitioning

- Partycje tworzone automatycznie, gdy są potrzebne
- Dla sprzedaży w 2014, dla każdego miesiąca zostanie utworzona osobna partycja

```
create table Sales_M
(ProductID varchar2(8),
 CustomerID varchar2(10) not null,
 ...,
 s_date date default sysdate)
PARTITION BY RANGE (s_date)
    INTERVAL(NumToYMInterval(1, 'MONTH'))
(PARTITION part_01 values
    LESS THAN (TO_DATE('01-01-2014', 'dd-mm-yyyy'))
);
```

partycja początkowa

```
select * from Sales_M
    partition for (to_date('29-11-2014', 'dd-mm-yyyy'));
```

Dynamic partitioning

```
create table sales_test
(sID number(6),
 custID number (6),
 amount number (5))
partition by range(amount)
    INTERVAL(1000)
(partition p1 values less than (1000),
 partition p2 values less than (2000)
);
```

```
select table_name,
partition_name
from user_tab_partitions
where table_name='SALES_TEST';
```

TABLE_NAME	PARTITION_NAME
SALES_TEST	P1
SALES_TEST	P2

```
insert into sales_test
values (10, 501, 2500);
```

```
select table_name,
partition_name
from user_tab_partitions
where table_name='SALES_TEST';
```

TABLE_NAME	PARTITION_NAME
SALES_TEST	P1
SALES_TEST	P2
SALES_TEST	SYS_P301



Dynamic partitioning

- ➔ **Tylko jeden atrybut partycjonujący**
- ➔ **Musi być typu NUMBER lub DATE**



DML na tabelach partycjonowanych

➔ Adresowanie konkretnej partycji w poleceniach SELECT i DELETE

```
select * from klienci partition (p_klienci_c) ;
```

```
delete from klienci partition (p_klienci_MAX) ;
```

DML na tabelach partycjonowanych

⇒ Modyfikacje wartości atrybutu

▪ starsze wersje Oracle

- mogą być wykonywane jedynie wtedy, gdy nie powodują przeniesienia rekordu do innej partycji, tj. nowa wartość modyfikowanego atrybutu znajduje się w tym samym zakresie partycji, co wartość sprzed modyfikacji
- w przeciwnym przypadku, system zgłasza błąd
- jedynym sposobem przeniesienia rekordu do innej partycji jest jego usunięcie, a następnie wstawienie, ale z nową wartością atrybutu partycjonującego

▪ nowsze wersje Oracle

- rekordy mogą być automatycznie przenoszone pomiędzy partycjami pod warunkiem wykonania polecenia:
- **ALTER TABLE ... ENABLE ROW MOVEMENT;**

Zarządzanie tabelami partycjonowanymi (1)

- Dodanie nowej partycji do tabeli (**alter table add partition**)
- Podział partycji na dwie (**alter table split partition**)
- Przeniesienie partycji do innej przestrzeni tabel (**alter table move partition**)
- Zmiana nazwy partycji (**alter table rename partition**)
- Scalenie zawartości dwóch partycji (**alter table merge partition**)
- Wymiana danych partycji z danymi wskazanej tabeli (**alter table exchange partition**)
- Zmodyfikowanie parametrów partycji (**alter table modify partition**), m.in. parametry składowania
- Usunięcie danych z partycji (**alter table truncate partition**)
- Usunięcie partycji z tabeli, usuwa również niepustą partycję (**alter table drop partition**)

Zarządzanie tabelami partycjonowanymi (2)

➔ Dodanie nowej partycji

```
alter table klienci1
add partition p_klienci_MAX values less than
(maxvalue)
    tablespace dane4
        storage (initial 128K
                next 128K
                pctincrease 0
                minextents 1);
```

partycja zakresowa

partycja listowa

```
alter table bilety1
add partition p_inne values ('1', '2', '3')
tablespace dane1;
```

partycja haszowa

```
alter table
klienci_part_hash3
add partition p_klienci_4
tablespace dane3;
```

Zarządzanie tabelami partycjonowanymi (3)

- Do tabeli z wyspecyfikowanym maksymalnym zakresem partycji za pomocą **maxvalue** - dla partycji zakresowych lub **default** - dla partycji listowych nie można dodawać nowych partycji
 - jedynym sposobem zwiększenia liczby partycji jest w takim przypadku podział ostatniej partycji na dwie
 - podział partycji jest możliwy jedynie dla partycji zakresowych i listowych
- Partycja **p_klienci_MAX** jest dzielona na dwie nowe, o nazwach **p_klienci_Q** i **p_klienci_MAX1**
 - zakres wartości dla **p_klienci_Q** to ciągi znaków od **N** do **Q**
 - zakres wartości dla **p_klienci_MAX1** rozpoczyna się od litery **R**

```
alter table klienci1
split partition p_klienci_MAX
at ('R')
into (partition p_klienci_Q tablespace dane4,
      partition p_klienci_R_MAX tablespace dane5);
```



Zarządzanie tabelami partycjonowanymi (4)

➤ Podział partycji listowej **p_inne**, w tabeli **bilety1**

- **values** umożliwia wskazanie zbioru wartości dla pierwszej partycji wymienionej w klauzuli **into**, czyli **p_inne1**

```
alter table bilety1
split partition p_inne
values ('1', '2')
into (partition p_inne1_2,
      partition p_inne3);
```

➤ Poniższe polecenie przenosi partycję **p_inne2** do przestrzeni tabel **dane3**

```
alter table bilety1
move partition p_inne2 tablespace dane3;
```

Zarządzanie tabelami partycjonowanymi (5)

➔ Zmiana nazwy partycji

```
alter table bilety1 rename partition p_inne2  
to p_klasa3;
```

➔ Scalenie dwóch partycji w jedną

- możliwe jedynie dla partycji zakresowych lub listowych
- poniższe polecenie scala zawartość partycji **p_inne1** i **p_klasa3**, a rezultat scalenia zostanie zapisany w nowej partycji **p_inne**
- partycje źródłowe po scaleniu są automatycznie usuwane

```
alter table bilety1  
merge partitions p_inne1, p_klasa3  
into partition p_inne;
```

Zarządzanie tabelami partycjonowanymi (6)

⇒ Wymiana danych pomiędzy partycją, a tabelą

- zawartość partycji **p_ekonomiczna** zostanie zamieniona z zawartością tabeli **rezerwacje**
- klauzula **without validation** powoduje wymianę danych nawet jeśli nie spełniają one kryteriów zakresu wartości dla partycji
- domyślną klauzulą jest **with validation**
 - dane nie zostaną wymienione jeśli choć jeden rekord naruszałby kryteria zakresu wartości dla partycji

```
alter table bilety1
exchange partition p_ekonomiczna
with table rezerwacje
without validation;
```

⇒ Wymiana danych partycja → tabela możliwa jeżeli:

- oba obiekty mają identyczną liczbę atrybutów oraz identyczne typy i długości odpowiadających sobie atrybutów
- nazwy odpowiadających sobie atrybutów mogą być różne

Zarządzanie tabelami partycjonowanymi (7)

➔ Usunięcie danych z partycji

```
alter table bilety1  
truncate partition  
p_ekonomiczna;
```

```
alter table bilety1  
drop partition  
p_klasa3;
```

➔ Usunięcie partycji

- możliwe tylko dla partycji zakresowych i listowych
- partycji haszowych nie można usunąć
- dla partycji haszowych należy zastosować COALESCE PARTITION
- COALESCE PARTITION - przenosi rekordy z wybranej przez system partycji do pozostałych partycji i usuwa tę partycję

```
alter table klienci_part_hash3 coalesce partition;
```

Informacje słownikowe

➤ Tabele partycjonowane – USER_PART_TABLES

- TABLE_NAME
- PARTITIONING_TYPE (HASH, LIST, RANGE)
- PARTITION_COUNT (liczba partycji)
- DEF_TABLESPACE_NAME

➤ Atrybuty partycjonujące – USER_PART_KEY_COLUMNS

- COLUMN_NAME
- COLUMN_POSITION

➤ Partycje tabeli – USER_TAB_PARTITIONS

- TABLE_NAME
- COMPOSITE (NO, YES – part. hybrydowe)
- PARTITION_NAME
- HIGH_VALUE
- TABLESPACE_NAME

➤ Podpartycje – USER_TAB_SUBPARTITIONS, USER_SUBPART_KEY_COLUMNS

Partycjonowanie indeksów (1)

⇒ Cel

- zwiększenie stopnia współbieżności transakcji
- minimalizacja rywalizacji transakcji, poprzez rozproszenie operacji wejścia/wyjścia wykonywanych na indeksie

⇒ Oracle umożliwia tworzenie indeksów:

- partycjonowanych zarówno dla tabel partycjonowanych, jak i niepartycjonowanych
- tabela partycjonowana może natomiast posiadać zarówno indeks partycjonowany, jak i niepartycjonowany

⇒ W zależności od sposobu partycjonowania, wyróżnia się dwa rodzaje indeksów: **lokalne i globalne**

Partycjonowanie indeksów (2)

- **Indeks lokalny** (local) → sposób partycjonowania indeksu jest identyczny ze sposobem partycjonowania indeksowanej tabeli
 - identyczność partycjonowania oznacza zgodność atrybutów partycjonujących i zakresów partycji
- **Indeks globalny** → sposób partycjonowania indeksu jest inny niż sposób partycjonowania tabeli indeksowanej
 - różne atrybuty partycjonujące tabeli i indeksu
 - różne zakresy partycji tabeli i indeksu

indeks lokalny

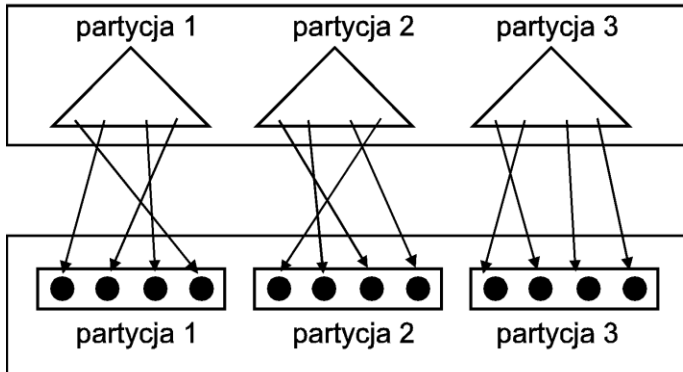


tabela partycjonowana

indeks globalny

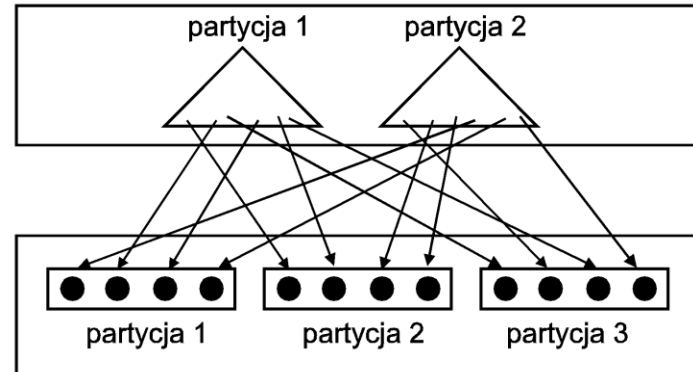


tabela partycjonowana

Partycjonowanie indeksów (3)

- Ze względu na zbiór indeksowanych atrybutów wyróżnia się indeksy prefiksowane i nieprefiksowane
 - Indeks prefiksowany (prefixed) → pierwsze kolumny indeksu bazują na atrybutach partycjonujących tabeli, z zachowaniem kolejności tych atrybutów
 - Indeks nieprefiksowany (nonprefixed) → pierwsze kolumny indeksu nie bazują na atrybutach partycjonujących tabeli
- W praktyce, najczęściej stosuje się prefiksowane indeksy lokalne, natomiast nie stosuje się indeksów globalnych bez prefiksu

Tworzenie indeksów (1)

➔ Indeks lokalny

- w poniższym przykładzie zostaną utworzone cztery odrębne indeksy — po jednym dla każdej z czterech partycji tabeli **klienci2**
- zakresy wartości indeksu dla danej partycji będą identyczne z zakresem tej partycji tabeli

```
create index klienci2_local_idx  
on klienci2(kod_miasta)  
local  
(partition p_klienci_C_idx tablespace dane1_idx,  
partition p_klienci_F_idx tablespace dane2_idx,  
partition p_klienci_M_idx tablespace dane3_idx,  
partition p_klienci_MAX_idx tablespace dane4_idx);
```

lub

```
create index klienci2_local_idx  
on klienci2(kod_miasta)  
local;
```



Tworzenie indeksów (2)

```
create table sprzedaz1
(id_sprzedazy number(10), kwota number (6,2),
 klient_id number(10), sklep_id number(10),
 rok number(4))
partition by range (rok)
(partition p_1990 values less than (1991),
 partition p_2000 values less than (2001),
 partition p_2010 values less than (2011));
```

```
create index sprzedaz1_rok_local_idx
on sprzedaz1(rok)
local;
```

```
create index
sprzedaz1_klient_local_idx
on sprzedaz1(klient_id)
local;
```

```
select index_name, partition_name,
       high_value
from user_ind_partitions;
```

```
select index_name, partition_name,
       high_value
from user_ind_partitions;
```

INDEX_NAME	PARTITION_NAME	HIGH_VALUE
SPRZEDAZ1_ROK_LOCAL_IDX	P_2010	2011
SPRZEDAZ1_ROK_LOCAL_IDX	P_2000	2001
SPRZEDAZ1_ROK_LOCAL_IDX	P_1990	1991

INDEX_NAME	PARTITION_NAME	HIGH_VALUE
SPRZEDAZ1_ROK_LOCAL_IDX	P_2010	2011
SPRZEDAZ1_ROK_LOCAL_IDX	P_2000	2001
SPRZEDAZ1_ROK_LOCAL_IDX	P_1990	1991
SPRZEDAZ1_KLIENT_LOCAL_IDX	P_2010	2011
SPRZEDAZ1_KLIENT_LOCAL_IDX	P_2000	2001
SPRZEDAZ1_KLIENT_LOCAL_IDX	P_1990	1991

Tworzenie indeksów (3)

➤ Indeks globalny

- nie może być stosowany dla tabel partycjonowanych haszowo
- w poniższym przykładzie indeks nie jest dzielony na partycje → jest indeksem niepartycjonowanym
- zawiera wskazania do wszystkich rekordów umieszczonych we wszystkich partycjach tabeli **klienci2**

```
create index klienci2_global_inx  
on klienci2(status)  
global;
```

lub

```
create index klienci2_global_inx  
on klienci2(status);
```

Tworzenie indeksów (4)

⇒ Indeks globalny podzielony na partycje

- liczba partycji tego indeksu i zakresy wartości każdej z partycji są różne od liczby i zakresu wartości partycji indeksowanej tabeli
- indeks zostanie podzielony na dwie partycje:
 - pierwsza będzie gromadzić klucze indeksu dla **kod_miasta < M**
 - druga — pozostałe

```
create index klienci2_global1_idx
on klienci2(kod_miasta)
global
partition by range (kod_miasta)
(partition p_klienci_L_idx values less than ('M')
 tablespace dane1_idx,
 partition p_klienci_MAX_idx values less than (maxvalue)
 tablespace dane2_idx);
```

musi istnieć partycja
z maxvalue



Zarządzanie indeksami partycjonowanymi

- Podział partycji na dwie (**alter index split partition**)
- Zmiana nazwy partycji (**alter index rename partition**)
- Zmodyfikowanie parametrów partycji (**alter index modify partition**)
- Oznaczenie partycji indeksu jako **unusable** (**alter index x modify partition y unusable**)
- Usunięcie partycji (**alter index drop partition**)
- Ponowne utworzenie indeksu dla wskazanej partycji (**alter index rebuild partition**)
- **Informacje słownikowe**
 - Indeksy partycjonowane – **USER_PART_INDEXES**
 - Partycje indeksu – **USER_IND_PARTITIONS** (atrybut **status {USABLE, UNUSABLE}**)

Kompresja partycji

```
create table klienci2
(klient_id number(10),
 imie varchar2(25),
 nazwisko varchar2(25),
 kod_miasta varchar2(6))
partition by range(kod_miasta)
(partition p_klienci_C values less than ('D')
    tablespace dane1 COMPRESS,
 partition p_klienci_F values less than ('G')
    tablespace dane2 COMPRESS,
 partition p_klienci_M values less than ('N')
    tablespace dane3 COMPRESS,
 partition p_klienci_MAX values less than (maxvalue)
    tablespace dane4);
```



Kompresja partycji

```
alter table Sales_M  
move partition for (to_date('29-11-2014', 'dd-mm-yyyy'))  
COMPRESS;
```