POLITECHNIKA KRAKOWSKA
IM. TADEUSZA KOŚCIUSZKI
WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI
KIERUNEK INFORMATYKA

JAKUB CZYRZEWSKI

PRZETWARZANIE W PAMIĘCI Z WYKORZYSTANIEM ORACLE DATABASE 12C

PROCESSING IN MEMORY WITH THE USE OF ORACLE DATABASE 12C

PRACA MAGISTERSKA
STUDIA STACJONARNE

Promotor: dr inż. Stanisława Plichta

Kraków 2014

Spis treści

1. Wstęp ... 4

1.1. Cel i zakres pracy ... 4

1.2. Przetwarzanie w pamięci ... 5

1.3. Baza danych w pamięci ... 6

2. Mechanizmy przetwarzania w pamięci ... 12

2.1. Oracle In-Memory Database Cache ... 12

2.1.1. Koncepcja ... 12

2.1.2. Struktury logiczne ... 14

2.1.3. Zarządzanie pamięcią operacyjną ... 18

2.1.4. Komunikacja pomiędzy Oracle Database i Oracle TimesTen ... 20

2.1.5. Delegacja żądań ... 22

2.1.6. Współbieżność ... 25

2.1.7. Przetwarzanie zapytań ... 26

2.2. Buforowanie w PL/SQL ... 28

2.2.1. Koncepcja ... 28

2.2.2. Charakterystyka ... 29

3. Środowisko pomiarowe ... 31

3.1. Maszyna wirtualna ... 31

3.2. Maszyna rzeczywista ... 32

3.3. Konfiguracja bazy danych ... 33

4. Aplikacja testowa ... 35

4.1. Koncepcja ... 35

4.2. Masowe przetwarzanie i ładowanie ... 36

4.3. Identyfikacja testowych obszarów ... 38

4.3.1. Przypadki użycia aplikacji ... 38

4.3.2. Warianty działania oprogramowania 39
4.4. Modyfikacje aplikacji 40
4.4.1. Grupy pamięci podręcznej 40
4.4.2. Logika działania 43
5. Pomiary wydajności 46
5.1. Konwencja numeracji 46
5.2. Testy 46
5.2.1. Ładowanie danych do bazy 46
5.2.2. Obliczanie statystyk 48
5.2.3. Generowanie raportów 49
5.3. Wyniki 49
5.4. Wnioski 52
6. Zakończenie 58
7. Bibliografia 61
8. Dodatek 1 – Struktura bazy danych 65
8.1. Konwencja nazewnictwa 65
8.2. Schemat aplikacji 66
9. Dodatek 2 - Spis tabel 69
10. Dodatek 3 - Spis rysunków 70

1. Wstęp

Rozdział opisuje główny cel pracy oraz jej zakres. Wprowadza czytelnika w zagadnienia związane z przetwarzaniem w pamięci oraz konfrontuje system zarządzania bazą danych w pamięci operacyjnej z konwencjonalną bazą danych, która rezyduje w pamięci masowej.

1.1. Cel i zakres pracy

Celem niniejszej pracy jest porównanie wydajności bazy danych *Oracle Database 12c Enterprise* z włączoną innowacyjną opcją *Oracle In-Memory Database Cache* oraz standardowej instancji, działającej bez pamięci podręcznej. Realizacja pozwoli zapoznać się z możliwościami mechanizmu *Oracle In-Memory Database Cache*, zwanego dalej IMDB Cache, oraz ocenić zasadność zakupu i użycia podczas dostrajania biznesowych aplikacji. Dodatkowo, pamięć podręczna zostanie skonfrontowana z dedykowanym mechanizmem buforowania na poziomie PL/SQL, który wykorzystuje wiązanie masowe oraz pamięciowe struktury danych i był dotychczas najpopularniejszą metodą optymalizacji przetwarzania dużych woluminów danych.

Porównanie wydajności zostanie wykonane poprzez zaprojektowanie i przeprowadzenie testów generujących obciążenie charakterystyczne dla środowiska transakcyjnego i hurtowni danych. Uzyskane wyniki pozwolą ocenić wpływ selektywnego buforowania podzbiorów tabel w pamięci operacyjnej na czas odpowiedzi w systemach OLTP (ang. *Online Transaction Processing*) oraz szybkość przetwarzania wsadowego i raportowania w systemach OLAP (ang. *Online Analytical Processing*). Eksperymenty zostaną zdefiniowane i przeprowadzone dla aplikacji zaimplementowanej w ramach pracy inżynierskiej pt. „Nowe cechy, funkcje oraz mechanizmy dostępne w Oracle Database 10g i 11g". Zapewnia ona odpowiednio skomplikowaną strukturę i logikę działania. Ponadto, pozwala na załadowanie dowolnej ilości danych do przetworzenia, dzięki zautomatyzowanemu mechanizmowi periodycznego zasilania obrazem odzwierciedlającym stan bieżący [1].

Aby możliwe było przeprowadzenie pomiarów wydajności, należy zrozumieć zasadę i istotę działania IMDB Cache oraz zidentyfikować obszary aplikacji testowej, w których mechanizm może zostać wykorzystany do poprawy czasu odpowiedzi.

Oprogramowanie zostanie zmodyfikowane tak, aby było zdolne do realizacji założonych scenariuszy testowych. Wiązanie masowe i kolekcje w drastyczny sposób zmieniają podejście podczas przetwarzania danych, w stosunku do konwencjonalnego rozwiązania. Z tego powodu, zostanie przybliżona idea pamięci podręcznej na poziomie PL/SQL. Buforowanie zostało zaimplementowane w aplikacji testowej w celu poprawy wydajności, zanim IMDB Cache został wprowadzony.

1.2. Przetwarzanie w pamięci

Przetwarzanie w pamięci jest zagadnieniem informatyki, którego zadaniem jest zapewnienie szybkiego oraz łatwego dostępu do danych w celu wydajnego wykonywania operacji i efektywnego podejmowania decyzji.

Historycznie, komputery są wyposażone w dwa główne rodzaje pamięci – operacyjną i masową. Pamięć operacyjna jest realizowana z wykorzystaniem RAM (ang. *Random Access Memory*), natomiast napędy HDD (ang. *Hard Disk Drive*) i SSD (ang. *Solid State Drive*) stanowią najbardziej powszechne rozwiązania dla pamięci masowej. Nowoczesne komputery posiadają do dyspozycji zdecydowanie więcej pamięci dyskowej, która charakteryzuje się wielokrotnie gorszą wydajnością od pamięci operacyjnej. W przypadku wykonywania operacji dla dużego woluminu danych, który jest w całości przechowywany na dysku, wydajność drastycznie spada. Przetwarzanie w pamięci stało się możliwe dzięki wprowadzeniu 64-bitowej adresacji RAM, która dostarcza rozwiązaniu skalowalność, poprzez zapewnienie przestrzeni roboczej mierzonej w terabajtach [2]. Podczas przetwarzania w pamięci dane posiadają gęstą, upakowaną strukturę oraz wysoki współczynnik kompresji, co powoduje, że zajmują one zdecydowanie mniej miejsca niż oryginalne informacje zapisane na dysku [3]. Pomimo tego, przechowywanie całych woluminów danych w RAM bywa niemożliwe do realizacji lub jest nieekonomiczne, ze względu na wysokie koszty sprzętu. Przetwarzanie w pamięci zapobiega problemom wydajnościowym poprzez przechowywanie najbardziej istotnych danych permanentnie w szybkiej pamięci operacyjnej.

Podobne rozwiązania stosuje się od wielu lat w centralnych jednostkach przetwarzających, wykorzystując przenoszenie istotnych danych z pamięci operacyjnej do pamięci podręcznej procesora [3]. Współcześnie, zagadnienie przetwarzania

w pamięci może być również utożsamiane z budową jednostek PIM (ang. *Processing In Memory*), w których procesor i pamięć operacyjna są zintegrowane w jeden układ scalony, aby zapewnić jeszcze mniejsze opóźnienie w komunikacji oraz większą wydajność [4].

1.3. Baza danych w pamięci

Przetwarzanie w pamięci, ze względu na swoją ideę i charakterystykę, jest obecnie najpowszechniej stosowane w bazach danych. Dotyczy to szczególnie tych, które są wykorzystywane przez narzędzia analityki biznesowej (ang. *Business Intelligence*). Baza danych w pamięci (ang. *In-Memory Database*), w porównaniu do konwencjonalnej bazy, która rezyduje na dysku, jest przechowywana w całości w pamięci operacyjnej, co zapewnia szybki dostęp [5]. Istnieje trwała kopia danych przechowywana na dysku, która jest używana podczas uruchamiania instancji, jednak to replika w RAM jest uznawana za główną. Inne charakterystyki pamięci masowej i operacyjnej wymuszają, aby system zarządzania bazą danych działał w inny sposób niż tradycyjnie. Literatura wskazuje kilka głównych różnic między pamięcią masową i operacyjną, które muszą zostać uwzględnione podczas implementacji mechanizmu zarządzania i optymalizacji dostępu do danych [5]. Są to m.in.:

a) Czas dostępu do pamięci operacyjnej jest zdecydowanie niższy niż w przypadku pamięci masowej.
b) Pamięć operacyjna jest ulotna, natomiast pamięć masowa trwała.
c) Pamięć masowa posiada wysoki, stały koszt dostępu, który musi zostać poniesiony bez względu na ilość żądanych informacji do odczytu.
d) Pamięć masowa jest zorganizowana w sposób blokowy, natomiast pamięć operacyjna adresuje bajty.
e) Sposób rozmieszczenia danych na nośniku pamięci masowej ma krytyczne znaczenie przy sekwencyjnym odczycie.
f) Dane z pamięci operacyjnej są żądane bezpośrednio przez procesor.

Dla pamięci operacyjnej nie jest istotny sposób organizacji, ponieważ dostęp do danych jest swobodny. Aby obsłużyć pamięć dyskową, wymagany jest kontroler. Jego głównym zadaniem jest efektywne sterowanie ruchem głowicy poprzez racjonalne kolejkowanie

żądań. Praca kontrolera opiera się na oprogramowaniu w pamięci wewnętrznej, które może powodować błędy. W przypadku współcześnie coraz bardziej popularnych napędów SSD, które charakteryzują się natychmiastowym dostępem do danych, za porządkowanie danych odpowiada sterownik, którego celem jest reorganizacja tak, aby występowały jak najdłuższe ciągłe obszary pustych bloków, gotowych do zapisu [6]. Mechanizm przechowywania danych ma zasadniczy wpływ na obsługę współbieżności. Od systemu bazy danych opartego o pamięć masową (ang. *Disk-Based Database*) wymaga się granulacji na poziomie pól rekordu i wierszy, w celu zapewnienia odpowiednio wysokiego stopnia współbieżności. Natomiast dla bazy danych przechowywanej w pamięci operacyjnej można oczekiwać, że transakcje będą trwały bardzo krótko, co spowoduje, że blokady również będą nakładane na zdecydowanie krótszy czas niż przy blokowaniu danych przechowywanych w pamięci masowej. Powoduje to, że poziom granulacji w bazie danych rezydującej w pamięci operacyjnej, powinien być niższy. Korzystne może okazać się blokowanie całej bazy danych i przetwarzanie transakcji w sposób uszeregowany [5]. Takie rozwiązanie niesie ze sobą następujące korzyści:

a) Całkowita eliminacja kosztu obsługi współbieżności,
b) Brak podatności na błędy spowodowane anomaliami współbieżności;

Rezygnacja z wysokiego poziomu granulacji powinna być najkorzystniejsza, w przypadku, gdy baza danych jest obsługiwana przez jeden procesor. Dla systemów wieloprocesorowych, duża granulacja danych może być konieczna nawet, jeśli czas przetwarzania pojedynczej transakcji jest bardzo krótki.

Ulotna pamięć operacyjna jest bardziej podatna na awarię zasilania, co powoduje, że, musi istnieć kopia bazy danych w pamięci masowej oraz log transakcji, powszechnie stosowany w konwencjonalnych systemach zarządzania bazą danych. Aby mechanizm odtwarzający instancję bazy danych w pamięci, w przypadku awarii, nie był wąskim gardłem i nie zniweczył korzyści wynikających z wydajnego przetwarzania transakcji, musi być on w odpowiedni sposób zorganizowany i zaimplementowany. Jednym spośród możliwych rozwiązań jest przeznaczenie niewielkiego obszaru pamięci operacyjnej w celu zapisu logów. Każda transakcja musi zostać zarejestrowana w logu przed jej formalnym zatwierdzeniem. Wykorzystanie pamięci operacyjnej w tym celu skraca czas oczekiwania na zakończenie transakcji, ponieważ żadna z nich nigdy nie czeka

na operacje dyskowe. Za zapis logu transakcji w trwałej pamięci masowej odpowiedzialny jest periodycznie uruchamiany proces, co sprawia, że cała operacja wykonywana jest w pojedynczym żądaniu do dysku. Przygotowany log transakcji pozwala w razie awarii odtworzyć stan bazy danych z momentu tuż przed wystąpieniem defektu. Jest to realizowane poprzez załadowanie bazy danych z pamięci masowej do pamięci operacyjnej oraz odtworzenie wszystkich zmian dokonanych przez transakcje. W celu ograniczenia ilości logów transakcji, jaka musi być przechowywana i aplikowana w trakcie odtwarzania, stosuje się punkty kontrolne. W momencie wystąpienia punktu kontrolnego, bieżące logi transakcji są wykonywane dla kopii bazy danych przechowywanej w pamięci masowej. Nie będą już one dłużej potrzebne, ponieważ zostały utrwalone. Zajmowana przestrzeń jest nadpisywana przez operacje wykonane po utworzeniu punktu kontrolnego.

Pomimo szybkiego dostępu do pamięci operacyjnej, istnieją metody, których zadaniem jest zwiększenie efektywności wyszukiwania danych, podobnie jak w przypadku baz danych opartych o pamięć masową. Zasadnicza różnica uwidacznia się przy strukturach danych wykorzystywanych przez indeksy. W przypadku konwencjonalnych baz danych, najpopularniejsze jest B-drzewo. Dla baz danych rezydujących w pamięci zostało specjalnie zaprojektowane T-drzewo [5]. Struktura B-drzewa gwarantuje, że wysokość będzie ograniczona i uzależniona od rzędu oraz ilości kluczy, co jest bardzo ważne dla wydajności, gdy przeszukiwany jest indeks zapisany w pamięci masowej. Wzór 1 [7] pokazuje zależność wysokości h B-drzewa od rzędu m i ilości kluczy przechowywanych n, przy założeniu, że jest ono co najmniej w połowie wypełnione. Dla $m = 200$ i $n = 2\,000\,000$, maksymalna wysokość będzie wynosić 4. Stopniem minimalnego wypełnienia B-drzewa można dowolnie sterować, aby osiągnąć optymalną wydajność, dostosowaną do konkretnego urządzenia blokowego, na co pozwala wiele systemów zarządzania bazą danych opartej na pamięci dyskowej. Istnieje uogólnienie w postaci B^n-drzewa, którego minimalne wypełnienie węzłów wynosi $\frac{n+1}{n+2}$ [7].

$$h \leq log_q \frac{n+1}{2} + 1\,, \quad q = \left\lceil \frac{m}{2} \right\rceil \qquad (1)$$

W przypadku T-drzewa wysokość nie jest restrykcyjnie ograniczona, ponieważ przechodzenie między węzłami odbywa się bardzo szybko w pamięci operacyjnej.

W bazach danych rezydujących w pamięci masowej skupiano się na technikach, których zadaniem jest optymalizacja sekwencyjnego dostępu do danych. W przypadku pamięci operacyjnej, techniki te tracą zalety [5]. Jednym z przykładów jest złączenie sortująco-scalające. Zasada jego działania polega na niezależnym przeglądzie dwóch tabel, których wiersze są sortowane po kluczu złączenia, a następnie na naprzemiennym odczycie wierszy z obu posortowanych list (od początku do końca) i odrzucaniu wierszy, które nie powinny znaleźć się wyniku [8]. Wykorzystanie całkiem innych struktur danych pozwala systemom zarządzania bazą danych w RAM na zdecydowane przyśpieszenie realizacji złączeń a w rezultacie, przyśpieszenie całych zapytań.

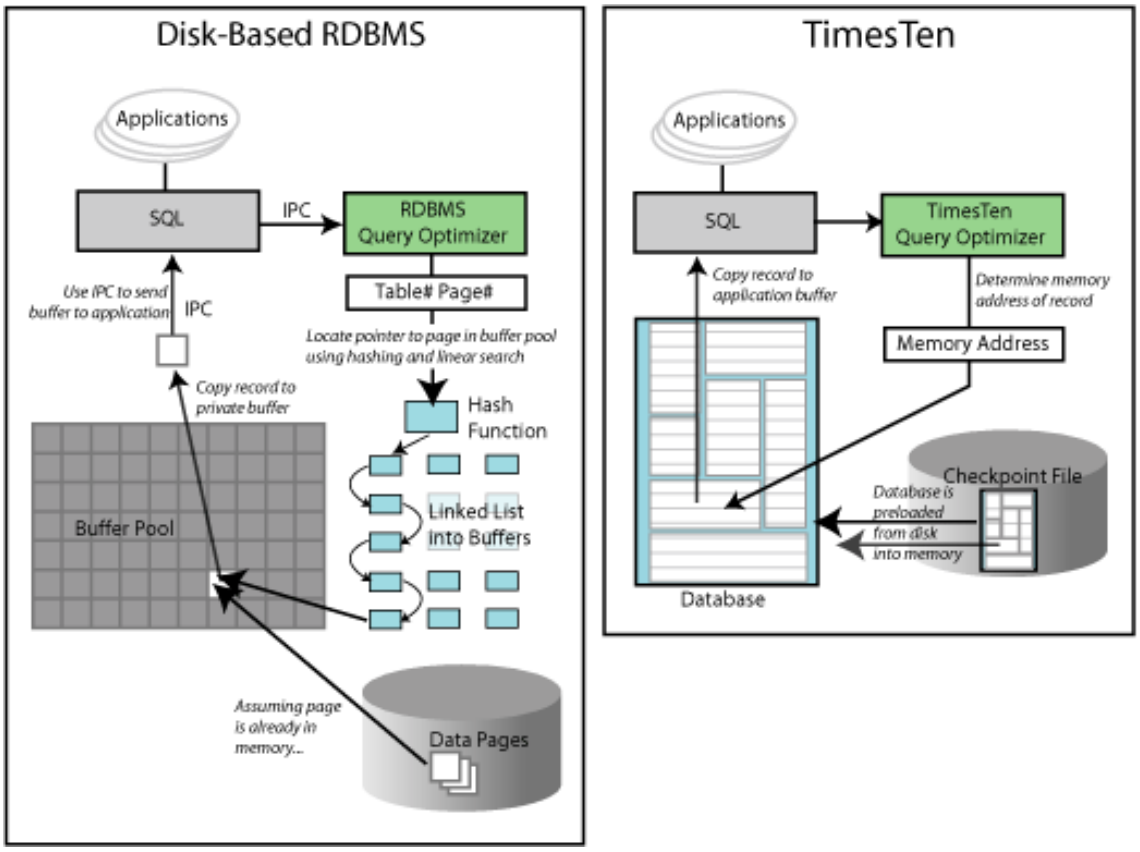
Podczas rozważań nad użyciem bazy danych w pamięci operacyjnej należy poczynić kroki, których zadaniem jest identyfikacja kluczowych obszarów działania aplikacji. W tym celu klasyfikuje się dane na dwie główne kategorie [5]:

a) Dane gorące,
b) Dane zimne;

Dane gorące to mały wolumin danych, które są często wykorzystane i dostęp do nich wymaga krótkiego czasu, aby ogólna wydajność systemu gwałtownie wzrosła. Natomiast, mianem danych zimnych określa duży zbiór danych, które są rzadko żądane, co sprawia, że koszt poniesiony podczas ich uzyskiwania nie ma wielkiego wpływu na ogólną wydajność systemu. Można ulec złudzeniu, że lepszym rozwiązaniem od klasyfikacji danych i wdrażania bazy przechowywanej w pamięci operacyjnej, będzie zwiększenie buforu zwykłej instancji bazy danych tak, aby więcej informacji mogło być jednocześnie przechowywanych w pamięci operacyjnej. Nie wniesie to jednak ze sobą pełnych korzyści, jakie można osiągać z przechowywania danych w pamięci [5]. Związane jest to z narzutem, jaki musi zostać poniesiony podczas uzyskiwania dostępu do danych przez menadżera buforu, którego zadaniem jest obliczenie adresu dyskowego żądanych informacji oraz weryfikacja, czy dany blok znajduje się aktualnie w pamięci operacyjnej. W przypadku systemu zarządzania bazą danych w RAM, dokonywana jest bezpośrednia translacja żądań na adresy w pamięci operacyjnej. Potwierdza to również *Oracle Corporation* zakładając, że nawet gdyby wszystkie dane znajdowały się w buforze *Oracle Database* to wydajność będzie kuleć z powodu mechanizmów, które zakładają, że dane znajdują się na dysku [9]. Ponadto, założenia te nie mogą być w prosty sposób zmienione, ponieważ są głęboko zaszyte w logice oprogramowania. Hector Garcia-

Molina i Kenneth Salem uważali, że dobre systemy zarządzania bazą danych oparte na pamięci masowej powinny w przyszłości wykrywać przypadki, gdy krytyczne dane znajdują się ciągle w pamięci i odnosić się do nich bezpośrednio, co spowoduje zatarcie różnić między *In-Memory Database* oraz *Disk-Based Database* [5]. Można założyć, że tak się nie stało, ponieważ obecnie istnieje wiele niezależnych implementacji systemów zarządzania bazą danych w pamięci operacyjnej, których funkcjonalność i wydajność różni się zasadniczo od relacyjnych baz danych opartych na pamięci masowej [10].

Firma *Oracle* rozwija obie koncepcje baz danych – *Oracle Database*, będącą przedstawicielem produktów bazujących na pamięci masowej oraz *Oracle TimesTen,* która opiera swoje działanie na pamięci operacyjnej. Przetwarzanie w pamięci podążyło w kierunku rozwiązania, w którym dodatkowa instancja *Oracle TimesTen* występuje w warstwie aplikacji. Kooperuje ona z *Oracle Database*, dostarczając bardzo szybkiej pamięci podręcznej dla oprogramowania [11].



Rysunek 1. Porównanie działania Oracle Database z Oracle TimesTen [9].

Technologia ta nosi nazwę *In-Memory Database Cache* i według producenta może zapewnić aż kilkudziesięciokrotne przyśpieszenie w stosunku do zwykłej instancji bazy danych, a w rezultacie zwiększyć wielokrotnie maksymalną ilość transakcji przetwarzanych w ciągu minuty [12]. Jak przedstawia rysunek 1, który konfrontuje współczesne systemy zarządzania bazą danych w pamięci masowej i operacyjnej, istnieje zgodność z literaturą [5], co do ogólnych zasad działania bazy danych w RAM oraz różnić pomiędzy dwoma modelami.

2. Mechanizmy przetwarzania w pamięci

Rozdział traktuje o mechanizmach przetwarzania w pamięci, które mogą być wykorzystane w implementacji aplikacji na platformie *Oracle Database 12c Enterprise*. Opisuje funkcjonalność oraz sposób działania *Oracle In-Memory Database Cache*. Wskazuje różnice pomiędzy nową technologią a używanym dotychczas buforowaniem na poziomie PL/SQL.

2.1. Oracle In-Memory Database Cache

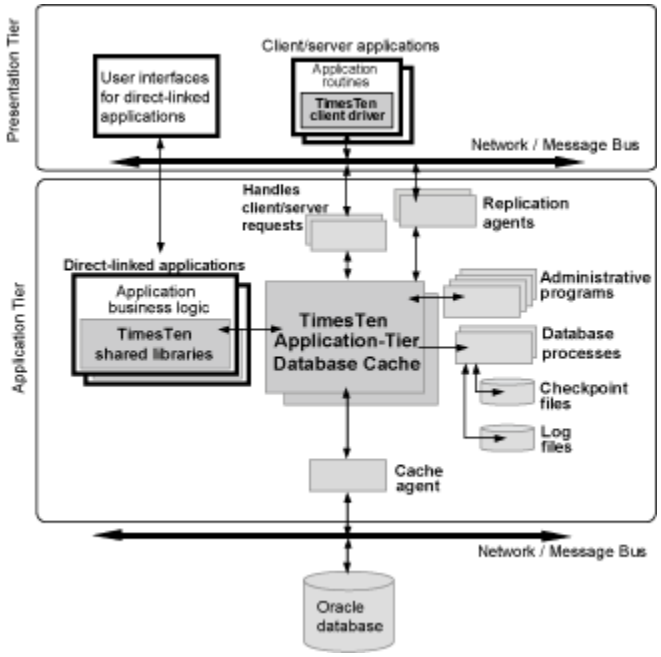
2.1.1. Koncepcja

Technologia *Oracle In-Memory Database Cache* jest produktem, którego zadaniem jest buforowanie krytycznych zbiorów danych, przechowywanych w *Oracle Database*, w celu skrócenia czasu odpowiedzi dla aplikacji użytkownika końcowego [9]. Coraz powszechniej, można spotkać się z nazwą *Oracle TimesTen Application-Tier Database Cache*. Mechanizm został zrealizowany poprzez dodanie instancji *Oracle TimesTen* w warstwie aplikacji, z którą komunikuje się bezpośrednio oprogramowanie klienckie. Baza danych *Oracle TimesTen* charakteryzuje się wysoką wydajnością, ponieważ bazuje na założeniach, że informacje są przechowywane w pamięci operacyjnej w trakcie przetwarzania. Producent podkreśla, że poprzez użycie odpowiednich struktur danych, zoptymalizowane zarządzanie pamięcią i algorytmy dostępu do danych, *Oracle TimesTen* pozwala na osiągnięcie ogromnego wzrostu przepustowości, nawet w porównaniu z instancją *Oracle Database*, która w całości byłaby zbuforowana w RAM, dzięki odpowiednio dużej przestrzeni SGA (ang. *System Global Area*) [9]. Tabele w pamięci podręcznej mogą być udostępnione w trybie tylko do odczytu lub do odczytu i zapisu. Dane tylko do odczytu mogą być automatycznie lub manualnie aktualizowane z bieżącej wersji zapisanej w bazie danych opartej o pamięć masową. W przypadku danych do odczytu i zapisu, zmiany wprowadzone w *Oracle TimesTen* są automatycznie propagowane do *Oracle Database* w trybie synchronicznym lub asynchronicznym albo manualnie. Wszystkie operacje na danych odbywają się z wykorzystaniem klasycznego SQL [9]. Zapewnione jest też wsparcie dla języka PL/SQL, w postaci zaimplementowanej w *Oracle Database* w wersji 11.2.0.2 [13].

Technologia współpracuje z następującymi wydaniami:

a) Oracle Database 12c Release 1,
b) Oracle Database 11g Release 2 w wersji co najmniej 11.2.0.2.0,
c) Oracle Database 11g Release 1 w wersji co najmniej 11.1.0.7.0,
d) Oracle Database 10g Release 2 w wersji co najmniej 10.2.0.5.0;

Rysunek 2 przedstawia schemat komunikacji między elementami wchodzącymi w skład rozważanego mechanizmu. Oprogramowanie klienckie znajdujące się w warstwie prezentacji, używając sterownika bazy danych *Oracle TimesTen*, łączy się do modułu w warstwie aplikacji, którego zadaniem jest obsługa żądań. Alternatywnie, istnieje możliwość skorzystania z interfejsu zapewniającego bezpośrednie połączenie.



Rysunek 2. Architektura mechanizmu IMDB Cache [9].

Następnie, żądanie jest kierowane do pamięci podręcznej. Za jej działanie odpowiadają procesy tła, które obsługują i zarządzają punktami kontrolnymi oraz logami transakcji. Komunikacja z *Oracle Database* odbywa się poprzez agenta buforu, który odpowiada za wczytywanie danych, odświeżanie informacji tylko do odczytu i propagowanie zmian w trybie synchronicznym lub manualnym do *Oracle Database* dla danych do odczytu i zapisu. Agenci replikacji odpowiadają za wprowadzanie zmian w innych, nadmiarowych instancjach rezydujących w RAM [14]. Rysunek 2 jest pewnym uproszczeniem architektury, ponieważ pomija szczególny proces agenta replikacji, który

wykonuje operacje w przypadku propagowania zmian w trybie asynchronicznym do bazy danych rezydującej w pamięci masowej [15]. Komunikacja pomiędzy warstwami może odbywać się za pośrednictwem sieci komputerowej lub kolejki wiadomości.

2.1.2. Struktury logiczne

Dane, które mają charakteryzować się wysokim stopniem dostępności i krótkim czasem odpowiedzi organizuje się w grupy pamięci podręcznej. Jest to podstawowa jednostka, dla której są wykonywane operacje odczytu, zapisu, synchronizacji i odświeżenia. Każda grupa pamięci podręcznej odzwierciedla jedną tabelę lub kilka powiązanych tabel z *Oracle Database* [9]. Buforowane dane mogą obejmować wszystkie kolumny i wiersze relacji lub tylko wybrane elementy. Pozwala to na horyzontalne i wertykalne skalowanie zawartości pamięci podręcznej. Każda grupa posiada tabelę-korzeń, której klucz główny jest jednocześnie kluczem głównym całej grupy. Tabela główna może być w relacji jeden-do-wielu z tabelami podrzędnymi. Każda tabela podrzędna musi posiadać klucz obcy odnoszący się do klucza głównego tabeli-korzenia lub do klucza głównego dowolnej innej tabeli podrzędnej. Istnieją cztery podstawowe typu grup pamięci podręcznej, w które można organizować buforowane tabele:

a) Tylko do odczytu,
b) Asynchronicznie propagowana,
c) Synchronicznie propagowana,
d) Manualnie zarządzana;

Dane w grupie tylko do odczytu nie mogą być modyfikowane. Zatwierdzone transakcje, wykonane w kontekście bazy przechowywanej w pamięci masowej, mogą być automatycznie propagowane do pamięci podręcznej, w celu zachowania spójności danych. W przypadku pamięci podręcznej synchronicznie propagowanej, zmodyfikowane dane w buforze są automatycznie utrwalane w bazie danych przechowywanej w pamięci masowej, w sposób asynchroniczny. Oznacza to, że zatwierdzona transakcja w kontekście *Oracle TimesTen*, nie oczekuje aż analogiczna transakcja zostanie zatwierdzona w *Oracle Database*. Grupa tego typu zapewnia krótki czas odpowiedzi i wysoką przepustowość transakcji [16]. Dodatkowo, istnieje możliwość zrównoleglenia propagacji poprzez wprowadzenie wielu wątków. Pomimo

współbieżności w zapisie, zachowywane są zależności między transakcjami. Grupa propagowana asynchronicznie gwarantuje, że żadna transakcja nie zostanie pominięta, z powodu zerwania połączenia między *Oracle Database* a *Oracle TimesTen* [15]. Jeśli agent replikacji nie jest uruchomiony lub utracił połączenie z *Oracle Database*, automatyczna propagacja zostanie wznowiona po uruchomieniu agenta replikacji lub przywróceniu połączenia z *Oracle Database*. Zatwierdzone transakcje są aplikowane w *Oracle Database* w dokładnie tej samej kolejności, co w *Oracle TimesTen*. Grupa asynchronicznie propagowana nie gwarantuje, że wszystkie transakcje zatwierdzone w pamięci podręcznej zostaną poprawnie wykonane w *Oracle Database*. Jeśli podczas zatwierdzania transakcji w bazie danych w pamięci masowej wystąpi błąd, to zostanie ona wycofana i nie będzie ponawiana. Taki scenariusz może zostać spowodowany przez naruszenie ograniczeń. Informację o transakcjach, które nie zostały wprowadzone w *Oracle Database* można uzyskać z plików **.awterr* [15]. Znajdują się one w tym samym katalogu, co pliki punktów kontrolnych. Dla grupy pamięci podręcznej synchronicznie propagowanej, zmodyfikowane dane w buforze są automatycznie utrwalane w bazie danych przechowywanej w pamięci masowej, w sposób synchroniczny. Oznacza to, że zatwierdzona transakcja w kontekście *Oracle TimesTen*, oczekuje aż analogiczna transakcja zostanie zatwierdzona w *Oracle Database*. Ten typ grupy pamięci podręcznej charakteryzuje się zdecydowanie dłuższym czasem odpowiedzi niż w przypadku asynchronicznej aktualizacji [16]. Grupa manualnie zarządzana pozwala na samodzielne zdefiniowanie zachowania. Część tabel umieszczonych w grupie pamięci podręcznej może być tylko do odczytu, natomiast pozostałe do odczytu i zapisu [15]. Dane mogą być ładowane do grup pamięci podręcznej na dwa sposoby, które są dostępne dla wszystkich typów [9]:

a) Statycznie,
b) Dynamicznie;

Dane grupy statycznie ładowanej do pamięci podręcznej są wstępnie wczytywane, zanim zostanie wykonana jakakolwiek operacja na bazie danych. Ten sposób jest właściwy, jeśli dane są na tyle niezmienne, że mogą być w większości przypadków aktualne dla uruchamianych aplikacji. Gdy nie zdefiniowano inaczej, ten tryb jest domyślny. W grupach dynamicznie wczytywanych do pamięci dane są ładowane wtedy, gdy zachodzi konieczność użycia ich. Sygnałem do umieszczenia informacji w RAM

są operacje *SELECT*, *INSERT*, *UPDATE* lub *DELETE* [15], ale tylko i wyłącznie wtedy, gdy dane jeszcze nie są załadowane lub zostały sklasyfikowane jako przestarzałe. Ten tryb może okazać się przydatny w działaniu aplikacji, jeśli cała baza danych jest większa niż dostępny rozmiar RAM w serwerze.

Dodatkowym kryterium podziału, jaki można zastosować dla grup pamięci podręcznej, jest dostępność w siatce pamięci podręcznej dla wielu instancji *Oracle TimesTen*. Ze względu na poziom widoczności, grupy dzieli się następująco:

a) Lokalna,
b) Globalna;

Jeśli grupa jest lokalna to jej zawartość nie jest współdzielona. Zachodzące zmiany mogą spowodować brak spójności danych między wieloma węzłami. Grup lokalnych należy używać, gdy istnieje tylko jedna instancja buforu lub gdy aplikacje przetwarzają dane poddane partycjonowaniu [9]. Każdy węzeł zawiera wtedy niezależny zbiór danych, związany z konkretną dziedziną działalności podmiotu. Lokalne grupy pamięci podręcznej mogą być wypełniane statycznie lub dynamicznie. W przypadku grup globalnych, zmiany zachodzące w jednej instancji buforu są propagowane do innych węzłów w obrębie siatki pamięci podręcznej, w celu zapewnienia spójności danych. Tylko asynchronicznie propagowane grupy ładowane statycznie lub dynamicznie mogą być ustawione, jako globalne.

Wyróżnia się również bardziej szczegółowe pojęcie, jakim jest instancja pamięci podręcznej. Jednostka ta wyznacza zbiór wierszy z tabel podrzędnych, powiązanych z danym rekordem tabeli nadrzędnej poprzez klucz obcy. W rezultacie, każda wartość klucza głównego z tabeli-korzenia determinuje nienależną instancję pamięci podręcznej. Ten logiczny poziom organizacji służy do zarządzania dynamicznym ładowaniem i wyznaczaniem wieku danej grupy wierszy. Dane mogą zostać odświeżone, jeśli bieżąca kopia zostanie uznana za przestarzałą. Aby idea instancji pamięci podręcznej miała sens, każda grupa musi spełniać niezależnie następujące warunki [15]:

a) Zawsze istnieje tylko jedna tabela-korzeń.
b) Tabela-korzeń nie może zawierać kluczy obcych.
c) Żadna tabela nie może mieć więcej niż jednego rodzica.

d) Każdy zbuforowany wiersz należy tylko i wyłącznie do jednej instancji pamięci podręcznej i ma tylko jeden wiersz nadrzędny.

Automatyczne ładowanie zadziała w przypadku grupy dynamicznie wczytywanej wtedy, gdy instrukcja dotyczy tylko i wyłącznie jednej instancji pamięci podręcznej [15]. Dostępne są dwa warianty poleceń, które spowodują uruchomienie mechanizmu. Pierwszym jest wykonywanie instrukcji *INSERT*, która wstawia wiersz do dowolnej tabeli podrzędnej w grupie, dla instancji pamięci podręcznej nieznajdującej się aktualnie w RAM. Drugi przypadek dotyczy instrukcji DML. Wymagają one warunku równości dla całego klucza głównego instancji pamięci podręcznej lub całego klucza obcego dowolnej tabeli w grupie pamięci podręcznej. Jeśli więcej niż jedna buforowana tabela jest obecna w klauzuli *FROM* to muszą występować między nimi złączenia warunkiem równości w klauzuli *WHERE* lub *JOIN ON*, zgodnie z występującą relacją jeden-do-wielu. Warunki muszą obejmować wszystkie kolumny kluczy głównych i obcych. Wartości filtrów mogą być sparametryzowane i bindowane na etapie wykonania. Jeśli instrukcja DQL zawiera podzapytanie to mechanizm dynamicznego ładowania może zadziałać tylko w kontekście głównego zapytania. Podczas analizy instrukcji brane są pod uwagę wyłącznie warunki jawnie zdefiniowane. Jeśli warunek wynika niejawnie z przechodniości i nie został umieszczony w zapytaniu, a jest wymagany do uruchomienia mechanizmu dynamicznego ładowania, to transparentne wczytanie danych nie nastąpi. Użycie w zapytaniu niezbuforowanych tabel nie eliminuje możliwości skorzystania z mechanizmu dynamicznego ładowania danych. Istotne jest natomiast wykorzystanie funkcjonalności postarzania instancji, opisanej w rozdziale 2.1.3. Jeśli grupa pamięci podręcznej korzysta ze starzenia się, bazującego na znaczniku czasowym, to instancja może być załadowana tylko i wyłącznie wtedy, gdy nie jest zbyt stara. Zastosowanie jednego z poniższych przypadków w SQL, eliminuje całkowicie dynamiczne ładowanie [15]:

a) Zapytanie zawiera operatory działające na zbiorach, tj. *UNION*, *INTERSECT* i *MINUS*.
b) Jedna buforowana tabela występuje więcej niż jeden raz w klauzuli *FROM*.

Na zachowanie mechanizmu transparentnego ładowania danych do RAM można wpływać poprzez manipulowanie wartością parametru *DynamicLoadEnable*, który posiada zasięg połączenia. Dostępne są trzy wartości – 0, 1 lub 2. Ustawienie wartości 0 powoduje całkowite wyłączenie mechanizmu. Wartość 1 sprawia, że dynamiczne ładowanie jest dostępne tylko i wyłącznie, gdy wszystkie buforowane tabele w zapytaniu należą do tej samej grupy pamięci podręcznej. W momencie, gdy parametr będzie równy 2, instrukcja może odnosić się do więcej niż jednej grupy pamięci podręcznej. Dla każdej z nich zostanie wykonana niezależna analiza możliwości użycia transparentnego wczytania instancji. Jeśli główne zapytanie zawiera w klauzuli *FROM* tabele z więcej niż jednej dynamicznej grupy pamięci podręcznej, to wszystkie z nich są rozważane przy wczytywaniu danych nawet, jeśli są one też używane w podzapytaniach. Podczas projektowania złożonych zapytań, które odnoszą się do wielu grup pamięci podręcznej i tabel niezbuforowanych, należy zwracać uwagę na kolejność złączeń, która może wpływać na to czy instancja zostanie wczytana do RAM [15]. Jeśli zapytanie zawiera więcej filtrów niż jest wymagane do uruchomienia dynamicznego ładowania, są one ignorowane i nie spowodują defektu.

2.1.3. Zarządzanie pamięcią operacyjną

Wyznaczanie wieku instancji pamięci podręcznej służy do automatycznego zarządzania zawartością buforu [9]. Jeśli dana grupa wierszy zostanie uznana za niepotrzebną, może zostać usunięta z pamięci operacyjnej w celu zwolnienia miejsca dla innych danych. Przestarzała i skasowana instancja, może być ponownie wczytana przez grupę dynamicznie ładowaną, gdy pojawi się żądanie dotyczące tychże danych [9]. Mechanizm zarządzania wiekiem instancji pamięci podręcznej może działać z wykorzystaniem dwóch metod [15]:

a) Znacznika czasowego,
b) Algorytmu LRU;

Obie metody mogą być wykorzystywane w tym samym czasie, w jednej instancji *Oracle TimesTen*, ale jedna grupa pamięci podręcznej jest w całości obsługiwana tylko przez jedną z nich. Tryb bazujący na znaczniku czasowym może być użyty, gdy buforowana

tabela zawiera, co najmniej jedno pole z ograniczeniem *NOT NULL* typu *DATE* lub *TIMESTAMP*. Działanie jest konfigurowane poprzez dwa atrybuty:

a) Czas życia,
b) Częstotliwość weryfikacji;

Czas życia i okres weryfikacji mogą zostać zdefiniowane w sekundach, minutach, godzinach lub dniach. Jeśli wartość kolumny typu *TIMESTAMP* lub *DATE* jest starsza niż zdefiniowany okres życia, w stosunku do aktualnej daty, to instancja jest uznawana za przestarzałą i usuwana z pamięci podręcznej. Częstotliwość weryfikacji informuje system zarządzania bazą danych, z jakim interwałem ma być uruchamiany mechanizm kasowania wierszy. Domyślnie, instancje są usuwane co 5 minut [15]. Jeśli podano 0 to weryfikacja wieku jest procesem, który sprawia wrażenie ciągłego. Gdy dla jakichkolwiek wierszy kolumna wykorzystywana do szacowania wieku nie ma sensu, a powinny one znajdować się w buforze, to należy ustawić domyślnie wartość wybiegającą daleko w przyszłość. Postarzanie wierszy wykorzystujące znacznik czasowy, w połączeniu z automatycznym okresowym odświeżaniem, tworzy okno przesuwne, które pozwala na dowolne sterowanie zawartością buforu, tak, aby zawsze była zgodna z oczekiwaniami [15]. Tryb bazujący na znaczniku czasowym może być wykorzystany w grupach pamięci podręcznej dowolnego typu. Postarzanie oparte o algorytm LRU pozwala na zarządzanie zajętością pamięci poprzez usuwanie najdawniej wykorzystywanych instancji pamięci podręcznej. Mechanizmem jest sterowany globalnie dla całej instancji *Oracle TimesTen* poprzez trzy atrybuty:

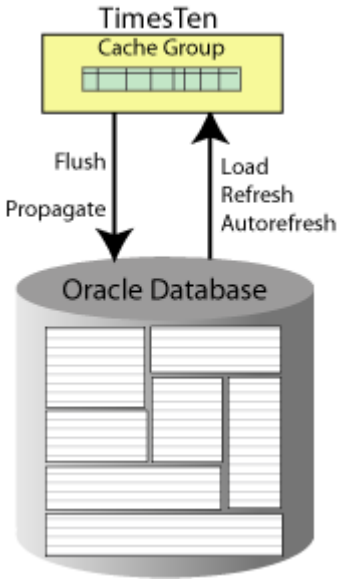
a) Wysoka granica użycia,
b) Niska granica użycia,
c) Częstotliwość weryfikacji;

Wysoka granica użycia określa aktualną zajętość pamięci, względem całej dostępnej przestrzeni dla bazy danych, przy której rozpocznie się usuwanie przestarzałych wierszy. Niska granica użycia określa zajętość pamięci, względem całej dostępnej przestrzeni dla instancji *Oracle TimesTen*, przy której usuwanie najdawniej używanych wierszy zostanie wstrzymane. Obie wartości są wyrażone w procentach i domyślnie wynoszą odpowiednio 90 i 80. Częstotliwość weryfikacji określa, z jakim interwałem jest uruchamiany mechanizm kasujący instancje pamięci podręcznej. Domyślna konfiguracja postarzania opartego na algorytmie LRU sprawia, że co 1 minutę sprawdzany jest stan zajętości

dostępnej pamięci. Jeśli poziom przekroczy 90% to usuwane są najdawniej używane wiersze, dopóki nie zostanie osiągnięta zajętość 80% lub mniej. Postarzanie na bazie algorytmu LRU współpracuje ze wszystkimi typami grup pamięci podręcznej, z wyjątkiem statycznie ładowanych.

2.1.4. Komunikacja pomiędzy Oracle Database i Oracle TimesTen

Jedną z najistotniejszych cech IMDB Cache jest zdolność do automatycznej synchronizacji danych pomiędzy *Oracle Database* a pamięcią podręczną. Na rysunku 3 przedstawiono schemat dwukierunkowej komunikacji, pomiędzy grupą pamięci podręcznej a bazą danych, zawierający wszystkie możliwe do wykonania operacje. Wymiana informacji charakteryzuje się złożoną logiką działania i innym zachowaniem, w przypadku różnych typów grup pamięci podręcznej.



Rysunek 3. Komunikacja między grupą pamięci podręcznej a Oracle Database [15].

Wykonanie *Propagate* lub *Flush* służy do aktualizacji danych w bazie przechowywanej na dysku, w przypadku wystąpienia zatwierdzonych transakcji w buforze. Pomiędzy tymi dwoma instrukcjami występuje jedna zasadnicza różnica. Pierwsza z nich jest wykonywana automatycznie przez system zarządzania bazą danych w pamięci, natomiast druga oznacza manualne zaaplikowanie wszystkich zmian, które oczekują aktualnie na utrwalenie w *Oracle Database*. Dla operacji *Flush* została zdefiniowana dedykowana klauzula języka SQL - *FLUSH CACHE GROUP* [17]. Podczas komunikacji

w przeciwną stronę, możliwe są operacje *Load*, *Refresh* i *Autorefresh*, których zadaniem jest dostarczenie danych do buforu. Operacja *Load* ma zastosowanie, gdy grupa pamięci podręcznej jest pusta i należy ją wypełnić aktualnymi informacjami z dyskowej bazy danych. Jeśli załadowanie zostanie wykonane dla uzupełnionej grupy, tylko nowe instancje pamięci podręcznej zostaną wczytane do buforu [15]. Pozostałe nie zostaną zmienione nawet, jeśli w oryginalnej bazie zostały zaktualizowane lub całkowicie usunięte. Tę operację można wykonać wykorzystując polecenie SQL - *LOAD CACHE GROUP* [17]. Wykonanie instrukcji *Refresh* spowoduje, że wszystkie instancje pamięci podręcznej zostaną zastąpione najbardziej aktualnymi danymi, odczytanymi z *Oracle Database*. W języku SQL odświeżenie danych odbywa się za pomocą klauzuli *REFRESH CACHE GROUP* [17]. Należy podkreślić, że występują zasadnicze różnice w działaniu dla grup pamięci podręcznej, które są ładowane na dwa sposoby – statycznie i dynamicznie. W przypadku grupy ładowanej statycznie, odświeżenie będzie odpowiednikiem usunięcia danych z buforu i ponownego załadowania za pomocą poleceń języka SQL - *UNLOAD CACHE GROUP* oraz *LOAD CACHE GROUP*. Dla grup inicjalizowanych dynamicznie, odświeżenie wpływa tylko na instancje pamięci podręcznej, które w *Oracle Database* zostały usunięte lub zmodyfikowane. Oznacza to, że po odświeżeniu w tabeli-korzeniu będzie tyle samo lub mniej wierszy [15]. Aby nowe wiersze pojawiły się, należy załadować instancje poleceniem *LOAD CACHE GROUP*. Alternatywnym rozwiązaniem jest wykonanie instrukcji DQL lub DML, która będzie sygnałem dla systemu zarządzania bazą danych, że należy załadować daną instancję pamięci podręcznej dynamicznie, na żądanie użytkownika. Opcja *Autorefresh*, jest wykorzystywana przez grupy pamięci podręcznej tylko do odczytu lub zarządzane manualnie. Automatycznie odświeżanie charakteryzuje się trzema atrybutami [15]:

a) Tryb odświeżenia,
b) Interwał,
c) Stan;

Tryb odświeżenia ma największy wpływ na czas wykonywania czynności. Dostępne są dwie opcje, które działają w zupełnie inny sposób:

d) Przyrostowy,
e) Pełny;

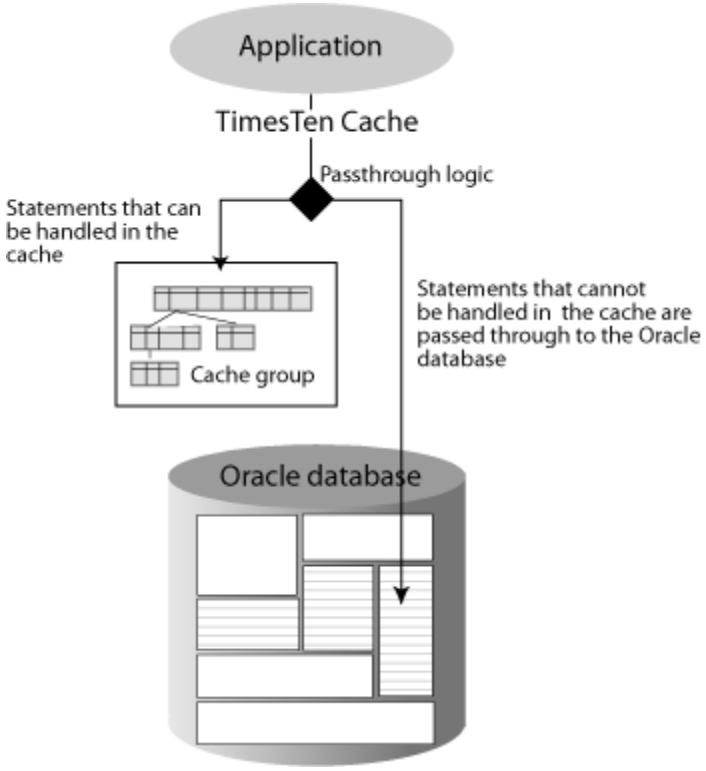
Podczas przyrostowego odświeżenia, grupa pamięci podręcznej jest periodycznie odtwarzana z wykorzystaniem plików śladu dla operacji wykonanych w *Oracle Database*. Pełne odświeżenie polega na opróżnieniu buforu i ponownym załadowaniu danych. Domyślnie wykorzystywane jest przyrostowe automatyczne odświeżenie. Interwał opisuje czas pomiędzy dwoma kolejnymi odświeżeniami. Predefiniowaną wartością jest 5 minut. Stan określa czy automatyczne odświeżenie jest wykonywane. Dopuszczalne są trzy wartości:

a) Włączone,
b) Wstrzymane,
c) Wyłączone;

Gdy odświeżanie automatyczne jest włączone, wykonuje się zgodnie z interwałem. Podczas wstrzymania, harmonogram odświeżenia jest zawieszony i zadanie nie jest wykonywane. Wszelkie zmiany w *Oracle Database* są śledzone, co pozwala na prawidłową pracę po powrocie do stanu aktywności. Jest to domyślna wartość. Jeśli automatyczne odświeżenie wyłączono to zadanie nie jest uruchamiane i pliki śladu dla zmian w dyskowej bazie danych nie są utrzymywane. Pomimo niewątpliwych korzyści wynikających z mechanizmu automatycznego okresowego odświeżania, grupa pamięci podręcznej korzystająca z tej funkcjonalności posiada wiele ograniczeń, które wskazuje literatura [15].

2.1.5. Delegacja żądań

Mechanizm IMDB Cache pozwala na przeźroczyste przekierowywanie żądań bezpośrednio do *Oracle Database*, w przypadku, gdy instancja *Oracle TimesTen* nie jest wstanie ich obsłużyć. Zasada działania została zaprezentowana na rysunku 4. Blok decyzyjny definiuje, jakie typy żądań i pod jakimi warunkami mogą zostać przesłane do realizacji w kontekście bazy danych przechowywanej w pamięci masowej. Logiką tą można manipulować poprzez ustawienie poziomu działania mechanizmu, który definiuje się na czas trwania transakcji. Po każdym zatwierdzeniu lub wycofaniu, ustawienie wraca do domyślnego trybu, który nie pozwala na delegowanie żądań.



Rysunek 4. Zasada działania mechanizmu delegacji żądań [9].

Istnieje sześć poziomów delegacji żądań, ponumerowanych od 0 do 5. Poziom zerowy jest domyślnym i oznacza, że wszystkie instrukcje powinny zostać wykonane w kontekście *Oracle TimesTen*. Jeśli nie jest to możliwe, zostanie zwrócony błąd informujący o przyczynie, np. nieistniejącej tabeli. Rozwiązaniem problemu jest ustawienie poziomu delegacji na wartość 1. Jeśli instrukcje *SELECT, INSERT, UPDATE* lub *DELETE* odwołują się do tabeli, która nie jest buforowana w *Oracle TimesTen,* to polecenie jest przekazywane do *Oracle Database* i może zakończyć się sukcesem. Instrukcje DDL nie są obsługiwane i próba wykonania kończy się zawsze błędem. Ustawienie poziomu delegacji na drugi powoduje, że instrukcje DML, wykonane dla grup pamięci podręcznej tylko do odczytu lub tabel tylko do odczytu w grupie zarządzanej manualnie, są przekazywane do *Oracle Database* w celu wykonania modyfikacji. We wszystkich innych możliwych scenariuszach poziom drugi zachowuje się tak jak pierwszy. Poziom trzeci sprawia, że wszystkie wysyłane żądania są przekazywane do wykonania w systemie zarządzania bazą danych w pamięci masowej. Wyjątkiem są instrukcje DML, które wpływają na tabele zbuforowane w globalnych grupach asynchronicznie propagowanych i spowodowały błąd w *Oracle TimesTen*. Gdy transakcja pracuje w trybie czwartym, instrukcje *SELECT* wykonane na globalnej,

dynamicznie ładowanej grupie pamięci podręcznej asynchronicznie propagowanej, które nie spełniają kryteriów dynamicznego ładowania instancji pamięci podręcznej, są przekazywane do *Oracle Database*. Pozostałe instrukcje są realizowane w buforze. Poziom piąty działa podobnie jak czwarty. Różnią się one zachowaniem w przypadku, gdy zapytanie nie umożliwia dynamicznego załadowania instancji pamięci podręcznej. Dla poziomu czwartego, żądanie zawsze zostanie oddelegowane do systemu zarządzania bazą danych w pamięci masowej, natomiast dla piątego ma to miejsce tylko wtedy, gdy wszystkie poprzednie transakcje, wykonane w kontekście połączenia, zostały spropagowane do *Oracle Database*. W przeciwnym przypadku, instrukcja DQL będzie wykonana dla bazy danych rezydującej w pamięci operacyjnej.

Dla grup pamięci podręcznej automatycznie propagowanych synchronicznie lub asynchronicznie występują anomalnie, które mogą ujawnić się podczas ręcznego wprowadzania zmian w obu instancjach niezależnie lub gdy żądanie zostanie oddelegowane. Jeśli dla grupy synchronicznie propagowanej, w tej samej transakcji jedno żądanie zostanie wykonane na buforze a drugie oddelegowane, to może nastąpić zakleszczenie podczas zatwierdzania. Wynika to z faktu, że blokady obu instrukcji mogą posiadać część wspólną, a ich wykonanie nastąpi w przybliżeniu w tym samym czasie. W przypadku grupy asynchronicznie propagowanej dane do *Oracle Database* są synchronizowane okresowo paczkami, w celu zwiększenia wydajności. Powoduje to opóźnienie w aplikowaniu zmian i szybkie zwrócenie kontekstu do aplikacji końcowej, która nie czeka na zatwierdzenie operacji w bazie danych w pamięci masowej. Jeśli w jednej transakcji, jedno żądanie zostanie wykonane na grupie pamięci podręcznej a drugie zostanie oddelegowane, to rezultat może być nieoczekiwany. Zmiany wprowadzone bezpośrednio w *Oracle Database* mogą zostać nadpisane przez automatyczną propagację danych nawet, jeśli polecenie zmieniające dane w buforze pojawiało się wcześniej w ciągu instrukcji zawartych w transakcji. W przypadku usunięcia wiersza bezpośrednio w bazie danych w pamięci masowej, *Oracle TimesTen* może próbować aktualizować wiersz, który już nie istnieje. Z powodu opisanych anomalii, należy unikać delegowania żądań dla grup automatycznie propagowanych oraz ręcznego wprowadzania niezależnych zmian w *Oracle Database* i *Oracle TimesTen*. Kod PL/SQL nie może zostać w żadnym trybie oddelegowany do bazy danych przechowywanej w pamięci masowej. Z tego względu, nie ma możliwości wykonania

procedury składowanej, która nie występuje w skompilowanej wersji w *Oracle TimesTen*, a jest w *Oracle Database*.

2.1.6. Współbieżność

Instancja *Oracle TimesTen* zapewnia współbieżny dostęp do buforu. Podstawową jednostką przetwarzania są transakcje, analogicznie jak w przypadku zwykłej bazy danych. Wspierane są dwa poziomy izolacji transakcji [9]:

a) Odczyt zatwierdzony,
b) Uszeregowany;

Jeśli transakcja jest uruchomiona w trybie odczytu zatwierdzonego to tworzy ona własne kopie rekordów i na nich wykonuje zmiany. Odczyt nie jest zablokowany dla innych transakcji. Jest to domyślny poziom izolacji. Tryb uszeregowany zapewnia niższy poziom współbieżności. Transakcja izolowana w ten sposób blokuje dane, które czyta lub modyfikuje. Próba dostępu do tych informacji przez drugą transakcję powoduje, że musi ona oczekiwać, dopóki transakcja blokująca nie zostanie zatwierdzona lub wycofana. Możliwość zakładania blokad na trzech poziomach, pozwala na sterowanie czasem odpowiedzi w zależności od zastosowania bazy danych i rodzaju operacji, jakie realizuje aplikacja korzystająca z buforu [9]:

a) Blokada wiersza,
b) Blokada tabeli,
c) Blokada bazy;

Blokady na poziomie wiersza są zakładane jedynie na wiersze, które wykorzystuje transakcja. Są zalecane, gdy wiele transakcji modyfikuje rozłączne zbiory tej samej relacji. Ze względu na największy dostępny poziom granulacji, charakteryzuje się największym opóźnieniem związanym z zapewnieniem współbieżności. W przypadku blokady na poziomie tabeli blokowana jest cała relacja. Zalecana, gdy transakcja korzysta z większości wierszy z danej tabeli. Pozwala poprawić czas odpowiedzi, ponieważ ponoszony jest mniejszy koszt obsługi współbieżności. Najniższy poziom współbieżności zapewnia blokada całej bazy danych. Transakcja, która wymaga tejże nie może rozpocząć działania, dopóki jest w toku jakakolwiek inna. Powinna być stosowana

podczas operacji, które dotykają znaczącego obszaru bazy. Charakteryzuje się najszybszym czasem odpowiedzi, ze względu na niemal całkowity brak kosztów związanych z zapewnieniem współbieżności.

2.1.7. Przetwarzanie zapytań

Zapytania kierowane do instancji *Oracle TimesTen* są przetwarzane przez dedykowany optymalizator. System zarządzania bazą danych w pamięci używa indeksów, podobnie jak *Oracle Database*, aby zoptymalizować i przyśpieszyć dostęp do danych. Do dyspozycji są trzy rodzaje indeksów:

a) Indeks mieszający,
b) Indeks zakresu,
c) Indeks bitmapowy;

Indeks mieszający wykorzystuje się do dokładnego dopasowania wartości jednej lub kilku kolumn. Jest bardzo szybki w przypadku złączeń zawierających warunek równości tzw. *equijoin* [18]. Wymaga więcej przestrzeni niż pozostałe typy indeksów. Może mieć unikalne klucze bądź nie. Z użyciem indeksu zakresu możliwe staje się znajdowanie wierszy, których wartości w jednej lub z kilku kolumn ograniczone są zakresem. Bywa pomocny w przypadku złączeń typu *theta*, które wykorzystują nierówność podczas dopasowania wierszy lub *double theta*, zawierających złączenie opisane równością i nierównością jednocześnie w dwóch osobnych warunkach. Jeśli nie zdefiniowano inaczej, użycie polecenia *CREATE INDEX*, spowoduje utworzenie indeksu tego typu [9]. Przechowywane klucze mogą być unikalne lub nie. Działanie indeksu bitmapowego nie różni się zasadniczo od standardowego, obecnego w *Oracle Database*. Służy on efektywnie, jeśli wartości występujące w kolumnie mają niewielką liczność. Zajmuje najmniej miejsca spośród wszystkich typów indeksu, ale koszt modyfikacji jest duży. Powinien być stosowany głównie w środowiskach analitycznych.

Możliwych jest kilka ścieżek dostępu do danych, których użycie zależy od informacji posiadanych przez optymalizator. Na wybór metody uzyskania wymaganych wierszy, wpływ ma selektywność filtra, operatory warunków złączeń oraz istnienie, typ i unikalność kluczy indeksu dla filtrowanych kolumn. Najprostszą metodą dostępu jest pełny przegląd tabeli (ang. *Full table scan*). Jest ona stosowana tylko

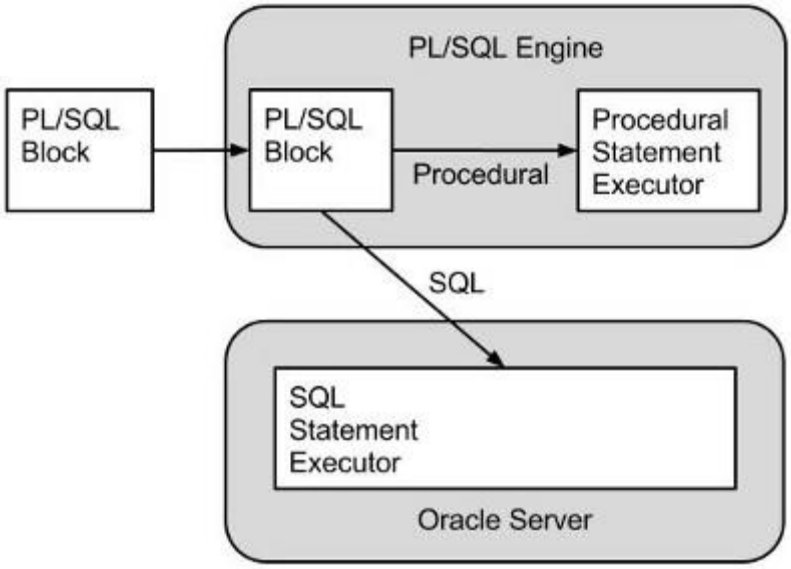
w ostateczności, ze względu na konieczność sięgnięcia do każdego wiersza [9]. Najszybszą dostępną metodą uzyskania wiersza jest wyszukiwanie po wewnętrznym identyfikatorze wiersza (ang. *Rowid lookup*). Jest on unikalny w całej bazie. Wiersz może zostać zidentyfikowany w planie zapytania z wykorzystaniem wyszukiwania po *ROWID* tylko wtedy, gdy jawnie pojawi się filtr dla pseudokolumny *ROWID*. Osobną rodziną metod dostępu do wierszy jest grupa, która wykorzystuje indeksy. Najczęściej stosowany jest przegląd zakresu indeksu (ang. *Range index scan*). Występuje on wtedy, gdy dla filtrowanych kolumn istnieje indeks zakresowy i wartości są ograniczone znakiem równości lub jednym ze znaków nierówności, z wyjątkiem nierówny. Operacja przeglądu indeksu mieszającego (ang. *Hash index lookup*) pojawi się w planie zapytania wyłącznie wtedy, gdy dla jednej bądź wielu kolumn filtrowanych równością istnieje indeks mieszający. Ostatnią dostępną ścieżką jest przegląd indeksu bitmapowego (ang. *Bitmap index lookup*). Zostanie ona wykorzystana, gdy kolumna, dla której istnieje indeks bitmapowy jest filtrowana równością.

Zadaniem optymalizatora jest też zadecydować, jaka metoda złączenia będzie najefektywniejsza. We wstępie wspomniano, że niektóre ścieżki dostępu do danych, wykorzystywane przez bazę przechowywaną w pamięci masowej, nie mają sensu w przypadku baz danych opartych na pamięci operacyjnej. Potwierdzeniem jest informacja od *Oracle*, że optymalizator buforu nie użyje złączeń innych niż pętli zagnieżdżonych (z wykorzystaniem indeksu i bez) i scalającego [9]. Złączenie pętli zagnieżdżonych bez wykorzystania indeksu może mieć miejsce wtedy, gdy kolumny po obu stronach warunku złączenia nie posiadają indeksu. Nie jest to jednak pewne. Czasami optymalizator może podjąć decyzję o stworzeniu tymczasowego indeksu dla kolumny złączenia z tabeli wewnętrznej [9]. Wtedy złączenie pętli zagnieżdżonych bez indeksu zostanie zmienione na złączenie z indeksem i operacja będzie wykona w dokładnie ten sam sposób, jak gdyby indeks był utworzony permanentnie. Złączenie scalające zostanie wykonane, gdy obie kolumny biorące udział w warunku złączenia posiadają indeks zakresowy, którego klucze są posortowane. Jeśli indeksy nie zostały zbudowane, optymalizator może zadecydować o stworzeniu tymczasowych, tylko po to, aby wykonać złączenie scalające [9].

2.2. Buforowanie w PL/SQL

2.2.1. Koncepcja

Dedykowany mechanizm buforowania w PL/SQL opiera swoje działanie na rekordach i kolekcjach oraz wiązaniu masowym (ang. *Bulk Binding*) [19]. Instancje rekordów i kolekcji są przechowywane w pamięci operacyjnej podczas wykonywania bloku kodu. Wiązanie masowe służą do wydajnego transferu danych między pamięcią operacyjną a bazą danych. Schemat komunikacji pomiędzy mechanizmami przetwarzania PL/SQL i SQL został przedstawiony na rysunku 5. Aby wykonać zapytanie z klauzulą *SELECT INTO* lub instrukcję DML, mechanizm przetwarzania PL/SQL wysyła zapytanie lub polecenie do interpretera i egzekutora SQL, który zwraca wynik wykonania do kontekstu działania języka PL/SQL. Wiązanie masowe minimalizuje narzut, jaki musi zostać poniesiony przy sekwencyjnym wykonywaniu instrukcji SQL. Przed rozpoczęciem przetwarzania dane są w całości lub paczkami ładowane do pamięci operacyjnej. Tak przygotowany zbiór danych może zostać przetworzony w pętli, nie komunikując się z system zarządzania bazą danych. Po zakończeniu dane, które znajdują się w RAM, mogą zostać utrwalone z ponownym wykorzystaniem wiązania masowego.



Rysunek 5. Schemat wykonania kodu SQL w bloku PL/SQL [20].

Możliwość przechowywania zbioru elementów dokładnie tego samego typu została wprowadzona wraz z *Oracle Database* w wersji 7 [21]. Początkowo był to tylko jeden typ kolekcji znany, jako tablica asocjacyjna (ang. *Associative array* lub *Index-by table*). W *Oracle Database* w wersji 8 zostały dodane dwa nowe typy – tablica

zagnieżdżona (ang. *Nested table*) i zwykła tablica statyczna (ang. *Varray*) [21]. Literatura [19] wskazuje zalety i wady każdego typu kolekcji oraz podaje zalecenia pozwalające na słuszny wybór odpowiedniego rodzaju, dla danego zastosowania. Wiązania masowe zostały wprowadzone w *Oracle Database* w wersji 8, a następnie rozszerzone w wersji 9 [20]. Ich zastosowanie jest bardzo proste i opiera się na dwóch instrukcjach:

a) FORALL,
b) BULK COLLECT;

Wyrażenie *FORALL*, które może być kojarzone z pętlą *FOR EACH* znaną z wielu języków programowania, wysyła porcję instrukcji DML do systemu zarządzania bazą danych, w celu przetworzenia w jednym żądaniu. Wykonanie z użyciem tej konstrukcji jest zawsze dużo wydajniejsze niż użycie *FOR LOOP* [18]. Klauzula zawsze zawiera definicję tylko jednego polecenia DML, która jest powtarzana dla każdego elementu kolekcji bez przełączania kontekstu. Zwykła pętla, może zawierać dowolną ilość operacji, ale każda z nich powoduje natychmiastowe wykonanie i komunikację z bazą danych. Instrukcja *BULK COLLECT* pobiera w jednym żądaniu porcję informacji wykorzystując DQL lub kursor i zapisuje wynik w tabeli zagnieżdżonej, rezydującej w pamięci operacyjnej. Jeśli wolumin danych jest na tyle duży, że nie mieści się w całości w RAM, to istnieje możliwość wczytywania danych paczkami o określonym rozmiarze.

2.2.2. Charakterystyka

Pamięć podręczna zrealizowana na poziomie PL/SQL za pomocą kolekcji i wiązań masowych jest dużo prostsza w implementacji niż mechanizmy dostępne w IMDB Cache. Posiada następujące cechy, które mogą być w większości przypadków rozważane, jako wady:

a) Zasięg sesji lub bloku kodu,
b) Brak współbieżności,
c) Niska odporność na awarię,
d) Manualna implementacja szczegółowych struktur danych i logiki działania;

Zasięg sesji ma zastosowanie dla buforowanych danych, które są przechowywane w globalnych kolekcjach na poziomie pakietów. Są one wtedy dostępne jedynie

w kontekście połączenia nawiązanego z serwerem *Oracle Database*. W przypadku lokalnych zmiennych, pamięć podręczna ma taki sam zasięg jak instancja kolekcji, czyli odpowiada zasięgowi zmiennej. Istnieje możliwość zaimplementowania komunikacji międzysesyjnej z użyciem pakietu *DBMS_PIPE* [19], co oznacza, że można wyeliminować tę wadę odpowiednim nakładem pracy. Struktury danych wbudowane w PL/SQL mogą być przetwarzane przez tylko jeden wątek jednocześnie, ze względu na brak blokad. Istnieje jednak możliwość współbieżnego przetwarzania z użyciem pakietu *DBMS_SCHEDULER*, który służy do uruchamiania wielu zadań jednocześnie [19]. Należy zaimplementować procedurę z parametrami, które będą ograniczać obustronnie zakres przetwarzania tak, aby woluminy danych dla każdego wywołania były rozłączne. Wielokrotne uruchomienie tej samej procedury z różnymi parametrami sprawi, że każda z nich załaduje własny zbiór danych do lokalnych kolekcji zadania. Przetwarzane dane nie są bezpiecznie, ponieważ awaria bazy lub zerwanie połączenia przed zapisem buforu oznacza całkowitą utratę wyników. Dla tej przypadłości, nie istnieje łatwy sposób na wyeliminowanie. Pakiet *UTL_FILE* pozwala na zapis plików na dysku [19], jednak zniwelowałoby to korzyści płynące z optymalizacji. W przypadku pamięci podręcznej na poziomie PL/SQL to programista jest odpowiedzialny za zdefiniowanie rekordów i kolekcji bazujących na niestandardowych typach. Dodatkowo, jego obowiązkiem jest zaimplementowanie logiki działania. W klasycznym przypadku mogą to być procedury *Load*, *Spool* i *Flush*, które posłużą odpowiednio do załadowania pamięci podręcznej, dodania zmiany oczekującej na zapis i utrwalenia oczekujących zmian w pamięci dyskowej.

3. Środowisko pomiarowe

Rozdział opisuje koncepcję oraz konfigurację środowiska pomiarowego. Szczególnie skupia się udostępnionych zasobach, rozmieszczeniu danych na nośniku fizycznym oraz ustawieniach sieciowych. Przedstawia parametry maszyny, która została wykorzystana do wykonania pomiarów oraz rezultaty testów wzorcowych dla pamięci masowej i operacyjnej. Zawiera również szczegółowe ustawienia środowiska bazodanowego. Może zostać ono odtworzone poprzez przygotowanie systemu operacyjnego oraz instalację *Oracle Database*. Przebieg obu jest opisany w pracy inżynierskiej [1]. Proces instalacji *Oracle TimesTen* [22] oraz konfiguracja mechanizmu IMDB Cache [23] zostały przedstawione dokładnie przez producenta w formie przykładu.

3.1. Maszyna wirtualna

Testowa instancja bazy danych została uruchomiona w kontekście systemu wirtualnego. Jako środowisko uruchomieniowe, wykorzystany został program *Oracle VM VirtualBox* w wersji 4.3.10. Rolę wirtualnego systemu operacyjnego pełni 64-bitowy *Oracle Linux 6.5* z jądrem w wersji 2.6.39. System ten zapewnia pełne wsparcie dla bazodanowych produktów firmy *Oracle*. Maszyna wirtualna posiada do dyspozycji 8 GB RAM, 4 rdzenie procesora (2 fizyczne z technologią *Hyper-Threading*), dwie wirtualne karty sieciowe oraz wirtualny dysk o wielkości 50 GB. W tabeli 1 znajduje się rozkład woluminów na tymże dysku. Przy podziale, skorzystano z *Logical Volume Manager*. Pozwoli to na późniejsze dostrojenie wielkości woluminów, w zależności od realnych potrzeb [24]. Strukturę zaprojektowano zgodnie ze standardem *Optimal Flexibile Architecture,* który opisuje hierarchię katalogów oprogramowania *Oracle* [25]. Punkty montowania */u01*, */u02* oraz */u03* przechowują odpowiednio oprogramowanie, obszar odzyskiwania oraz dane dla 64-bitowej instancji *Oracle Database* w wersji 12.1.0.1.0. Punkt montowania */u04* zawiera oprogramowanie 64-bitowej instancji *Oracle TimesTen* w wersji 11.2.2.6.0. Permanentne dane bazy danych rezydującej w pamięci są przechowywane w katalogu */u05*. Interfejs sieciowy *eth0* został skonfigurowany w trybie *Bridged Adapter*, dzięki czemu będzie otrzymywał od serwera DHCP unikalny adres IP w sieci lokalnej. Interfejs *eth1* został

skonfigurowany w trybie *Host-only Adapter*. Pozwoli to na korzystanie z serwera bazy danych, na rzeczywistym systemie hosta, w przypadku braku dostępu do sieci.

Urządzenie	Wielkość	Punkt montowania	System plików
/dev/sda1	1 GB	/boot	Ext4
/dev/mapper/oradbserver-lv_root	7 GB	/	Ext4
/dev/mapper/oradbserver-lv_swap	8 GB	-	SWAP
/dev/mapper/oradbserver-lv_tmp	2 GB	/tmp	Ext4
/dev/mapper/oradbserver-lv_var	5 GB	/var	Ext4
/dev/mapper/oradbserver-lv_home	5 GB	/home	Ext4
/dev/mapper/oradbserver-lv_oradbsoft	6,5 GB	/u01	Ext4
/dev/mapper/oradbserver-lv_oradbfra	5,5 GB	/u02	Ext4
/dev/mapper/oradbserver-lv_oradbdata	5 GB	/u03	Ext4
/dev/mapper/oradbserver-lv_orattsoft	2 GB	/u04	Ext4
/dev/mapper/oradbserver-lv_orattdata	2 GB	/u05	Ext4

Tabela 1. Rozkład woluminów na dysku maszyny wirtualnej

3.2. Maszyna rzeczywista

Maszyna rzeczywista, będąca hostem dla wirtualnego systemu, działa pod kontrolą systemu operacyjnego *Windows 8.1 Pro*. W tabeli 2 znajduje się jej specyfikacja.

Podzespół	Model
Procesor	Intel® Core™ i7-3610QM
Pamięć operacyjna	16 GB DDR3 1600 MHz
Pamięć masowa	VTX4-25SAT3-256G

Tabela 2. Specyfikacja rzeczywistej maszyny hosta.

W tabeli 3 znajduje się zestawienie opisujące wydajność pamięci masowej zastosowanej w maszynie hosta. Została ona zbadana za pomocą programu *Crystal Disk Mark* w wersji 3.0.3b. Każdy wynik jest średnią arytmetyczną pięciu pomiarów.

Rodzaj	Wielkość bloku	Odczyt	Zapis
Sekwencyjny	-	397 MB/s	421 MB/s
Swobodny	4 KB QD32	131 MB/s	102 MB/s
	4 KB QD1	20 MB/s	35 MB/s
	512 KB	324 MB/s	407 MB/s

Tabela 3. Wydajność pamięci masowej maszyny hosta.

W tabeli 4 znajduje się zestawienie opisujące wydajność pamięci operacyjnej zastosowanej w maszynie hosta. Została ona zbadana za pomocą programu *MaxxMEMM* w wersji 2.01. Każdy wynik jest średnią arytmetyczną pięciu pomiarów.

Operacja	Rezultat
Kopiowanie	17,1 GB/s
Odczyt	15,6 GB/s
Zapis	14,6 GB/s
Opóźnienie	51 ns

Tabela 4. Wydajność pamięci operacyjnej maszyny hosta.

3.3. Konfiguracja bazy danych

W tabeli 5 przedstawiono wartości parametrów dla instancji *Oracle TimesTen*, która została wykorzystana, jako część składowa mechanizmu IMDB Cache. Spośród najważniejszych, należy wyróżnić obszar w pamięci operacyjnej przydzielony dla bazy danych oraz tymczasowy obszar roboczy, który służy do przechowywania częściowych wyników złączeń oraz posortowanych zbiorów danych. Wartości te zostały ustalone odpowiednio, jako 1024 MB i 512 MB. Parametr *OracleNetServiceName*, wskazuje instancję *Oracle Database*, która jest buforowana. Jeśli nie podano, to baza danych pracuje z domyślną wartością, zgodną z dokumentacją producenta [26].

Parametr instancji	Wartość
Driver	/u04/app/timesten/product/11.2.2/dbhome_1/lib/libtten.so
DataStore	/u05/app/timesten/ttdata/testcachett/testcachett
LogDir	/u05/app/timesten/ttlog/testcachett
PermSize	1024
TempSize	512
PLSQL	1
DatabaseCharacterSet	AL32UTF8
OracleNetServiceName	testdb

Tabela 5. Konfiguracja testowej instancji Oracle TimesTen.

W tabeli 6 pokazano konfigurację instancji *Oracle Database*, która wykorzystuje pamięć podręczną w celu zwiększenia wydajności. Baza danych ma do dyspozycji ponad 3 GB pamięci operacyjnej, która może zostać przydzielona w zależności od potrzeb, dla SGA i PGA (ang. *Program Global Area*). Dodatkowo, dysponuje 5 GB pamięci masowej, która

służy za obszar odzyskiwania danych. Wielkość bloku jest równa 8 KB. Jeśli nie podano, to instancja pracuje z domyślną wartością, zgodną z dokumentacją producenta [27].

Parametr instancji	Wartość
db_block_size	8192
open_cursors	300
db_domain	""
db_name	"testdb"
core_dump_dest	?/dbs
control_files	("/u03/app/oracle/oradata/testdb/control01.ctl", "/u02/app/oracle/oradata/testdb/control02.ctl")
db_recovery_file_dest	"/u02/app/oracle/fast_recovery_area"
db_recovery_file_dest_size	5120m
compatible	12.1.0.0.0
dg_broker_config_file1	?/dbs/dr1testdb.dat
diagnostic_dest	/u01/app/Oracle
memory_target	3160m
local_listener	LISTENER_TESTDB
processes	300
audit_file_dest	"/u01/app/oracle/admin/testdb/adump"
audit_trail	db
remote_login_passwordfile	EXCLUSIVE
dispatchers	"(PROTOCOL=TCP) (SERVICE=testdbXDB)"
undo_tablespace	UNDOTBS1

Tabela 6. Konfiguracja testowej instancji Oracle Database.

4. Aplikacja testowa

Niniejszy rozdział opisuje zagadnienia związane z aplikacją testową. Przedstawia opis słowny systemu oraz modelowanego środowiska, który jest wyciąganiem istotnych informacji z pracy inżynierskiej [1]. Identyfikuje obszary zastosowania IMDB Cache oraz przypadki testowe dla pomiarów wydajności. Prezentuje zmiany, jakie muszą zostać wprowadzone w aplikacji przed wykonaniem zasadniczej części pracy.

4.1. Koncepcja

Aplikacja służy do zbierania, przetwarzania, ładowania oraz przechowywania informacji z gry *Ogame*, dostępnej przez przeglądarkę internetową. Jest ona zlokalizowana dla dużej liczby krajów. W obrębie jednej domeny (kraju) występuje wiele światów. Na każdym z nich rywalizuje rzesza graczy, którzy mogą jednoczyć się w sojuszach. Polega ona na eksploracji przestrzeni kosmicznej, kolonizowaniu i rozbudowie planet oraz przeprowadzaniu ataków na innych graczy w celach zarobkowych, za pomocą zbudowanej flotylli. W grze dostępne są statystyki aktualizowane w czasie rzeczywistym. Dotyczą one zarówno sojuszy jak i kont graczy. Ponadto można wyróżnić stany punktowe różnego typu, które mają zastosowanie w kontekście sojuszu i konta gracza. Celem gry jest uzyskanie jak największej liczby punktów ogólnych. Sojusze oraz konta graczy zmieniają się w trakcie trwania rozgrywki. Gracz może zmienić nazwę lub przenieść swoją planetę macierzystą. Sojusz może zmienić nazwę lub znacznik. Zadaniem aplikacji jest periodyczne zbieranie statystyk całego świata dla graczy i sojuszy, które egzystują na nim. Ilość załadowanych jednorazowo stanów punktowych z jednego świata n_{sp} jest wyrażona wzorem 2, gdzie n_g to ilość graczy a n_s określa ilość sojuszy. Liczba 8 oznacza ilość typów stanów punktowych, które są możliwe do pobrania.

$$n_{sp} = 8 * n_g + 8 * n_s \qquad (2)$$

Okres zbierania statystyk można ustawić na dowolny czas, co pozwala na skalowanie wielkości testowej bazy danych według uznania. W produkcyjnym zastosowaniu aplikacja zabiera stany punktowe raz na dobę. Zakładając, że na jednym świecie rozgrywkę prowadzi 3000 graczy, którzy są zrzeszeni w 500 sojuszy, to zgodnie

ze wzorem 2, pozwoli to jednorazowo na zgromadzenie 28 000 stanów punktowych. W ciągu tygodnia zapewnia to na załadowanie 196 000 wierszy. Ponadto, baza danych przechowuje informacje o wszystkich graczach i sojuszach wraz z pełną historią zmian, które zachodziły. Przy ładowaniu statystyk są tworzone śladowe ilości wierszy historycznych. Pierwsze użycie programu zawsze prowadzi do utworzenia rekordu dla każdego sojuszu i konta. Podsumowując, baza danych oraz oprogramowanie do przetwarzania i ładowania rekordów, są dostosowane w pełni do modelowania realiów gry. Istotne, z punktu widzenia aplikacji, są następuje fakty:

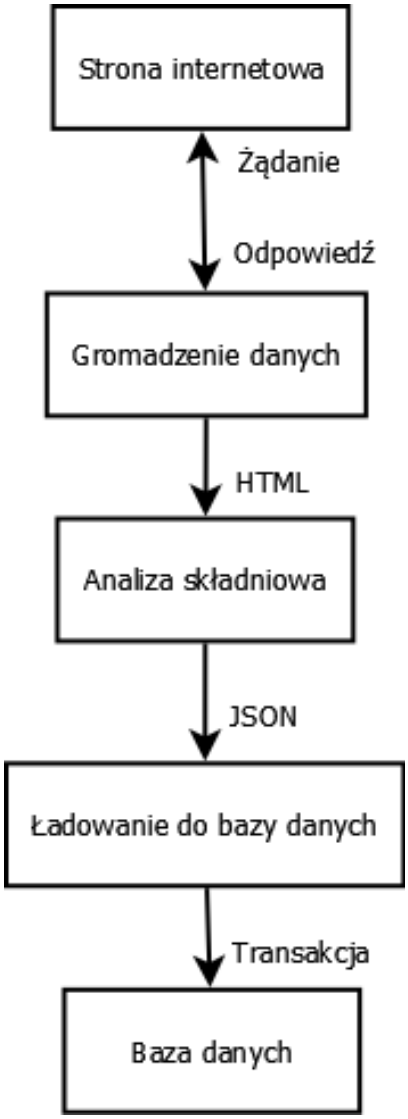
a) Gra dostępna jest w wielu krajach.
b) Gra toczy się na wielu niezależnych światach w obrębie jednego kraju.
c) Dla każdego kraju nazwy światów są takie same.
d) Każdy świat zawiera w sobie pewną ilość kont graczy i sojuszy.
e) Każdy gracz może, ale nie musi należeć do sojuszu.
f) Stany punktowe obejmują zarówno konta graczy jak i sojusze.
g) Istnieje wiele typów stanów punktowych, które są stosowane w kontekście sojuszu i konta gracza.
h) Statystyki są zbierane periodycznie.
i) Dane kont graczy oraz sojuszy mogą zmieniać się.
j) Encje posiadają szereg atrybutów niewymienionych w opisie, które należy przechowywać w bazie danych.

4.2. Masowe przetwarzanie i ładowanie

Aplikacja do masowego przetwarzania i ładowania stanów punktowych została zaimplementowana w języku *JAVA SE* i ma charakter wsadowy. Wybrane podejście zapewnia bezpieczeństwo, które należy rozumieć, jako uniknięcie możliwości utraty danych ze względu na skomplikowane procesy przetwarzania. Każdy etap jest realizowany w pełni niezależnie. Wyróżniono następujące etapy:

a) Gromadzenie danych,
b) Analiza składniowa,
c) Ładowanie do bazy danych;

Na rysunku 6 został przedstawiony poglądowy schemat przepływu danych. Pokazuje on jak zmienia się postać informacji w trakcie procesu masowego przetwarzania i ładowania.



Rysunek 6. Przepływ informacji pomiędzy modułami aplikacji [1].

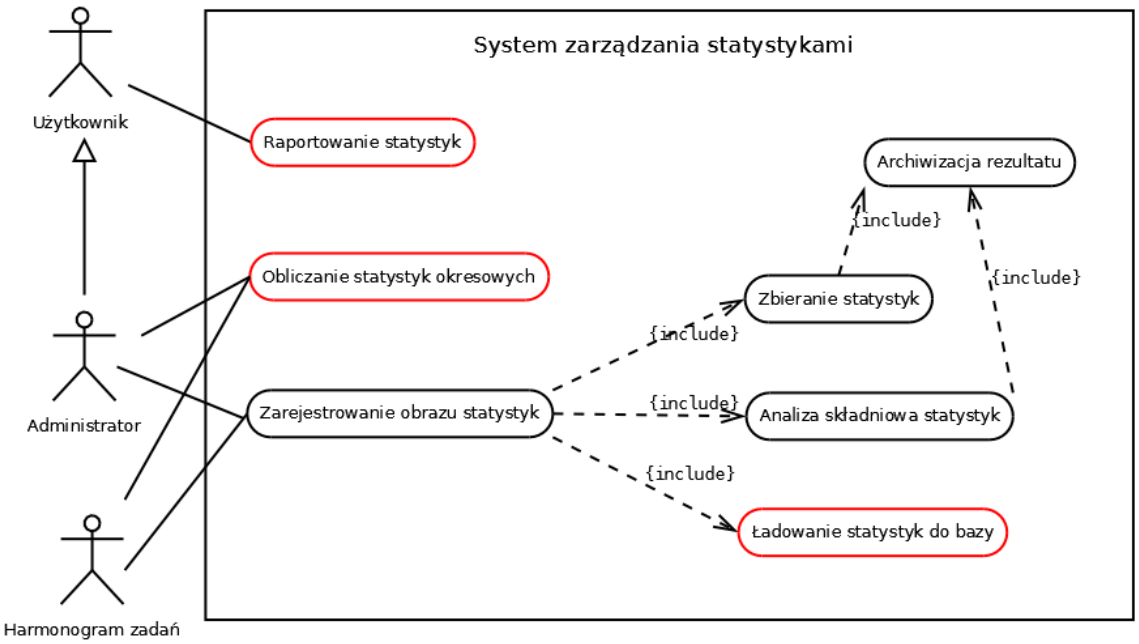
Elementem działania aplikacji, który odpowiada za bezpośrednią komunikację z bazą danych, jest proces ładowania. W początkowej fazie informacje z plików, zarchiwizowanych przez etap analizy składniowej, są wczytywane do pamięci operacyjnej. W trakcie przetwarzania poszczególne rekordy są wstawiane do tabel za pośrednictwem interfejsu, stworzonego w PL/SQL. Logika encji, odpowiedzialna za kontrolę integralności oraz historii, umieszczona jest w pakietach o tej samej nazwie, co encja, ale z prefiksem „p”. Takie podejście pozwala na szybką implementację oraz skorzystanie w przyszłości z obiektów oraz widoków obiektowych, poprzez import

istniejącej logiki z ciała pakietu do metod obiektu. Atomowa operacja wstawienia stanu punktowego polega na weryfikacji czy dany gracz lub sojusz istnieje. Jeśli istnieje to należy sprawdzić czy zaszły jakiekolwiek zmiany oraz zaktualizować rejestr historii, jeśli jest to konieczne. W przeciwnym przypadku należy stworzyć nowy rekord. Gracze i sojusze są odszukiwane za pomocą zewnętrznego identyfikatora, który jest uzyskiwany na etapie analizy składniowej. Nie ulega on zmianie przez cały czas istnienia danego konta lub sojuszu. Po załadowaniu wszystkich stanów punktowych unieważniane są konta i sojusze, dla których nie pojawił się żaden stan punktowy w aktualnie przetwarzanym znaczniku czasowym. Oznacza to, że nie istnieją one już. Następnie weryfikowana jest integralność bazy danych poprzez sprawdzenie czy rozkład stanów punktowych ze względu na typ jest idealnie równomierny. Jeśli liczby rekordów dla poszczególnych typów stanu punktowego są różne, transakcja zostaje wycofana i podnoszony jest wyjątek.

4.3. Identyfikacja testowych obszarów

4.3.1. Przypadki użycia aplikacji

Na rysunku 7 przedstawiono diagram przypadków użycia aplikacji, która zostanie wykorzystana do pomiarów wydajności.



Rysunek 7. Przypadki użycia aplikacji testowej.

Operacje na bazie danych wykonywane są podczas ładowania, obliczania i raportowania statystyk. Rozważane przypadki zostały wyróżnione kolorem czerwonym. Oznacza to, że te trzy w/w obszary powinny zostać zawarte w testach.

4.3.2. Warianty działania oprogramowania

Podczas testów należy rozważyć różne warianty działania oprogramowania dla poszczególnych operacji. Podczas ładowania statystyk do bazy należy wziąć pod uwagę następujące parametry, które wpływają na wydajność przetwarzania:

a) Grupowanie danych,
b) Grupowanie instrukcji,
c) Zakres transakcji,
d) Wykorzystanie buforu PL/SQL,
e) Użycie IMDB Cache;

Grupowanie danych opisuje formę wysyłania informacji do systemu zarządzania bazą danych. Może się to odbyć w jednej instrukcji, gdy ładowany jest cały obraz jednocześnie w formie paczki reprezentowanej przez kolekcję. Drugą możliwością jest wysłanie danych w wielu poleceniach, gdzie każdy stan punktowy jest przesyłany osobno, jako typ rekordowy. Grupowanie instrukcji polega na manipulowaniu sposobem realizacji poleceń PL/SQL. Możliwe jest przesyłanie wielu instrukcji w jednej paczce lub każdej z nich osobno do systemu zarządzania bazą danych. Grupowanie można rozważać tylko i wyłącznie wtedy, gdy stany punktowe są ładowane w wielu niezależnych instrukcjach. Zgrupowane instrukcje zostaną przesłane do bazy wraz z danymi w jednym żądaniu. Zakłada się, że grupowanie instrukcji powinno dać podobne korzyści, co wykonanie jednej instrukcji dla całego zbioru danych, w przypadku przesłania danych zgrupowanych w kolekcji. Pozwoli to na usunięcie opóźnienia powodowanego przez przełączanie kontekstu aplikacji pomiędzy maszyną wirtualną języka *JAVA* a systemem zarządzania bazą danych. Dodatkowo, działanie można zróżnicować poprzez ładowanie każdego stanu punktowego w osobnej transakcji lub wszystkich w jednej, co wpływa na charakter głównego zastosowania systemu bazodanowego. Wykorzystanie buforu PL/SQL określa koncepcję, z jaką logika aplikacji odnosi się do danych. W przypadku buforu PL/SQL operacje będą wykonywane częściej na kolekcjach niż na tabelach. Ilość pojedynczych

instrukcji DML zostaje mocno ograniczona. Użycie bądź pominięcie IMDB Cache polega na wprowadzeniu lub usunięciu dodatkowej warstwy danych umieszczonej ponad logiką aplikacji, która jest zaimplementowana w PL/SQL. Będzie ona wykonywać dokładnie ten sam skompilowany kod, ale wykorzystując *Oracle TimesTen* lub *Oracle Database* w zależności od wybranego sterownika połączenia. Manipulowanie tym parametrem pozwoli na realizację tematu przewodniego tejże pracy. Przy obliczaniu statystyk istnieje zdecydowanie mniej możliwości, ponieważ operacje wykonywane są na danych, które w całości znajdują się już bazie. Należy modyfikować następujące atrybuty:

a) Wykorzystanie buforu PL/SQL,
b) Użycie IMDB Cache;

Podczas generowania raportów, również wszystkie dane już są obecne w bazie danych. W trakcie testów wydajności dla raportowania należy wziąć pod uwagę następujące parametry:

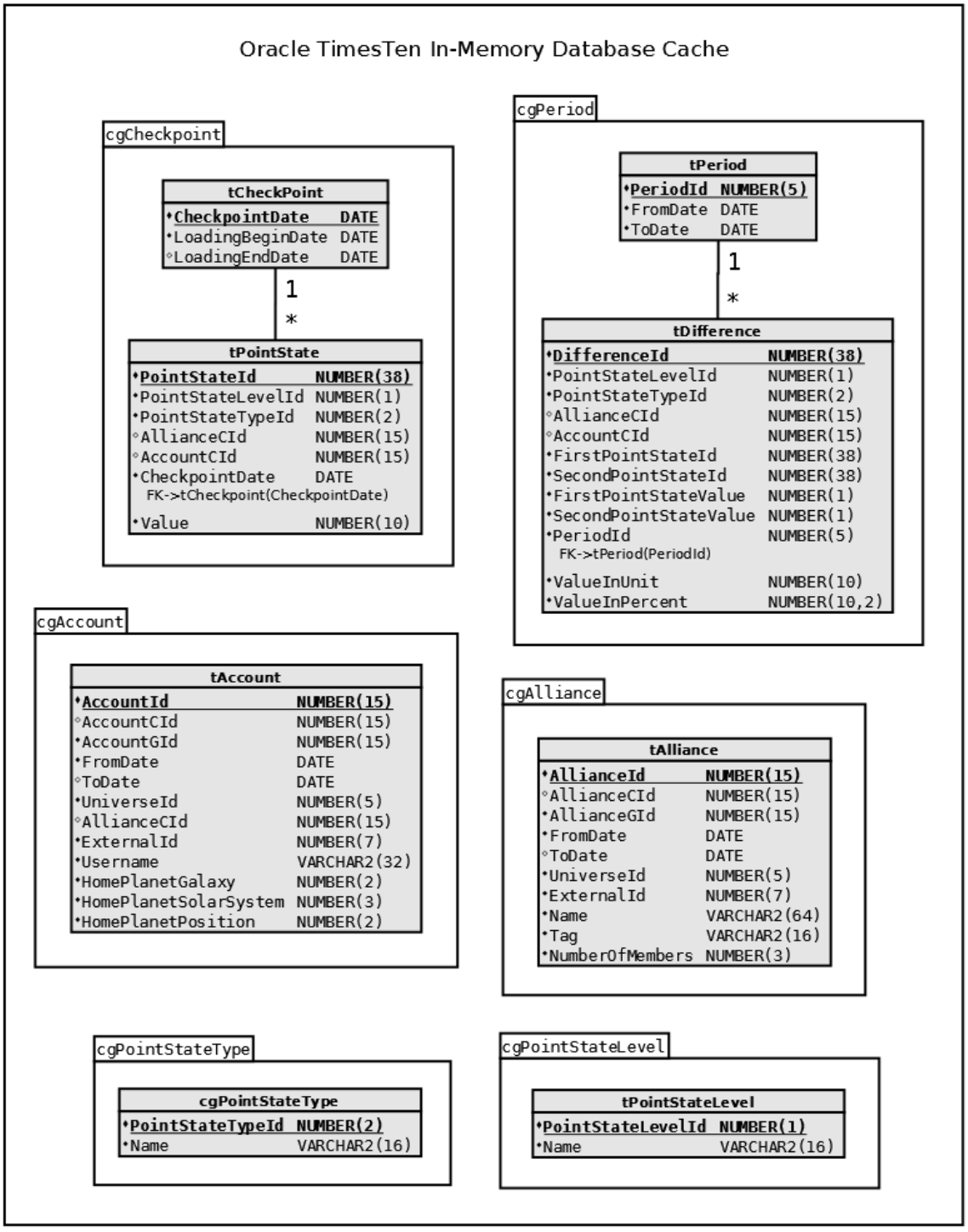
a) Rodzaj raportu,
b) Użycie IMDB Cache;

Rodzaj raportu określa czy został od wstępnie policzony przez harmonogram zadań i znajduje się już w bazie danych czy też nie. Standardowy raport pokazuje dane dla jednego z charakterystycznych okresów tj. tygodnia, miesiąca, kwartału, półrocza lub roku. Może on zostać w łatwy sposób zwrócony z tabeli *tDifference*, która zawiera dane w postaci końcowej. Raport zdefiniowany dla niestandardowego przedziału czasu, wymaga przeprowadzenia obliczeń w locie.

4.4. Modyfikacje aplikacji

4.4.1. Grupy pamięci podręcznej

Zidentyfikowano, że istnieje w aplikacji łącznie 8 relacji, które są intensywnie wykorzystywane we wszystkich operacjach wykonywanych na bazie. Zaprojektowano i zaimplementowano organizację buforu w *Oracle TimesTen*, jak przedstawiono na rysunku 8. Z użyciem zaproponowanego schematu pamięci podręcznej, zostaną wyeliminowane wszystkie operacje dyskowe podczas przetwarzania i nie będzie konieczności delegowania instrukcji do *Oracle Database*.



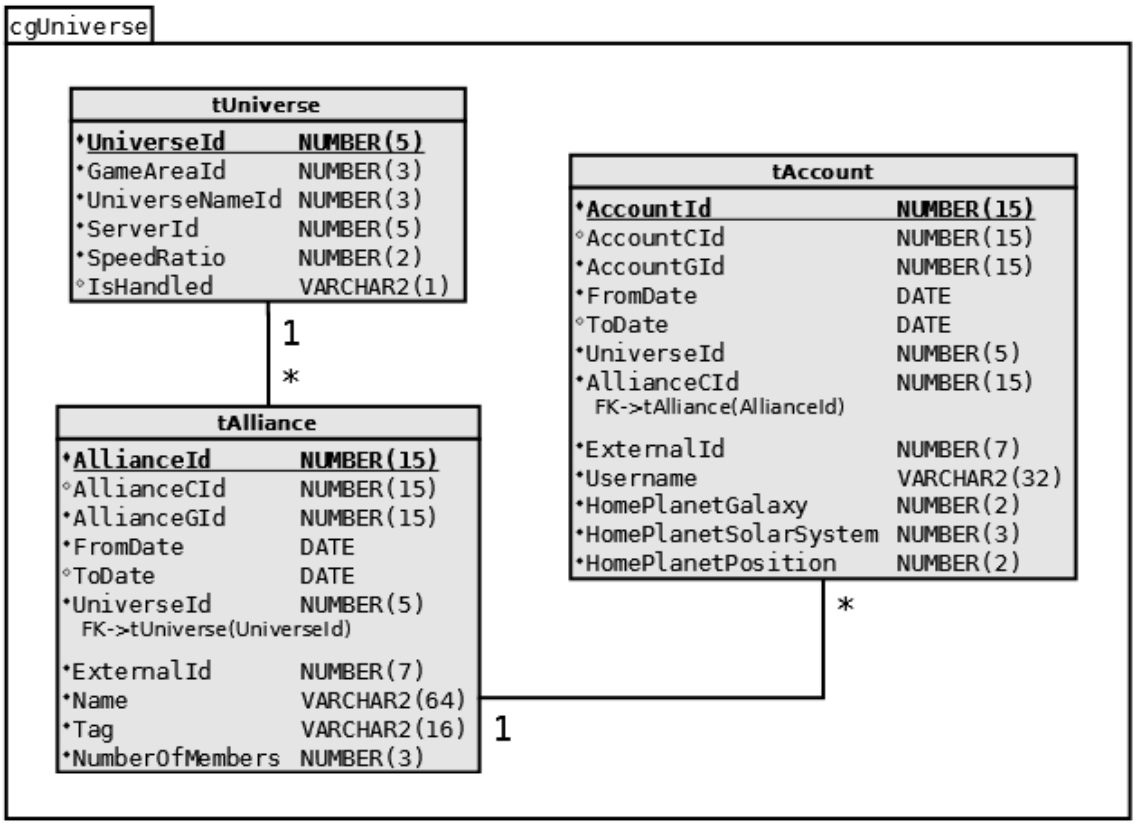
Rysunek 8. Organizacja grup pamięci podręcznej w testowanej aplikacji.

Wszystkie grupy pamięci podręcznej są lokalne, ponieważ zdefiniowano samodzielnie uruchomioną instancję bazy danych rezydującej w pamięci. Nie ma konieczności replikacji. Grupy *cgPointStateLevel* i *cgPointStateType* przechowują tabele, których zawartość jest statyczna. Zostały one stworzone, jako tylko do odczytu z automatycznym odświeżeniem co 1440 minut. Wybrano pełną aktualizację, ponieważ zawierają niewielki

wolumin danych. Bufory *cgAlliance* i *cgAccount* mają za zadanie udostępniać odpowiednio tabele *tAlliance* i *tAccount*. Istnieje między nimi relacja zero lub jeden-do-wielu, przez co nie mogły zostać umieszczone w jednej grupie, w zależności rodzic-dziecko. Podczas przetwarzania na grupach wykonywane są operacje *SELECT, UPDATE* i *INSERT*. Typ zdefiniowano, jako asynchronicznie propagowany, statycznie ładowany. Na obecnym etapie eksploatacji oprogramowania, przechowują ilości wierszy mierzone w tysiącach. Grupa pamięci podręcznej *cgCheckpoint*, zawiera tabele *tCheckpoint*, która jest korzeniem, oraz *tPointState*. Kluczem głównym grupy pamięci podręcznej jest data punktu kontrolnego. Dla każdej wartości identyfikatora grupy istnieje w przybliżeniu 30 tysięcy stanów punktowych. Łączna ilość wierszy jest obecnie wyrażona w milionach. W późniejszym etapie eksploatacji bufor powinien zostać ograniczony do przechowywania danych tylko i wyłącznie z ostatniego roku, ze względu na duży rozmiar tabeli *tPointState*. Przeciętnie, roczne stany punktowe zajmują w relacji około 1 GB, co jest akceptowalną wartości w przypadku umieszczania danych w pamięci operacyjnej. Typ *cgCheckpoint* zdefiniowano, jako asynchronicznie propagowany, statycznie ładowany. Grupa *cgPeriod* zawiera tabele *tDifference* oraz *tPeriod*, która jest korzeniem. Jej zadaniem jest udostępnienie w pamięci podręcznej danych, które służą do generowania standardowych raportów. Podczas obliczania statystyk, dane do obu relacji są dodawane. Grupa jest asynchronicznie propagowana, statycznie ładowana.

Istnieją przesłanki, aby tabele *tAlliance* i *tAccount* znajdowały się w jednej grupie pamięci podręcznej, pomimo, że relacja występująca miedzy nimi nie pozwala na to. Relacja zero lub jeden-do-wielu powoduje, że konta, które nie mają przypisanego sojuszu, nie posiadają też własnej instancji pamięci podręcznej, ponieważ nie ma dla nich odpowiadającego wiersza w tabeli-korzeniu. Wynika to z faktu, że konta niezrzeszone mają wartość klucza obcego równą *NULL*. Możliwym rozwiązaniem jest stworzenie nieznaczącego sojuszu, który będzie oznaczał w aplikacji brak zrzeszenia. Pozwoli to na zmianę relacji na wymagany typ jeden-do-wiele. Ponadto, pozwoliłoby to na identyfikowanie instancji pamięci podręcznej kluczem głównym tabeli *tUniverse*, która stałaby się tabelą-korzeniem nowej grupy *cgUniverse*. Opisana koncepcja została zobrazowana na rysunku 9. Przy zaproponowanym rozwiązaniu wszystkie dane historyczne byłyby w logiczny sposób zgrupowane. Koncepcja ta nie została jednak wdrożona w ramach tejże pracy, ze względu na niewielkie znaczenie dla pomiarów

wydajności. Wprowadzenie relacji innego typu oznaczałoby zmianę założenia, na którym bazuje cała logika aplikacji. Przykład ten przytoczono, aby uzmysłowić, że w niektórych przypadkach lepsza organizacja grup pamięci podręcznej w IMDB Cache może ponosić za sobą pewne zmiany w oryginalnej aplikacji lub wpływać na architekturę implementowanego oprogramowania. Inny sposób organizacji buforu może przełożyć się na efektywniejsze działanie mechanizmu odświeżania, co nie jest przedmiotem badań.



Rysunek 9. Hipotetyczna grupa pamięci cgUniverse po zmianie założeń aplikacji.

4.4.2. Logika działania

Dostosowano aplikację do wykonywania operacji z użyciem IMDB Cache. W zależności od trybu pracy oprogramowanie do masowego ładowania danych do bazy, łączy się do instancji *Oracle TimesTen* lub *Oracle Database*. Odpowiada za to drugi parametr wiersza poleceń, który może przyjąć dwie wartości:

a) ORATT,
b) ORADB;

Gdy aplikacja do masowego ładowania danych zostanie uruchomiona z wartością *ORATT*, nastąpi połączenie z instancją pamięci podręcznej *Oracle TimesTen* i wszystkie instrukcje będą wykonywane w kontekście buforu, który zostanie potem asynchronicznie spropagowany do *Oracle Database*. W przypadku wartości *ORADB* wystąpi konwencjonalny tryb działania, w którym wszystkie operacje są wykonywane w kontekście *Oracle Database*. Aby wprowadzone zmiany stały się dostępne w pamięci podręcznej, należy wykonać pełne odświeżenie instancji *cgAccount*, *cgAlliance* i *cgCheckpoint*.

Za sposób załadowania danych do bazy odpowiada procesor, który wywołuje odpowiednią sekwencję instrukcji PL/SQL i sprawdza integralność bazy danych po wprowadzeniu zmian. Jeśli wszystko przebiegło pomyślnie, transakcja jest zatwierdzona. W przeciwnym przypadku, następuje wycofanie i podniesienie wyjątku. Dane przesyłane do systemu zarządzania bazą danych są zgrupowane i przekazywane poprzez kolekcję w jednym wykonaniu PL/SQL. Na potrzeby pomiarów wydajności stworzono implementację procesora, który zamiast przesyłać wszystkie stany punktowe w jednej kolekcji i instrukcji, przekazuje każdy z nich w osobnym wywołaniu PL/SQL. Dodatkowy parametr procesora steruje trybem wykonania. Jeśli wybrano system transakcyjny, po pomyślnym wykonaniu operacji załadowania jednego stanu punktowego następuje zatwierdzenie transakcji. Gdy wykorzystany jest tryb wsadowy, wszystkie instrukcje są wykonywane w pojedynczej transakcji, podobnie jak do tej pory. Druga stworzona implementacja procesora grupuje instrukcje za pomocą interfejsu *JDBC*. Operacje są wykonywane na sam koniec przetwarzania, po jednym przełączeniu kontekstu między maszyną wirtualną języka *JAVA* oraz systemem zarządzania bazą danych. W przypadku zgrupowania instrukcji, nie ma sensu wykonywanie każdej z nich w osobnej transakcji, ponieważ wiązałoby się to z umieszczeniem komendy *COMMIT* bezpośrednio w logice PL/SQL. Jest to ogromne naruszenie zasady transakcyjności, więc nie należy takiego przypadku rozważać. Za podjęcie decyzji o zatwierdzeniu lub wycofaniu transakcji, zawsze powinna odpowiadać aplikacja kliencka, która znajduje się na samym początku łańcucha przetwarzania i jest czynnikiem sprawczym.

Dostosowania wymagała też logika zaimplementowana w języku PL/SQL. Kod aplikacji testowej, działający bezpośrednio w bazie danych, został podzielony na dwie niezależne gałęzie. W pierwszej, pakiety PL/SQL, zaimplementowane w ramach pracy

inżynierskiej [1], używają buforowania natywnego w celu zwiększenia wydajności. Podczas uruchomienia logiki wszystkie wymagane dane są wczytywane do RAM, w celu eliminacji przełączania kontekstu między mechanizmami przetwarzania PL/SQL i SQL. Przed wykonaniem instrukcji DML dane są w całości przetwarzane w pamięci i wyniki zapisywane, jako oczekujące. Po zakończeniu obliczeń oczekujące instrukcje są jednorazowo wysłane do systemu zarządzania bazą danych i wykonywane. W drugiej gałęzi, która musiała zostać zaimplementowana w całości na potrzeby testów, występuje konwencjonalny przepływ informacji. Jest on powszechnie stosowany w aplikacjach, które nie przetwarzają ogromnych ilości informacji. Wymagane dane są wczytywane w momencie, gdy istnieje potrzeba użycia ich. Powoduje to ogromny narzut związany z ciągłą komunikacją logiki z bazą danych. Operacje DML nie są zapisywane w pamięci operacyjnej, jako oczekujące, tylko od razu wysyłane do systemu zarządzania bazą danych, w celu natychmiastowego wykonania.

Podczas wdrażania logiki w języku PL/SQL okazało się, że *Oracle TimesTen* nie pozwala na tworzenie typów obiektowych na poziomie schematu [13], z których aktualnie aplikacja korzysta w celu grupowania danych i przesyłania w formie kolekcji do systemu zarządzania bazą danych. W rezultacie, nie można w interfejsie *JDBC* skorzystać z typów *STRUCT* i *ARRAY*. Istnieje jedynie możliwość przesyłania do bazy tablic asocjacyjnych indeksowanych za pomocą liczby całkowitej, której elementy są wyłącznie ciągami znaków lub liczbami [28]. Jedyną alternatywą pozostaje przesyłanie do bazy strumienia instrukcji, w której każda z nich będzie ładować do bazy jeden stan punktowy. Ponadto, testy dla wariantów z ustawionym grupowaniem instrukcji nie są możliwe do wykonania, ze względu na błąd w implementacji interfejsu *JDBC*, który powoduje, że instrukcja *addBatch* nie działa poprawnie [29]. Problem zostanie naprawiony w jednym z następnych wydań sterownika, ale nie jest określony czas, w jakim zostanie to zrealizowane [29].

5. Pomiary wydajności

Niniejszy rozdział prezentuje specyfikację zaprojektowanych testów, które powstały z wykorzystaniem informacji o aplikacji, zawartych w rozdziale 4. Wykazuje słuszność wykorzystanych scenariuszy poprzez analogię do oprogramowania, którego głównym celem jest operacyjne wykonywanie podobnych czynności. Przedstawia wyniki pomiarów oraz wnioski płynące z porównania mechanizmów przetwarzania w pamięci z konwencjonalną instancją *Oracle Database*.

5.1. Konwencja numeracji

Każdy wykonany pomiar posiada unikalny numer identyfikacyjny, który składa się z trzech liczb, oddzielonych kropkami. Sposób numeracji służy do klasyfikacji testów i grupuje pomiary względem, których są liczone przyśpieszenia. Liczby w numerze pomiaru oznaczają kolejno następujące właściwości:

a) Numer testu,
b) Użycie buforu PL/SQL,
c) Wykorzystanie IMDB Cache;

Testy są numerowane kolejnymi liczbami od 1 do n. Dla pomiarów, które nie używały buforu PL/SQL, drugą liczbą jest 1. W przeciwnym przypadku, pomiar jest identyfikowany liczbą 2. Podobnie przyjęto dla wykorzystania IMDB Cache. Jeśli dany pomiar był wykonany bez pamięci podręcznej to trzecią liczbą jest 1. W przeciwnym przypadku, pomiar oznaczono liczbą 2.

5.2. Testy

5.2.1. Ładowanie danych do bazy

Ładowanie danych do bazy posiada najwięcej możliwych wariantów realizacji i zostanie przetestowane najintensywniej spośród wszystkich rozważanych operacji. W tabeli 7 przedstawiono szczegółowo zaprojektowane testy dla systemu transakcyjnego. Charakteryzują się one uruchomieniem wielu niezależnych transakcji, o niewielkim zasięgu i ilości operacji oraz krótkim czasem wykonania. Pomimo, że w osobnych transakcjach ładowane będą stany punktowe, to można znaleźć analogię w działaniu do

systemu internetowej sprzedaży. Załadowanie każdego stanu punktowego może oznaczać zlecane zamówienie, które należy zarejestrować w bazie danych. Opisane wykorzystanie systemu bazodanowego jest powszechnie realizowane od samego początku istnienia definicji transakcyjności.

Nr pomiaru	Zgrupowane dane/instrukcje	Wiele transakcji	Użyty bufor PL/SQL	Użyty IMDB Cache
1.1.1	Nie	Tak	Nie	Nie
1.1.2	Nie	Tak	Nie	Tak
1.2.1	Nie	Tak	Tak	Nie
1.2.2	Nie	Tak	Tak	Tak

Tabela 7. Zaplanowane warianty ładowania danych do bazy dla systemu transakcyjnego.

Scenariusze dla wsadowego zasilenia hurtowni zostały przedstawione w tabeli 8. Są one zawsze realizowane w jednej transakcji, której zakres jest globalny. Załadowanie danych nastąpi w całości poprawnie lub w ogóle, w przypadku błędu. Zasilenie hurtowni wykonuje się okresowo w celu aktualizacji danych w środowisku analitycznym, zazwyczaj raz na dobę, w ciągu nocy. Proces zasilenia może być bardzo złożony i czasochłonny. Jeśli czas wykonania nie przekracza jednej doby, ale trwa dłużej niż okno serwisowe uzgodnione w umowie SLA (ang. *Service Level Agreement*), to stosuje się dwa węzły, które są przełączane na zmianę w środowisku produkcyjnym. Gdy zasilenie wykonuje się ponad dobę, to należy zorganizować odpowiednią ilość węzłów, które są wykorzystywane cyklicznie. Hurtownia danych powinna dostarczać najbardziej aktualne dane, jakie są możliwe do osiągnięcia dostępnymi środkami technicznymi. Pomiary 3.1.2, 3.2.2, 4.1.2 i 4.2.2 nie mogą zostać zrealizowane z powodu problemów opisanych w rozdziale 4.4.2. Pozostałe scenariusze dla testu 3 i 4 zostaną wykonane w celach informacyjnych. Zakłada się, że dzięki optymalnym metodom przekazywania danych do *Oracle Database*, których nie wspiera *Oracle TimesTen*, zostanie osiągnięty krótszy czas wykonania niż dla najlepszego rezultatu z użyciem IMDB Cache.

Nr pomiaru	Zgrupowane dane/instrukcje	Wiele transakcji	Użyty bufor PL/SQL	Użyty IMDB Cache	Wykonywalny
2.1.1	Nie	Nie	Nie	Nie	Tak
2.1.2	Nie	Nie	Nie	Tak	Tak
2.2.1	Nie	Nie	Tak	Nie	Tak
2.2.2	Nie	Nie	Tak	Tak	Tak
3.1.1	Dane	Nie	Nie	Nie	Tak
3.1.2	Dane	Nie	Nie	Tak	Nie
3.2.1	Dane	Nie	Tak	Nie	Tak
3.2.2	Dane	Nie	Tak	Tak	Nie
4.1.1	Instrukcje	Nie	Nie	Nie	Tak
4.1.2	Instrukcje	Nie	Nie	Tak	Nie
4.2.1	Instrukcje	Nie	Tak	Nie	Tak
4.2.2	Instrukcje	Nie	Tak	Tak	Nie

Tabela 8. Zaplanowane warianty ładowania danych do bazy dla wsadowego zasilenia hurtowni.

5.2.2. Obliczanie statystyk

Obliczanie statystyk opiera się na przetwarzaniu stanów punktowych, załadowanych do bazy danych. Wyniki są zapisywane w tabeli, która jest wykorzystywana przez moduł raportujący, w przypadku generowania standardowych raportów. Technika wstępnej kalkulacji danych analitycznych jest często używana w hurtowniach danych, w celu zwiększenia przepustowości. Informacje, które najczęściej są żądane przez użytkowników, zostają umieszczone w bazie danych w formie końcowej, aby skrócić czas oczekiwania i odciążyć instancję. Formatowanie jest jedyną czynnością, jaką wykonuje się po pobraniu danych a przed wyświetleniem w aplikacji. W tabeli 9 zostały przedstawione warianty, jakie zostały zaplanowane dla operacji obliczania statystyk.

Nr pomiaru	Użyty bufor PL/SQL	Użyty IMDB Cache
5.1.1	Nie	Nie
5.1.2	Nie	Tak
5.2.1	Tak	Nie
5.2.2	Tak	Tak

Tabela 9. Zaplanowane warianty obliczania statystyk.

5.2.3. Generowanie raportów

W tabeli 10 pokazano warianty testów dla raportowania. Raporty są generowane w celu otrzymania żądanych informacji. Dane powinny być aktualne i posiadać odpowiednią postać tak, aby miały wartość analityczną i wymagały jak najmniej przetwarzania ręcznego, w celu uzyskania istotnych informacji. Raporty wykorzystuje się podczas podejmowania strategicznych decyzji, dotyczących działalności podmiotu.

Nr pomiaru	Standardowy raport	Pierwsze wykonanie	Użyty IMDB Cache
6.1.1	Nie	Tak	Nie
6.1.2	Nie	Tak	Tak
6.2.1	Nie	Nie	Nie
6.2.2	Nie	Nie	Tak
7.1.1	Tak	Tak	Nie
7.1.2	Tak	Tak	Tak
7.2.1	Tak	Nie	Nie
7.2.2	Tak	Nie	Tak

Tabela 10. Zaplanowane warianty generowania raportów.

5.3. Wyniki

W tabeli 11 zostały przedstawione wyniki dla testów, które nie mogły zostać w pełni zrealizowane. Zaprezentowane scenariusze zostały wykonane tylko i wyłącznie z wykorzystaniem instancji *Oracle Database* i jej pełnych możliwości, których nie wspiera *Oracle TimesTen*.

Nr pomiaru	Zgrupowane dane/instrukcje	Wiele transakcji	Użyty bufor PL/SQL	Czas wykonania [min]	Przyśpieszenie z buforem PL/SQL
3.1.1	Dane	Nie	Nie	1,72	**1,54**
3.2.1	Dane	Nie	Tak	1,12	
4.1.1	Instrukcje	Nie	Nie	3,20	**1,37**
4.2.1	Instrukcje	Nie	Tak	2,33	

Tabela 11. Pomiary wydajności wykonane tylko dla Oracle Database.

W tabelach 12, 13 i 14 zestawiono wszystkie pomiary, wykonane w ramach testów w pełni zrealizowanych. Obliczono przyśpieszenia różnych konfiguracji względem siebie, w celu porównania wydajności.

Nr pomiaru	Zgrupowane dane/instrukcje	Wiele transakcji	Użyty bufor PL/SQL	Użyty IMDB Cache	Czas wykonania [min]	Przyśpieszenie z IMDB Cache	Przyśpieszenie z buforem PL/SQL	Przyśpieszenie Oracle Database z IMDB Cache względem Oracle Database z buforem PL/SQL
System transakcyjny								
1.1.1	Nie	Tak	Nie	Nie	33,70	**8,99**	Dla Oracle Database	**5,02**
1.1.2	Nie	Tak	Nie	Tak	3,75		**1,79**	
1.2.1	Nie	Tak	Tak	Nie	18,82	**6,27**	Dla Oracle Database z IMDB Cache	
1.2.2	Nie	Tak	Tak	Tak	3,00		**1,25**	
Wsadowe zasilenie hurtowni danych								
2.1.1	Nie	Nie	Nie	Nie	3,60	**1,38**	Dla Oracle Database	**1,02**
2.1.2	Nie	Nie	Nie	Tak	2,62		**1,35**	
2.2.1	Nie	Nie	Tak	Nie	2,67	**1,08**	Dla Oracle Database z IMDB Cache	
2.2.2	Nie	Nie	Tak	Tak	2,47		**1,06**	

Tabela 12. Pomiary wydajności ładowania danych do bazy.

Nr pomiaru	Użyty bufor PL/SQL	Użyty IMDB Cache	Czas wykonania [s]	Przyśpieszenie z IMDB Cache	Przyśpieszenie z buforem PL/SQL	Przyśpieszenie Oracle Database z IMDB Cache względem Oracle Database z buforem PL/SQL
5.1.1	Nie	Nie	5,67	**1,40**	Dla Oracle Database	**1,22**
5.1.2	Nie	Tak	4,06		**1,14**	
5.2.1	Tak	Nie	4,96	**4,59**	Dla Oracle Database z IMDB Cache	
5.2.2	Tak	Tak	1,08		**3,76**	

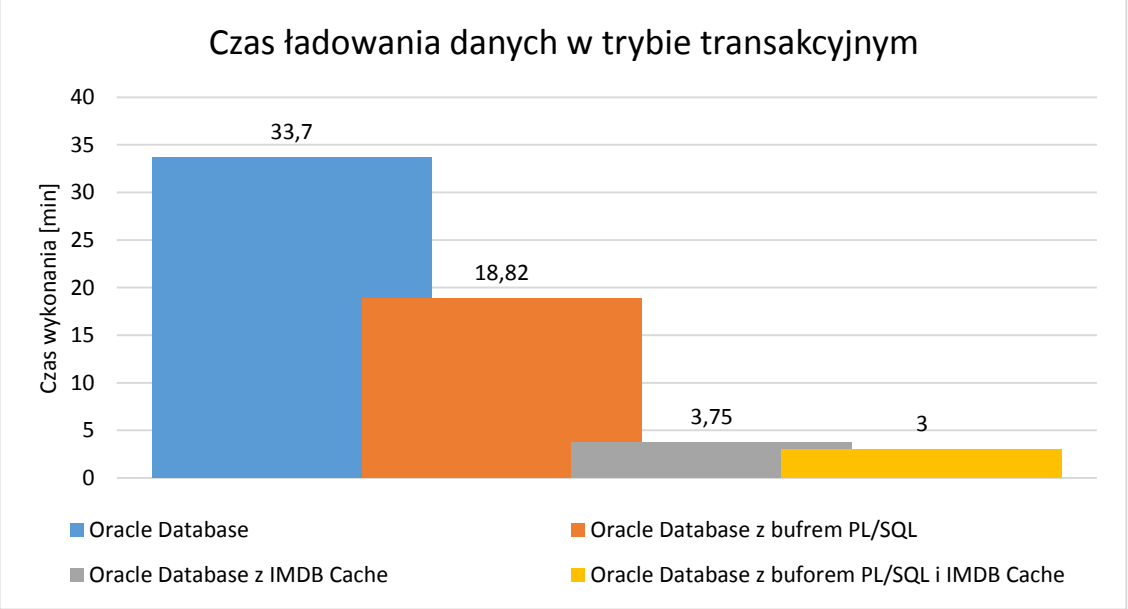
Tabela 13. Pomiary wydajności obliczania statystyk.

Nr pomiaru	Standardowy raport	Pierwsze wykonanie	Użyty IMDB Cache	Czas wykonania [ms]	Przyśpieszenie z IMDB Cache	Przyśpieszenie między kolejnym a pierwszym wykonaniem
6.1.1	Nie	Tak	Nie	111	**15,86**	Dla Oracle Database
6.1.2	Nie	Tak	Tak	7		**7,40**
6.2.1	Nie	Nie	Nie	15	**3,75**	Dla Oracle Database z IMDB Cache
6.2.2	Nie	Nie	Tak	4		**1,75**
7.1.1	Tak	Tak	Nie	70	**10,00**	Dla Oracle Database
7.1.2	Tak	Tak	Tak	7		**10,00**
7.2.1	Tak	Nie	Nie	7	**2,33**	Dla Oracle Database z IMDB Cache
7.2.2	Tak	Nie	Tak	3		**2,33**

Tabela 14. Pomiary wydajności generowania raportów.

5.4. Wnioski

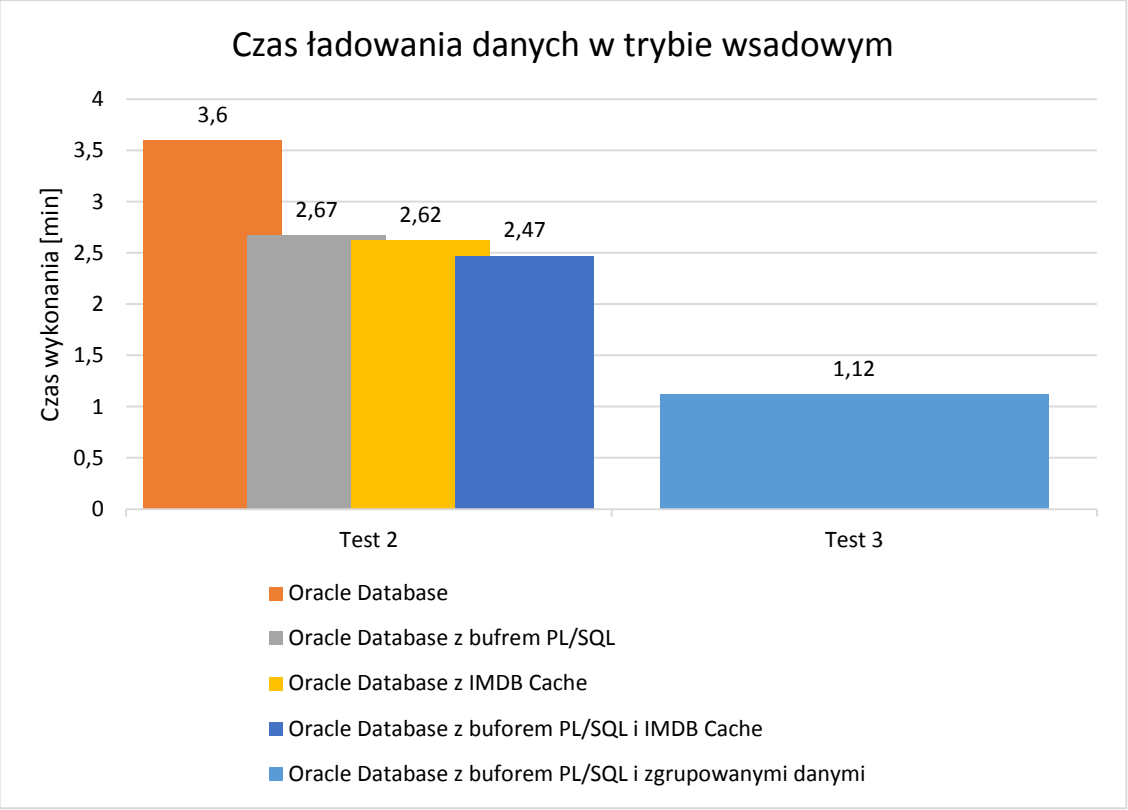
Z rysunku 10, który przedstawia pomiary dla pierwszego testu, wynika, że podczas operacji ładowania danych w trybie transakcyjnym najwolniejszy czas wykonania osiągnęła standardowa instancja *Oracle Database*. Dane zostały najszybciej załadowane w konfiguracji korzystającej z pamięci podręcznej i buforu PL/SQL. Różnica między skrajnymi pomiarami jest drastyczna. W przypadku systemów transakcyjnych, wykorzystanie IMDB Cache jest uzasadnione, a dla mocno obciążonych konieczne. Uzyskany rezultat potwierdza rzetelność testu wzorcowego, opracowanego i wykonanego przez *Oracle Corporation*, w którym odnotowano ponad 7-krotny wzrost przepustowości systemu transakcyjnego [12]. Różnica między dwoma rezultatami, uzyskanymi niezależnie, wynika z innej specyfiki aplikacji testowej oraz starszej wersji systemu zarządzania bazą danych. Producent zastosował w pomiarach *Oracle Database 10g*.



Rysunek 10. Wykres czasu ładowania danych w trybie transakcyjnym.

Dla drugiego testu, który realizował zasilenie hurtowni danych, pomiary zostały zaprezentowane na rysunku 11. Wskazują one, że *Oracle Database* okazał się najwolniejszy a baza danych z pamięcią podręczną i buforowaniem PL/SQL najszybsza, podobnie jak dla systemu transakcyjnego. Zasadniczą różnicą jest jednak zdecydowanie mniejsze skrócenie czasu wykonania pomiędzy kolejnymi testowanymi konfiguracjami. Szczególną uwagę należy zwrócić na fakt, że tak jak zakładano, najszybszy czas

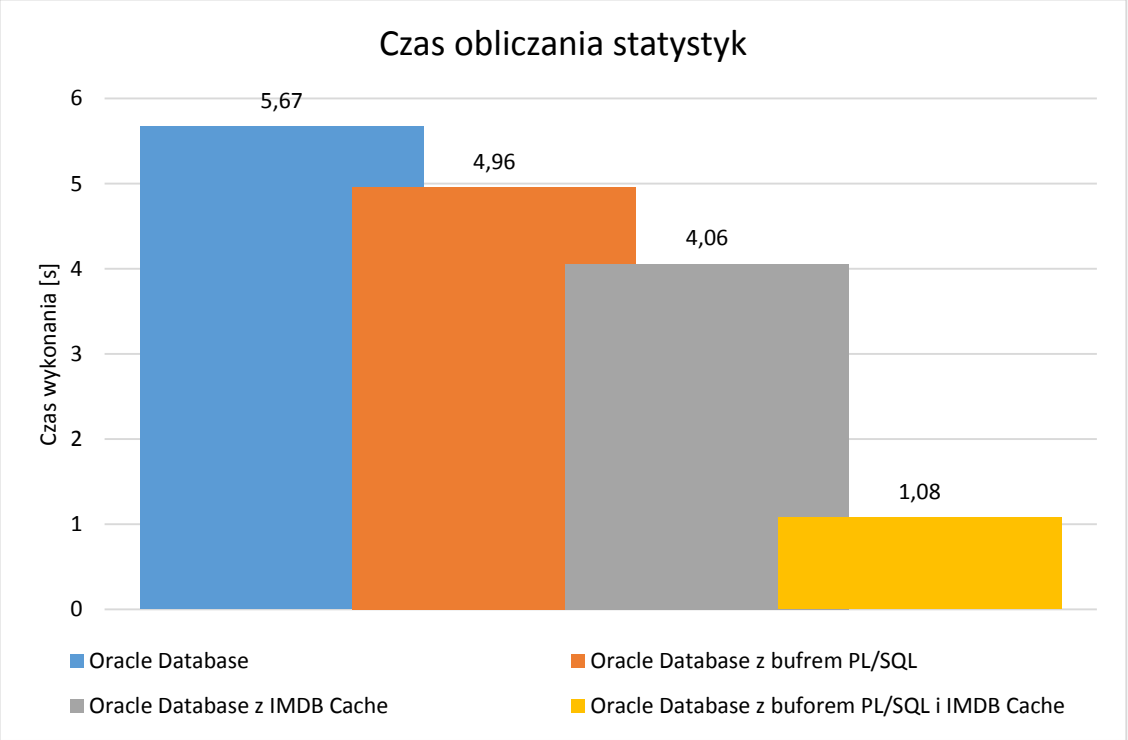
wykonania dla operacji wsadowego zasilenia hurtowni danych został osiągnięty bez użycia IMDB Cache, w jednym z pomiarów dokonanych w ramach testu trzeciego. Został on przedstawiony na rysunku 11, po prawej stronie. Ponad dwukrotnie szybciej, od najkrótszego czasu wykonania z IMDB Cache, spisała się instancja *Oracle Database* z buforem PL/SQL, do której dane zostały przesłane w formie kolekcji z kontekstu maszyny wirtualnej języka *JAVA*. Wariant ten nie był możliwy do realizacji wykorzystaniem bazy danych *Oracle TimesTen*, podobnie jak scenariusz z grupowaniem instrukcji. W rezultacie, dla testu trzeciego i czwartego nie zostało przedstawione pełne zestawienie wyników.



Rysunek 11. Wykres czasu ładowania danych w trybie wsadowym.

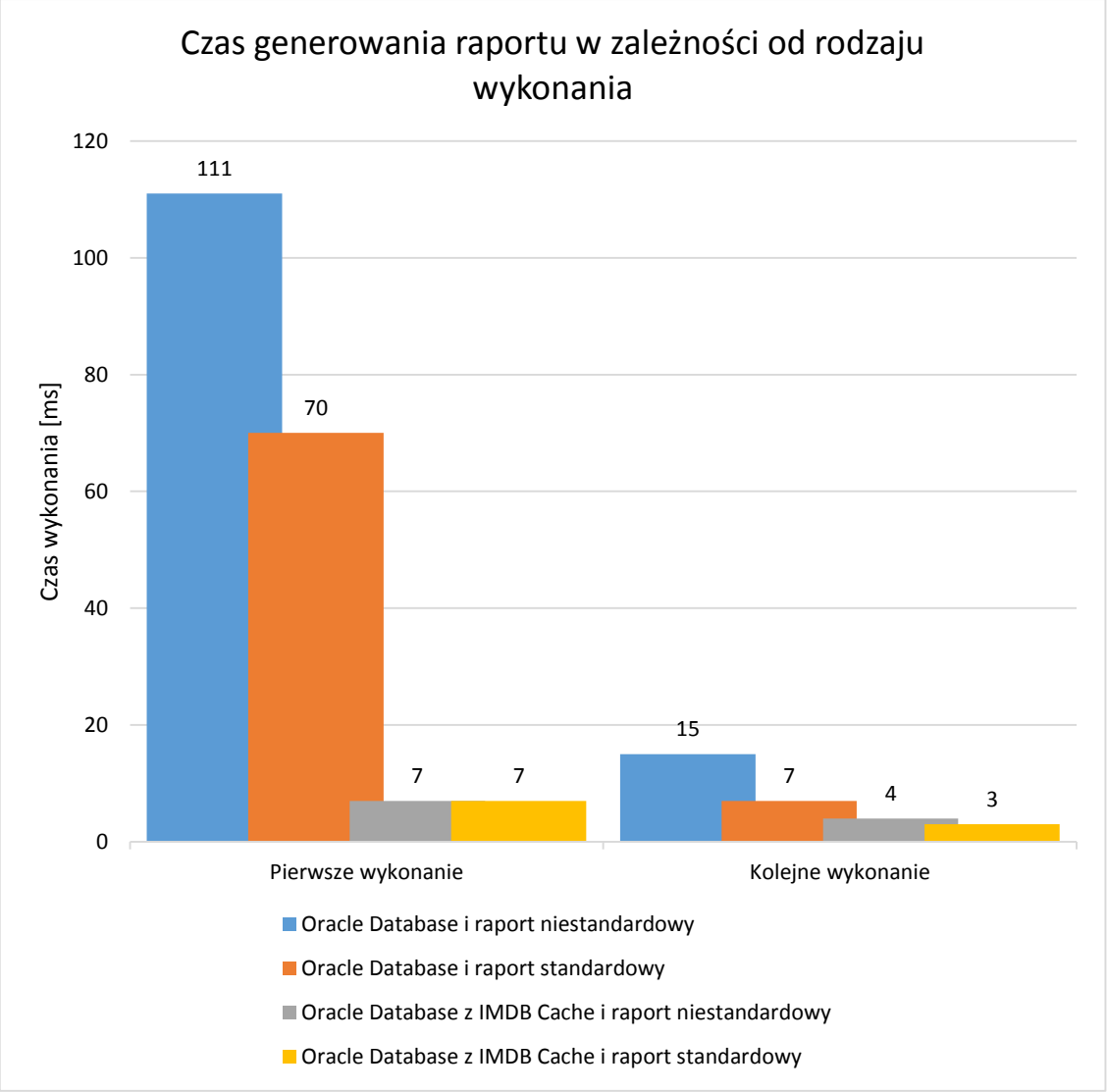
Z wykorzystaniem piątego testu, zmierzono czas obliczania statystyk dla stanów punktowych i zapisania rezultatu w tabeli. Ze stworzonych w ten sposób danych można generować standardowe raporty, ponosząc mniejszy koszt niż podczas dokonywania obliczeń w locie. Wyniki pomiarów zostały przedstawione na rysunku 12. Najwolniej obliczenia trwały na samodzielnie uruchomionej instancji *Oracle Database* a najkrócej z użyciem konfiguracji korzystającej z IMDB Cache i buforowania PL/SQL. W przypadku zastosowania samego mechanizmu IMDB Cache, nie uzyskano znaczącego

skrócenia czasu wykonania. Dodanie buforu PL/SQL do konfiguracji z pamięcią podręczną, spowodowało prawie 4-krotne przyśpieszenie.



Rysunek 12. Wykres czasu obliczania statystyk.

W teście szóstym i siódmym zbadano czas odpowiedzi dla generowania raportów standardowych i niestandardowych. Raport standardowy jest wstępnie policzony i zapisany w bazie danych za pomocą mechanizmu obliczania statystyk, natomiast niestandardowy jest uzyskiwany w locie. Wyniki zostały pokazane na rysunku 13. Dla pierwszego wykonania raportu niezapisanego w bazie, standardowa instancja *Oracle Database* okazała się kilkanaście razy wolniejsza od bazy danych z pamięcią podręczną. Jest to oczekiwany rezultat, ponieważ system zarządzania bazą danych w pamięci masowej poniósł wysoki koszt dostępu do danych znajdujących się na dysku. W przypadku kolejnego żądania raportu niestandardowego, czas wykonania w instancji *Oracle Database* drastycznie spadł, ale i tak był ponad trzykrotnie wyższy niż w przypadku bazy danych z IMDB Cache. Skrócenie czasu realizacji jest naturalne, ze względu na to, że raport znajdował się już w pamięci operacyjnej po pierwszym wykonaniu. Baza danych *Oracle Database* nie jest jednak wstanie dorównać mechanizmowi pamięci podręcznej, ze względu na założenia dotyczące dostępu do danych, jakimi jest obarczony system zarządzania bazą danych.

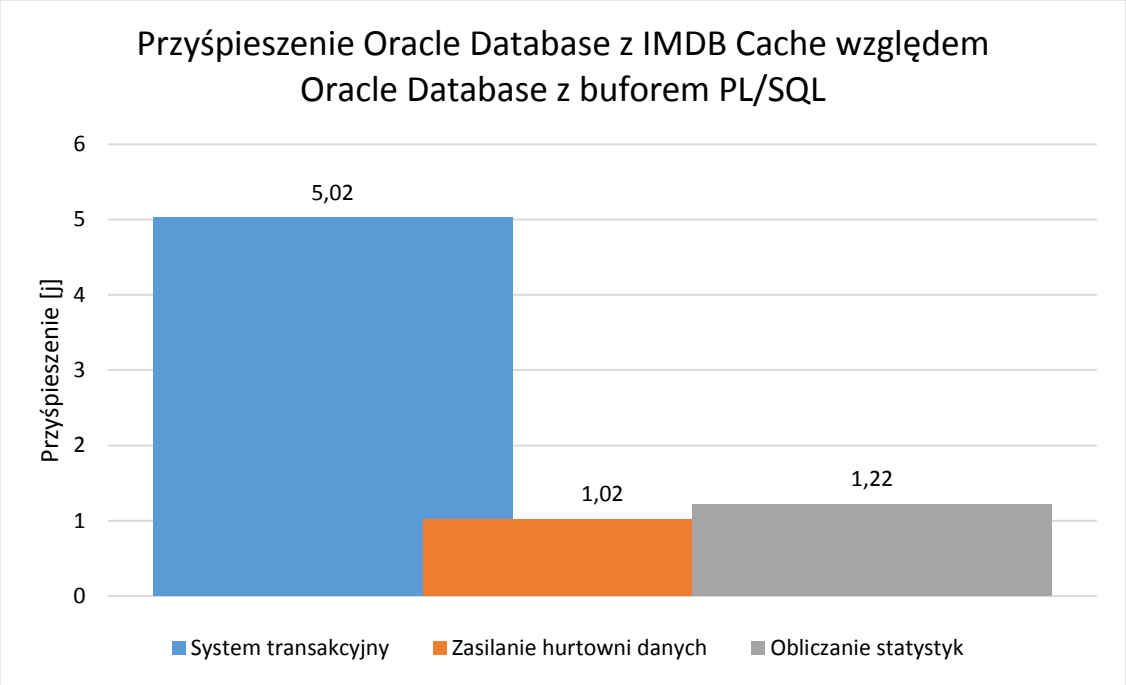


Rysunek 13. Wykres czasu generowania raportu w zależności od rodzaju wykonania.

Wstępne obliczenie raportu daje korzyści w przypadku zwykłej instancji *Oracle Database*. Podczas pierwszego wykonania, czas generowania standardowego raportu był o około jedną trzecią krótszy. Dla kolejnych uruchomień, korzyść była większa i przyśpieszenie wyniosło ponad 2. Gdy raport był generowany z użyciem bazy danych z pamięcią podręczną, nie było różnicy w czasie pierwszego wykonania dla raportu standardowego i niestandardowego. W przypadku kolejnego uruchomienia, różnica była tak niewielka, że przy pomiarach rzędu kilku milisekund może zostać pominięta.

Na rysunku 14 przedstawiono przyśpieszenie *Oracle Database* z IMDB Cache względem instancji bazy danych z buforem PL/SQL, ale bez pamięci podręcznej. Wyróżniono przetestowane operacje, do których wykorzystywane są współcześnie

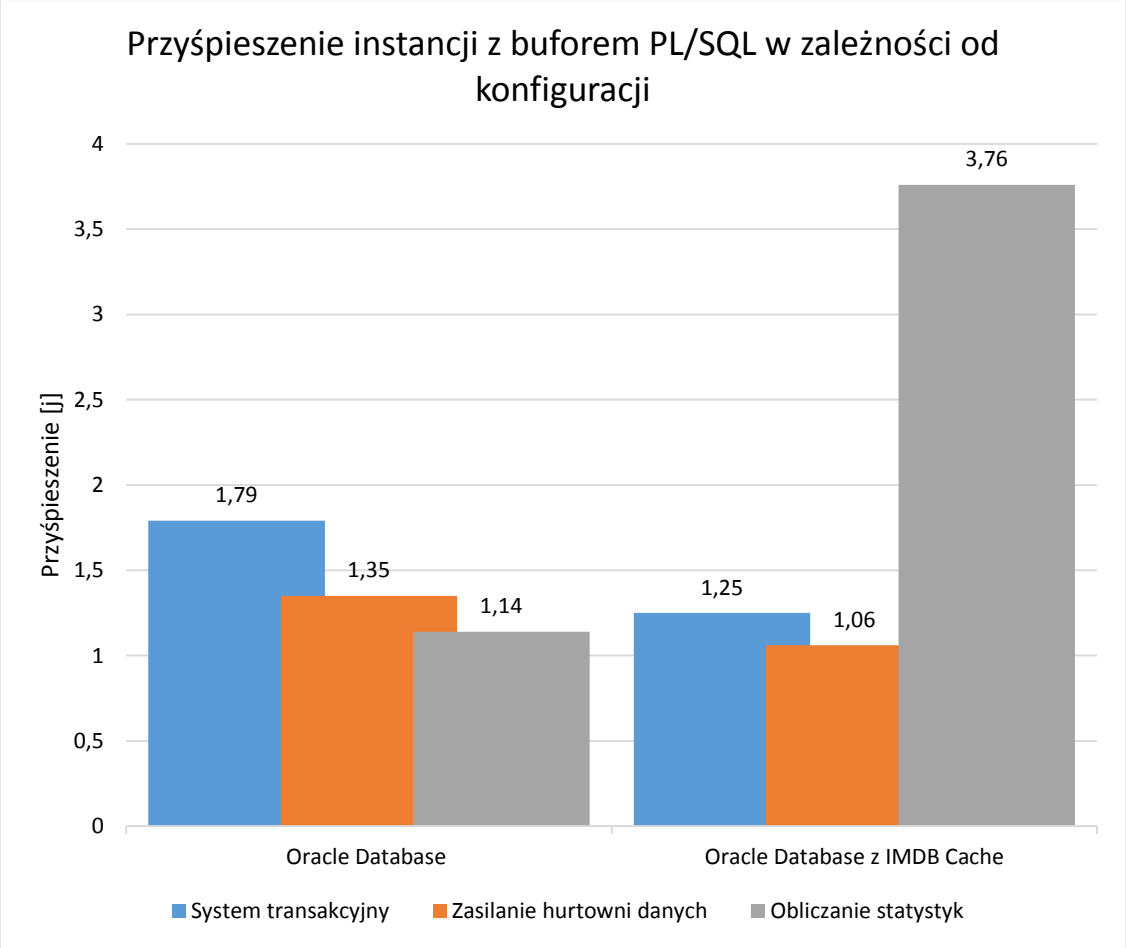
systemy zarządzania bazą danych. Dla wszystkich zastosowań zanotowano korzyści, ale o różnym stopniu. Zestawienie pokazuje, że w przypadku systemu transakcyjnego bardziej opłacalne może być dodanie pamięci podręcznej w warstwie aplikacji, niż próba optymalizacji działania logiki aplikacji. Dla zasilenia hurtowni danych uruchomienie aplikacji z użyciem pamięci podręcznej dało na tyle małe przyśpieszenie, że zwykła instancja bazy danych z logika opartą o bufor PL/SQL nie odbiega wydajnością w znaczący sposób. W rezultacie, dla systemów realizujących wsadowe ładowanie danych z zewnętrznego źródła w formie przetwarzanych plików, korzyści płynące z IMDB Cache są niewielkie. Lepsze rezultaty mogłyby zostać osiągnięte, gdyby zasilenie następowało z innych baz danych. W takim przypadku źródłowe bazy danych mogłyby posiadać IMDB Cache, który byłby wykorzystywany przez zapytania pobierające dane do hurtowni danych. Wyniki w teście szóstym i siódmym dowiodły, że w przypadku wykonywania zapytań IMDB Cache zwiększa znacząco wydajność. Przeciętne przyśpieszenie uzyskano również dla obliczania statystyk, co powoduje, że zastosowanie samej pamięci podręcznej może być niewystarczające w przypadku złożonych obliczeń, wykonywanych w nieoptymalny sposób.



Rysunek 14. Wykres przyśpieszenia Oracle Database z IMDB Cache względem Oracle Database z buforowaniem PL/SQL.

Należy rozważyć połączenie technik IMDB Cache oraz buforowania w PL/SQL, w celu osiągnięcia maksymalnej wydajności. Jak pokazuje rysunek 15, w przypadku obliczania

statystyk znaczący wzrost osiągnięto dopiero po zastosowaniu w instancji *Oracle TimesTen* ulepszonej logiki, która spowodowała niemal czterokrotne przyśpieszenie. Oznacza to, że wysokiej wydajności nie udałoby się uzyskać poprzez zaniechanie optymalizacji oprogramowania. W pozostałych przypadkach dodanie buforu PL/SQL, dla instancji bazy danych z pamięcią podręczną, pozwalało tylko na niewielkie przyśpieszenie. Jest to oczekiwane zachowanie, ponieważ oba mechanizmy bazują na umieszczeniu danych w pamięci operacyjnej. Uzyskane skrócenie czasu wykonania wynika głównie z redukcji kosztu, jaki był ponoszony podczas przełączania kontekstu między mechanizmami przetwarzania SQL i PL/SQL. W przypadku systemu transakcyjnego i standardowej instancji bazy danych, zanotowano drugie w kolejności największe przyśpieszenie przy zastosowaniu gałęzi kodu, która korzysta z buforu PL/SQL. Zoptymalizowana logika pozwoliła na niemal dwukrotne przyśpieszenie przetwarzania, co znacząco zredukowało przewagę *Oracle Database* uruchomioną z IMDB Cache.



Rysunek 15. Wykres przyśpieszenia instancji z buforem PL/SQL w zależności od konfiguracji

6. Zakończenie

Porównano wydajność bazy danych *Oracle Database 12c Enterprise* z włączoną opcją *Oracle In-Memory Database Cache* oraz standardową instancję, działającą bez pamięci podręcznej. Analiza przypadków użycia aplikacji testowej pozwoliła na zaprojektowanie 7 testów, których zadaniem było zasymulowanie różnych środowisk pracy oraz znalezienie mocnych i słabych stron przetwarzania w pamięci z wykorzystaniem innowacyjnej technologii. Jeden test nie był możliwy do zrealizowania z powodu ograniczeń bazy danych *Oracle TimesTen*, która jest częścią składową mechanizmu IMDB Cache. Nie istnieje możliwość definiowania typów obiektowych na poziomie schematu. Kolejny z nich musiał zostać pominięty z powodu błędu w implementacji interfejsu *JDBC*, który uniemożliwiał prawidłowe działanie aplikacji testowej w konfiguracji z IMDB Cache. Ostatecznie, możliwe było wykonanie 5 testów. Uzyskano łącznie 20 pomiarów, ponieważ każdy scenariusz testowy uruchomiono w czterech wariantach:

a) Oracle Database,
b) Oracle Database z IMDB Cache,
c) Oracle Database z buforem PL/SQL,
d) Oracle Database z buforem PL/SQL i IMDB Cache;

We wszystkich zestawieniach instancja *Oracle Database* uruchomiona w trybie z pamięcią podręczną okazała się szybsza niż samodzielna baza danych. W przypadku systemu transakcyjnego, przetwarzanie w pamięci zdeklasowało standardową instancję *Oracle Database*. Przyśpieszenie wyniosło prawie 9. Należy jednak zwrócić uwagę, że logika korzystająca z buforowania na poziomie PL/SQL w standardowej instancji bazy danych zredukowała przyśpieszenie do 5, a więc niemal dwukrotnie. Warto już na etapie implementacji systemów przetwarzających duże ilości informacji zadbać o odpowiednią jakość rozwiązania, poprzez testy wydajnościowe. W przypadku istniejących systemów, optymalizacja jest zazwyczaj nieekonomiczna oraz obarczona dużym ryzykiem. Bardzo dobre wyniki uzyskano również podczas wykonywania zapytań w środowisku analitycznym. Zanotowano prawie 16-krotne przyśpieszenie przy założeniu, że raport został obliczony w locie i nie był żądany wcześniej. W przypadku ponownego generowania tego samego raportu, przyśpieszenie jest zdecydowanie niższe i wyniosło niecałe 4. Skrócenie czasu ponownego wykonania w *Oracle Database* wynika

z obecności wymaganych bloków danych w liście LRU. Wstępnie obliczone raporty w instancji z pamięcią podręczną wykonywały się tak samo szybko, jak generowane w locie. Oznacza to, że niekiedy można zrezygnować z implementacji dodatkowej logiki, której zadaniem jest przygotowanie raportów tak, aby podczas żądania jak najmniej operacji było wykonywanych w bazie danych. Dla zasilenia hurtowni danych, ze źródła zewnętrznego w formie plików, uzyskane przyśpieszenie wyniosło około 1,4 i było na tyle niskie, że użycie buforu PL/SQL niemal całkowicie zniwelowało przewagę mechanizmu IMDB Cache. Rezultat wynika z faktu, że najbardziej kosztownym etapem zasilenia hurtowni danych jest dostarczenie informacji do systemu zarządzania bazą danych. Najkrótszy czas wykonania w tej kategorii uzyskano dla pomiaru w teście, który nie mógł być zrealizowany w całości przez ograniczenia *Oracle TimesTen*. Instancja *Oracle Database*, której wysłano wszystkie dane w formie kolekcji obiektów, okazała się ponad dwa razy szybsza niż najszybszy możliwy do wykonania wariant z użyciem IMDB Cache. Również podczas wykonywania i zapisu wstępnych obliczeń w środowisku OLAP, przyśpieszenie wyniosło zaledwie 1,4, w stosunku do instancji *Oracle Database* z buforem PL/SQL. Wyjątkowo, w tym przypadku uzyskano znaczne skrócenie czasu wykonania przy połączeniu obu przetestowanych mechanizmów przetwarzania w pamięci – buforu PL/SQL i IMDB Cache. Maksymalne przyśpieszenie uzyskane w kategorii obliczeń analitycznych wyniosło ponad 5. Wysokiej wydajności nie udałoby się uzyskać poprzez zaniechanie optymalizacji oprogramowania.

Zakup i wykorzystanie pamięci podręcznej w systemach OLTP, które charakteryzują się wysokim obciążeniem, są uzasadnione. W przypadku środowisk OLAP, znaczący zysk uzyskiwany jest podczas raportowania. Technologia IMDB Cache idealnie sprawuje się, jeśli do bazy kierowanych jest wiele żądań odczytu. Wdrożenie pamięci podręcznej podczas zasilania hurtowni danych z zewnętrznego źródła w formie plików jest nieuzasadnione, ponieważ *Oracle TimesTen* nie wspiera najwydajniejszej opcji dostarczenia danych do systemu zarządzania bazą danych. Nie należy całkowicie przekreślać zastosowania mechanizmu podczas uzupełniania hurtowni danych. Lepsze rezultaty mogłyby zostać osiągnięte, gdyby zasilenie polegało na pobraniu informacji za pomocą zapytań z innych, źródłowych baz danych, które posiadają pamięć podręczną i zapewniają szybki dostęp do danych. Przed podjęciem decyzji należy przeprowadzić dodatkowe testy. W obliczeniach analitycznych, wykorzystanie mechanizmu IMDB

Cache, powinno zostać połączone z optymalizacją kodu, który realizuje logikę. Jednoczesne zastosowanie obu mechanizmów, sprawia, że wprowadzenie pamięci podręcznej jest opłacalne. W innym przypadku, korzyści mogą będą niewielkie, co spowoduje nieekonomiczność zastosowania buforu.

Aby skorzystać z mechanizmu IMDB Cache przeanalizowano logikę aplikacji testowej i zidentyfikowano możliwe obszary zastosowania, tak jak powinno mieć to miejsce w przypadku wdrażania produktu w rzeczywistym oprogramowaniu. W rezultacie, zaprojektowano i zaimplementowano strukturę grup pamięci podręcznej, której zadaniem była całkowita eliminacja komunikacji bezpośredniej z *Oracle Database* podczas przetwarzania. Przeprowadzenie testów wymagało dostosowania aplikacji. Zmodyfikowano narzędzie, zaimplementowane w języku *JAVA SE*, służące do ładowania danych do bazy. Dodano nowe procesory, które przetwarzają dane w różny sposób. Po wprowadzeniu zmian stało się ono zdolne do użycia IMDB Cache oraz sterowania wartościami parametrów, które są rozważane w scenariuszach testowych. Główna logika aplikacji, która jest zaimplementowana w języku PL/SQL i wykonywana na poziomie systemu zarządzania bazą danych, została podzielona na dwie niezależne gałęzie. W obu znajdują się dokładnie te same pakiety, ale z różnymi ciałami. Pierwsza z nich zawiera zoptymalizowany kod, korzystający z natywnego buforowania opartego na wiązaniu masowym i strukturach pamięciowych. Druga, została stworzona specjalnie na potrzeby testów. Bazuje ona na konwencjonalnym przepływie informacji, w którym wszystkie instrukcje SQL są wykonywane niezależnie.

7. Bibliografia

[1] J. Czyrzewski, Nowe cechy, funkcje oraz mechanizmy dostępne w Oracle Database 10g i 11g, Kraków, 2012.

[2] Yellowfin Corporation, „In-Memory Analytics," [Online]. Dostępny w Internecie: http://www.yellowfinbi.com/Document.i4?DocumentId=104879. [Data uzyskania dostępu: 10 sierpnia 2014].

[3] B. Howarth, „In-memory computing," [Online]. Dostępny w Internecie: http://www.cio.com.au/article/373945/in-memory_computing/. [Data uzyskania dostępu: 3 sierpnia 2014].

[4] G. Loh i in., „A Processing-in-Memory Taxonomy and a Case for Studying Fixed-function PIM," [Online]. Dostępny w Internecie: http://www.cs.utah.edu/wondp/wondp2013-paper2-final.pdf. [Data uzyskania dostępu: 10 sierpnia 2014].

[5] H. Garcia-Molina i K. Salem, „Main Memory Database Systems - An Overview," *IEEE Transactions on Knowledge and Data Engineering,* tom 4, nr 6, pp. 509-516, 1992.

[6] probins (pseud.), „How does an SSD write?," Styczeń 2012. [Online]. Dostępny w Internecie: http://www.blog.solidstatediskshop.com/2012/how-does-an-ssd-write/. [Data uzyskania dostępu: 11 sierpnia 2014].

[7] A. Drozdek, C++. Algorytmy i struktury danych, Helion, 2004.

[8] D. Tow, SQL. Optymalizacja, Helion, 2004.

[9] Oracle Corporation, „Oracle TimesTen Application-Tier Database Cache Introduction," Czerwiec 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/cd/E21901_01/doc/timesten.1122/e21631.pdf. [Data uzyskania dostępu: 14 lipca 2014].

[10] Wikipedia, The Free Encyclopedia, „List of in-memory databases,” [Online]. Dostępny w Internecie: http://en.wikipedia.org/wiki/List_of_in-memory_databases. [Data uzyskania dostępu: 10 sierpnia 2014].

[11] M. Morzy, „Oracle TimesTen – cała baza danych w pamięci operacyjnej,” w *XV Konferencja PLOUG*, Kościelisko, 2009.

[12] Oracle Corporation, „Using Oracle In-Memory Database Cache to Accelerate the Oracle Database,” Lipiec 2009. [Online]. Dostępny w Internecie: http://www.oracle.com/technetwork/database/performance/wp-imdb-cache-130299.pdf. [Data uzyskania dostępu: 4 sierpnia 2014].

[13] Oracle Corporation, „Oracle TimesTen In-Memory Database PL/SQL Developer's Guide,” Czerwiec 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/cd/E21901_01/doc/timesten.1122/e21639.pdf. [Data uzyskania dostępu: 12 sierpnia 2014].

[14] Oracle Corporation, „Oracle TimesTen In-Memory Database Replication Guide,” Czerwiec 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/cd/E21901_01/doc/timesten.1122/e21635.pdf. [Data uzyskania dostępu: 14 sierpnia 2014].

[15] Oracle Corporation, „Oracle TimesTen Application-Tier Database Cache User's Guide 11g Release 2 (11.2.2),” Czerwiec 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/cd/E21901_01/doc/timesten.1122/e21634.pdf. [Data uzyskania dostępu: 15 sierpnia 2014].

[16] Oracle Corporation, „Oracle In-Memory Database Cache Overview,” [Online]. Dostępny w Internecie: http://www.oracle.com/technetwork/products/timesten/overview/timesten-imdb-cache-101293.html. [Data uzyskania dostępu: 15 sierpnia 2014].

[17] Oracle Corporation, „Oracle TimesTen In-Memory Database SQL Reference 11g Release 2 (11.2.2),” Czerwiec 2014. [Online]. Dostępny w Internecie:

http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21642.pdf. [Data uzyskania dostępu: 8 sieprnia 2014].

[18] Oracle Corporation, „Oracle Database SQL Language Reference 12c Release 1 (12.1),” Lipiec 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/database/121/SQLRF/E41329-09.pdf. [Data uzyskania dostępu: 6 sierpnia 2014].

[19] Oracle Corporation, „Oracle Database PL/SQL Language Reference 12c Release 1 (12.1),” Lipiec 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/database/121/LNPLS/E50727-04.pdf. [Data uzyskania dostępu: 17 sierpnia 2014].

[20] Tim (pseud.), „Bulk Binds (BULK COLLECT & FORALL) and Record Processing in Oracle,” [Online]. Dostępny w Internecie: http://www.oracle-base.com/articles/9i/bulk-binds-and-record-processing-9i.php. [Data uzyskania dostępu: 12 sierpnia 2014].

[21] M. Klaene, „Oracle Programming with PL/SQL Collections,” Lipiec 2004. [Online]. Dostępny w Internecie: http://www.developer.com/db/article.php/3379271/Oracle-Programming-with-PLSQL-Collections.htm. [Data uzyskania dostępu: 15 sierpnia 2014].

[22] Oracle Corporation, „How to Install Oracle TimesTen 11g Release 2,” [Online]. [Data uzyskania dostępu: 6 sierpnia 2014].

[23] Oracle Corporation, „Configuring In-Memory Database Cache,” [Online]. Dostępny w Internecie: http://download.oracle.com/otn_hosted_doc/timesten/1122/quickstart/html/admin/imdb_cache.html. [Data uzyskania dostępu: 6 sierpnia 2014].

[24] W. Edward, Oracle Database 10g. Administracja bazy danych w Linuksie, Helion, 2007.

[25] Oracle Corporation, „Oracle Database Installation Guide, 12c Release 1 (12.1) for Linux,” Lipiec 2013. [Online]. Dostępny w Internecie: http://docs.oracle.com/database/121/LADBI/E41491-06.pdf. [Data uzyskania dostępu: 14 lipca 2014].

[26] Oracle Corporation, „Oracle TimesTen In-Memory Database Reference Release 11.2.1,” Czerwiec 2012. [Online]. Dostępny w Internecie: http://docs.oracle.com/cd/E13085_01/doc/timesten.1121/e13069.pdf. [Data uzyskania dostępu: 1 września 2014].

[27] Oracle Corporation, „Oracle Database Reference 12c Release 1 (12.1),” Sierpień 2014. [Online]. Dostępny w Internecie: http://docs.oracle.com/database/121/REFRN/E41527-12.pdf. [Data uzyskania dostępu: 1 września 2014].

[28] Oracle Corporation, „Oracle TimesTen In-Memory Database Java Developer's Guide 11g Release 2 (11.2.2),” Czerwiec 2014. [Online]. Dostępny w Internecie: http://download.oracle.com/otn_hosted_doc/timesten/1122/doc/timesten.1122/e21638.pdf. [Data uzyskania dostępu: 8 sierpnia 2014].

[29] ilkinesrefli (pseud.) i C. Jenkins, „TimesTen jdbc batch insert bug with CallableStatement. Batch insert dublicates parameters and inserts first parameter n times with CallableStatement.,” Luty 2014. [Online]. Dostępny w Internecie: https://community.oracle.com/thread/3519113. [Data uzyskania dostępu: 20 sierpnia 2014].

8. Dodatek 1 – Struktura bazy danych

Dodatek przedstawia szczegółowo logiczną i fizyczną strukturę bazy danych aplikacji testowej. Opis bazuje na informacjach zawartych w pracy inżynierskiej [1], jednak poddano je aktualizacji, zgodnie ze stanem bieżącym.

8.1. Konwencja nazewnictwa

Obiekty bazodanowe aplikacji, na poziomie schematu użytkownika, posiadają spójne i jednolite nazewnictwo. Podczas ustalania nazw tabel i kolumn, stosowano się do następującego zbioru reguł:

a) Wszystkie nazwy obiektów i atrybutów są w języku angielskim.
b) Słowa w nazwach obiektów i atrybutów tabel nie są oddzielone separatorem. Stosuje się zapis w notacji wielbłądziej.
c) Nazwy tabel zaczynają się od prefiksu „t", a potem następuje właściwa nazwa encji w liczbie pojedynczej.
d) Nazwy pakietów zaczynają się od prefiksu „p", a potem następuje właściwa nazwa będąca rzeczownikiem w liczbie pojedynczej.
e) Nazwy kluczy głównych składają się z nazwy encji w liczbie pojedynczej i postfiksu „Id".
f) Atrybuty służące do budowania historii w tabeli składają się z nazwy encji w liczbie pojedynczej i postfiksu „CId" lub „GId".
g) Kolumny kluczy obcych nazywają się tak samo jak pola kluczy głównych w tabelach, do których odwołują się one.
h) Jeśli wiele kolumn w obrębie jednej tabeli stosuje klucz obcy odnoszący się do tej samej tabeli to wtedy nazwy kolumn są poprzedzone prefiksem odróżniającym np. *FirstCheckpointDate* i *SecondCheckpointDate.*

8.2. Schemat aplikacji

Wszystkie obiekty związane z aplikacją testową zostały umieszczone w schemacie bazodanowym *OIMS*. Strukturę bazy danych można podzielić na cztery fizyczne części, dla których istnieją osobne przestrzenie tabel:

a) Stałą,
b) Ciągle rozbudowywaną i aktualizowaną,
c) Ciągle rozbudowywaną,
d) Raportową;

W skład części stałej wchodzą relacje wymienione w tabeli 15. Obiekty *tGameArea, tUniverseName, tServer* są wykorzystywane, jako źródło konfiguracji dla oprogramowania do masowego przetwarzania i ładowania. Tabela *tUniverse,* za pośrednictwem swojego klucza głównego, jest łącznikiem pomiędzy częścią stałą a resztą bazy danych. Tabele *tPointStateLevel* i *tPointStateType* powstały w procesie normalizacji. We wszystkich relacjach, należących do tejże grupy, zmiany zachodzą bardzo rzadko.

Nazwa	Zawartość
tGameArea	Obszary rozgrywki, mogą to być kraje, kontynenty i areny międzynarodowe.
tUniverseName	Nazwy światów, które są wspólne dla wielu obszarów rozgrywki.
tServer	Dane serwerów, wraz z nazwą gracza i hasłem dla konta, które jest wykorzystywane do gromadzenia stanów punktowych.
tUniverse	Światy, w których flaga *IsHandled*, decyduje czy oprogramowanie ma gromadzić stany punktowe dla danego uniwersum.
tPointStateLevel	Poziomy stanu punktowego, obecnie istnieją stany punktowe na poziomie konta i sojuszu.
tPointStateType	Typy stanów punktowych, obecnie istnieje 8 typów stanów punktowych. Są to punkty ogólne, ekonomiczne, badań, militarne aktualnie zbudowane, militarne zbudowane przez całą grę, militarne stracone przez całą grę, militarne zniszczone przez całą grę oraz honoru.

Tabela 15. Relacje rzadko rozbudowywane i aktualizowane.

Część ciągle rozbudowywana i aktualizowana wzrasta przy każdym masowym ładowaniu stanów punktowych. Relacje należące do tej grupy są przedstawione w tabeli 16. W ich przypadku, naturalną operacją jest też aktualizacja. Parametry inicjalizacyjne

przestrzeni tabel biorą to pod uwagę, aby nie dopuścić do łańcuchowania i migracji wierszy.

Nazwa	Zawartość
tAlliance	Sojusze wraz z pełną historią.
tAccount	Konta graczy wraz z pełną historią.

Tabela 16. Relacje ciągle rozbudowywane i aktualizowane.

Część ciągle rozbudowywana wzrasta przy każdym masowym ładowaniu stanów punktowych. Relacje należące do tej grupy są przedstawione w tabeli 17. Ich wiersze nigdy nie są modyfikowane. Naturalnymi operacjami są tylko instrukcje *INSERT* i *DELETE*. Wiedza ta pozwoliła na efektywniejsze przechowywanie wierszy w blokach, ponieważ nie jest wymagane zarezerwowane wolne miejsce na potrzeby przyszłych aktualizacji. Pełny odczyt tabeli *tPointState* jest mniej kosztowny, jeśli wiersze są gęsto upakowane w blokach.

Nazwa	Zawartość
tCheckpoint	Znaczniki czasowe załadowanych obrazów stanów punktowych.
tPointState	Stany punktowe dla wszystkich poziomów i typów.
tPeriod	Charakterystyczne okresy wraz z datą początkową i końcową – tygodnie, miesiące, kwartały, półrocza, lata.

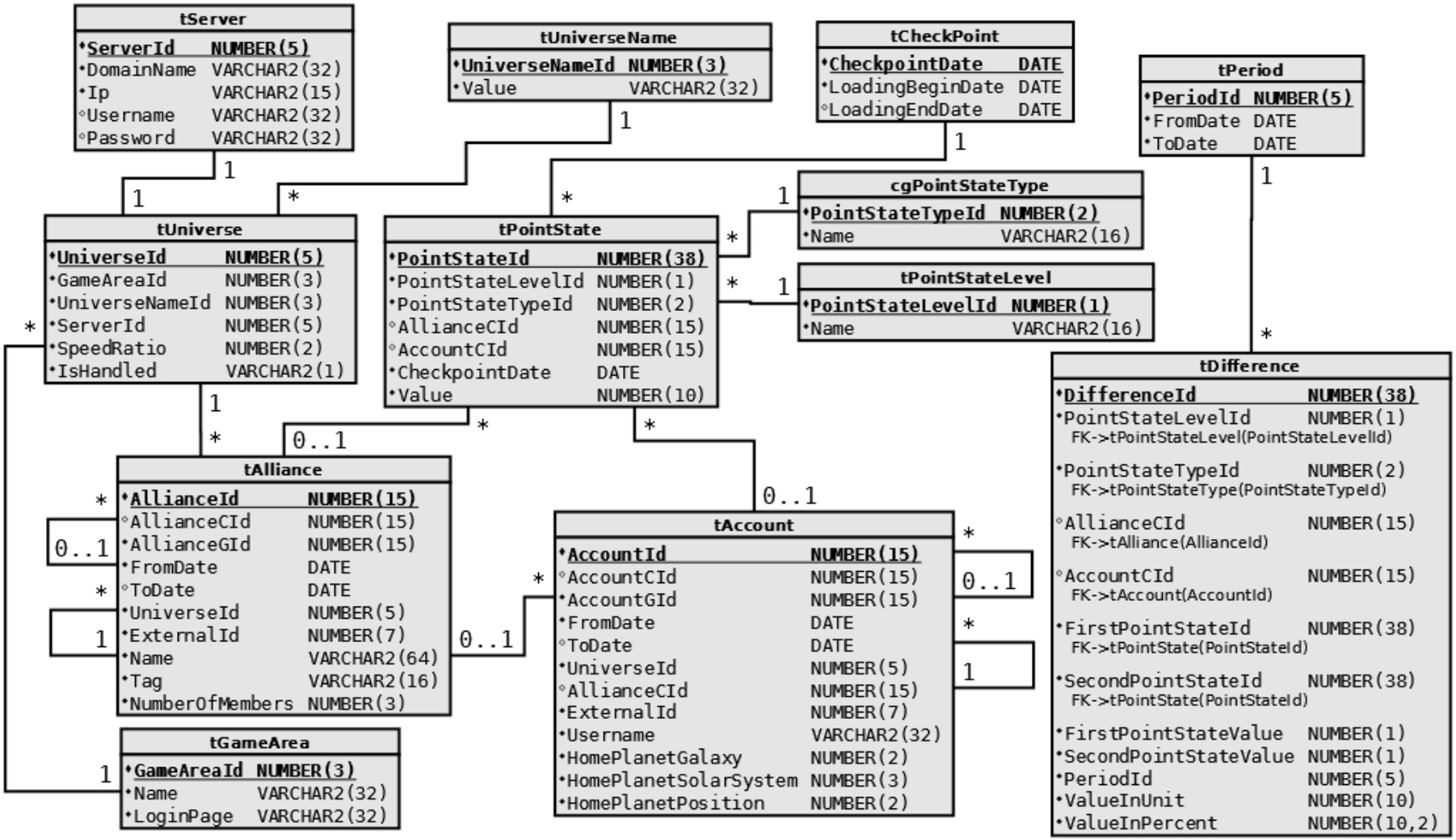
Tabela 17. Relacje ciągle rozbudowywane.

Część raportowa wzrasta tylko wtedy, gdy należy wyznaczyć statystyki dla okresu zdefiniowanego w tabeli *tPeriod*. Aktualnie, w skład tej grupy wchodzi tylko jedna relacja przedstawiona w tabeli 18. Obiekt *tDifference* został zdenormalizowany w celu zwiększenia wydajności. Zawiera on dane w prawie takiej samej postaci, jaka jest żądana przez użytkownika.

Nazwa	Zawartość
tDifference	Różnice pomiędzy dwoma stanami punktowymi tego samego poziomu i typu, dla tego samego gracza lub sojuszu. Jeśli stan punktowy z jego znacznika czasowego nie posiada swojego odpowiednika w drugim znaczniku czasowym to nie da się wyznaczyć różnicy i rekord nie powstaje.

Tabela 18. Relacje raportowe.

Rysunek 16 przedstawia graficznie relacje między tabelami stworzonymi na potrzeby aplikacji testowej.



Rysunek 16. Diagram ERD bazy danych.

9. Dodatek 2 - Spis tabel

Tabela 1. Rozkład woluminów na dysku maszyny wirtualnej 32

Tabela 2. Specyfikacja rzeczywistej maszyny hosta. 32

Tabela 3. Wydajność pamięci masowej maszyny hosta. 32

Tabela 4. Wydajność pamięci operacyjnej maszyny hosta. 33

Tabela 5. Konfiguracja testowej instancji Oracle TimesTen. 33

Tabela 6. Konfiguracja testowej instancji Oracle Database. 34

Tabela 7. Zaplanowane warianty ładowania danych do bazy dla systemu transakcyjnego. 47

Tabela 8. Zaplanowane warianty ładowania danych do bazy dla wsadowego zasilenia hurtowni. 48

Tabela 9. Zaplanowane warianty obliczania statystyk. 48

Tabela 10. Zaplanowane warianty generowania raportów. 49

Tabela 11. Pomiary wydajności wykonane tylko dla Oracle Database. 49

Tabela 12. Pomiary wydajności ładowania danych do bazy. 50

Tabela 13. Pomiary wydajności obliczania statystyk. 51

Tabela 14. Pomiary wydajności generowania raportów. 51

Tabela 15. Relacje rzadko rozbudowywane i aktualizowane. 66

Tabela 16. Relacje ciągle rozbudowywane i aktualizowane. 67

Tabela 17. Relacje ciągle rozbudowywane. 67

Tabela 18. Relacje raportowe. 67

10. Dodatek 3 - Spis rysunków

Rysunek 1. Porównanie działania Oracle Database z Oracle TimesTen [9]. 10

Rysunek 2. Architektura mechanizmu IMDB Cache [9]. 13

Rysunek 3. Komunikacja między grupą pamięci podręcznej a Oracle Database [15]. .. 20

Rysunek 4. Zasada działania mechanizmu delegacji żądań [9]. 23

Rysunek 5. Schemat wykonania kodu SQL w bloku PL/SQL [20]. 28

Rysunek 6. Przepływ informacji pomiędzy modułami aplikacji [1]. 37

Rysunek 7. Przypadki użycia aplikacji testowej. 38

Rysunek 8. Organizacja grup pamięci podręcznej w testowanej aplikacji. 41

Rysunek 9. Hipotetyczna grupa pamięci cgUniverse po zmianie założeń aplikacji. 43

Rysunek 10. Wykres czasu ładowania danych w trybie transakcyjnym. 52

Rysunek 11. Wykres czasu ładowania danych w trybie wsadowym. 53

Rysunek 12. Wykres czasu obliczania statystyk. 54

Rysunek 13. Wykres czasu generowania raportu w zależności od rodzaju wykonania. 55

Rysunek 14. Wykres przyśpieszenia Oracle Database z IMDB Cache względem Oracle Database z buforowaniem PL/SQL. 56

Rysunek 15. Wykres przyśpieszenia instancji z buforem PL/SQL w zależności od konfiguracji 57

Rysunek 16. Diagram ERD bazy danych. 68