PYTANIA EGZAMINACYJNE - ZARZĄDZANIE BAZAMI DANYCH 2017/18

Część teoretyczna – jedno pytanie na egzaminie – materiał pochodzi z plików PowerPointowych w katalogu P:\FTP(Public)\lech\zbd1

1. Trzy poziomy abstrakcji schematu bazy danych i niezależność danych
2. **Logiczny** – schemat tabel bazy danych
3. **Fizyczny** – struktury danych realizujące poziom logiczny w systemie plików
4. **Zewnętrzny** – perspektywy (widoki, views) użytkowników na obiekty w bazie danych

Z punktu widzenia użyteczności istotna jest wzajemna *niezależność* tych poziomów.

1. Zmiany na poziomie logicznym są niewidoczne dla użytkowników bazy danych (dla ich aplikacji).
2. Można dokonywać zmian na poziomie fizycznym nie dokonując zmian na poziomie logicznym.
3. Użycie perspektyw do realizacji zewnętrznego poziomu w bazie danych

 Zapamiętana w bazie danych definicja zapytania SQL do późniejszego użycia:

 perspektywa jest “wirtualną” tabelą,

* 1. używana tak jak tabela,
	2. jej wiersze nie są przechowywane w bazie danych,
		1. są wyliczane na żądanie.

Opcja WITH CHECK OPTION - przy INSERT i UPDATE następuje sprawdzenie, czy wstawiany bądź modyfikowany wiersz spełnia warunek określony w klauzuli WHERE podzapytania:

* + jeśli spełnia, operacja jest wykonywana;
	+ jeśli nie spełnia, operacja nie zostanie wykonana.
* Umożliwiają realizację poziomów zewnętrznych bazy danych i logicznej niezależności danych.
* Udostępnia się je konkretnym grupom użytkowników. Określają widok na bazę danych zaprojektowany dla tej grupy użytkowników.
	+ Ułatwia to użycie przez nich danych.
	+ Stanowią element ochrony przed niepowołanym lub nieprawidłowym dostępem do danych.
	+ Każdy użytkownik bazy danych ma dostęp tylko do danych dotyczących jego działalności w firmie.
* W wersji zmaterializowanej służą bądź jako *repliki danych* przechowywane w odległych węzłach bądź jako *agregacje danych* do dalszych analiz w hurtowni danych.
1. Instrukcje SQL z klauzulą WITH

Można poprzedzić instrukcje SELECT, INSERT, DELETE i UPDATE klauzulą WITH, definiując lokalne perspektywy widoczne tylko w danej instrukcji SQL. Umożliwiają programowanie w samym SQL.

WITH
**dept\_costs** AS **(** SELECT department\_name, SUM(salary) dept\_total
 FROM employees e INNER JOIN departments d
 ON e.department\_id = d.department\_id
 GROUP BY department\_name**)**,
**avg\_cost** AS **(** SELECT SUM(dept\_total)/COUNT(\*) avgo
 FROM **dept\_costs)**

SELECT \* FROM **dept\_costs**, **avg\_cost**
WHERE dept\_total > avgo
ORDER BY department\_name;

1. Słownik systemowy (danych)
* Ma postać zbioru tabel i perspektyw.
* z przedrostkiem User - informacja o wszystkich obiektach, których dany użytkownik jest właścicielem:
	+ tabele: User\_Tables (Tabs),
	+ tabele i kolumny: User\_Tab\_Columns (Cols),
	+ więzy spójności: User\_Constraints, User\_Cons\_Columns,
	+ indeksy: User\_Indexes (Ind), User\_Ind\_Columns,
	+ synonimy: User\_Synonyms (Syn),
	+ perspektywy: User\_Views,
* z przedrostkiem All - dotyczy wszystkich obiektów, do których użytkownik ma uprawnienia (np. All\_Tables),
* z przedrostkiem Dba - informacja o obiektach dostępnych dla administratorów systemu (np. Dba\_Users).

SELECT Table\_Name

FROM User\_Tables; -- Podobnie All\_Tables

 wypisuje nazwy tabel, których właścicielem jest dany użytkownik.

SELECT Column\_Name, Data\_Type

FROM User\_Tab\_Columns

WHERE Table\_Name = 'EMP';

1. Wyzwalacze w bazie danych

Fragmenty kodu:

* wiązane z użyciem jednego z obiektów:
	+ tabeli,
	+ perspektywy,
	+ schematu,
	+ bazy danych;
* wywoływane (*odpalane*) przez system przy zajściu odpowiedniego zdarzenia, które może być albo zdarzeniem systemowym albo realizacją jednej z instrukcji SQL.

*Wyzwalacze typu INSTEAD OF*

* *Realizacja postulatu modyfikowania danych w bazie danych za pomocą perspektyw.*
* *CREATE [OR REPLACE] TRIGGER nazwa
INSTEAD OF specyfikacja\_instrukcji ON perspektywa
blok\_PL/SQL*
	+ *gdzie specyfikacja\_instrukcji – ciąg do trzech nazw INSERT, DELETE i UPDATE połączonych spójnikiem OR*
1. Obiektowo-relacyjny model danych
	1. Nowe typy danych dla wartości w kolumnach, jak obiekty, LOB, XML, JSON, listy, zbiory - odejście od I postaci normalnej
	2. Programy Java i C# po stronie serwera baz danych
	3. Generyczne aplikacje baz danych jak zarządzanie zasobami i/lub klientami przedsiębiorstwa, czy witryny do wprowadzania treści - gotowe do wdrożenia
	4. Hurtownie danych – zastosowania analityczne
	5. Połączenia z magazynami danych BigData
	6. Realizacja baz danych w chmurze
	7. Rozproszone bazy danych dostosowane do zastosowań typu NoSQL
2. Typy kolekcji
	1. **Atrybut typu obiektowego lub kolumna tabeli nie tylko może być typu obiektowego ale także typu kolekcji:**
	2. **1. VARRAY – typ tablicy jednowymiarowej (jak wektor) o ustalonym maksymalnym rozmiarze;**
	3. **2. TABLE – typ tabeli zagnieżdżonej o nie ustalonym maksymalnym rozmiarze.**
3. Typy referencyjne
	1. Zamiast użycia klucza obcego i indeksu na kluczu głównym – referencja
	2. Dla każdego typu obiektowego Type jest automatycznie definiowany jego typ referencyjny oznaczany przez REF Type. Typu tego można używać jako typu atrybutów obiektów bądź kolumn w tabeli relacyjnej. Daje to możliwość wiązania wartości atrybutu bądź wartości w kolumnie z innym obiektem podobnie jak klucz obcy odwołuje się do klucza głównego. Jest to alternatywny sposób tworzenia powiązań.
	3. Reasumując, w obiektowo-relacyjnej bazie danych pojawiają się dwa nowe rodzaje wartości związane z zapisem danych na dysku: referencje do obiektów oraz jednoznaczne identyfikatory obiektów. Każda implementacja obiektowo-relacyjnej bazy danych może używać innych zbiorów tych wartości, co powoduje, że obiektowo-relacyjna baza danych przestaje być przenaszalna (jako zbiór tabel z atomowymi, skalarnymi wartościami) między różnymi systemami. Tracimy więc wielką zaletę relacyjnych baz danych.
	4. SELECT o.name, o.Dept\_ref.name
	FROM obj\_emp o;
	5. NAME DEPT\_REF.NAME
	6. -------------------- -------------------------
	7. KOWALSKI KADRY
4. Duże obiekty LOB w bazie danych
	1. Na ogół fizycznie w wierszu tabeli jest zapisywany adres do miejsca na dysku, gdzie składowany jest duży obiekt LOB
	2. BLOB – *binarny duży obiekt* - strumień bitów jak w przypadku LONG RAW.
	3. CLOB – *znakowy duży obiekt* - strumień znaków.
	4. NCLOB – *uogólniony* (dla języków narodowych wielobajtowych) *znakowy duży obiekt*.
	5. BFILE – plik binarny przechowywany poza bazą danych.
	6. Duże obiekty są przechowywane albo wewnętrznie - wewnątrz bazy danych – dotyczy to wartości typów CLOB, NCLOB, BLOB, albo zewnętrznie w plikach systemu operacyjnego – dotyczy to wartości typu BFILE, które są dostępne z serwera Oracle tylko w trybie odczytu. Serwer Oracle nie potrafi automatycznie dokonywać konwersji danych między różnymi typami danych LOB, np. z wartości typu CLOB do wartości typu BLOB.
	7. Z każdym dużym obiektem LOB jest związana:
	8. wartość LOB: przechowywana w bazie danych zawartość dużego obiektu,
	9. lokalizator LOB: wskaźnik do wartości LOB przechowywanej w bazie danych.
5. Reprezentacja dokumentu XML w bazie danych
6. W bazie danych zapisanie obok siebie na dysku powiązanych danych w celu przyśpieszenia do nich dostępu (zamiast w kilku tabelach/kolumnach) – tworzenie paczek danych.
7. Integracja danych pochodzących z różnych baz danych i aplikacji.
	* Ułatwienie wymiany danych biznesowych między aplikacjami i bazami danych,
		+ np. między przeglądarką i bazą danych.
8. Dołączenie metadanych do klasycznych dokumentów.
9. Oddzielenie prezentacji od struktury dokumentu, przez co umożliwienie różnych prezentacji tego samego dokumentu.
10. Typ XMLType
	1. W Oracle dokument XML może być reprezentowany jako obiekt wbudowanego typu obiektowego XMLType. Za pomocą metod typu XMLType można tworzyć, wydobywać i indeksować dane XML przechowywane w bazie danych. Typ XMLType tak jak każdy typ obiektowy może być używany jako typ danych kolumn lub jako typ tabeli obiektowej.
11. Dyskowy model fizyczny bazy danych

***Dyskowy model fizyczny – Oracle***

* Gdy rozmiar rekordu jest większy niż rozmiar strony, rekord jest dzielony na części przechowywane na osobnych stronach.
* Gdy schemat dostępu do danych polega na użyciu powiązanych danych z dwóch lub więcej tabel (np. departamenty i ich pracownicy; klienci, zamówienia i pozycje zamówień), w jednym pliku są zbierane dane z kilku tabel w oparciu o wspólny klucz (np. numer departamentu czy identyfikator klienta).

***Dyskowy model fizyczny - SQL Server***

* Jeśli wiersz nie mieści się na stronie (8kB), nie mieszcząca się część wiersza zostaje zapisana na liście stron nadmiarowych a wskaźnik do strony nadmiarowej zostaje zapisany w głównym wierszu.
* Dane większych rozmiarów (>8000Bajtów) w kolumnach typu: text, image, ntext, varchar(MAX) są zapisywane na osobnych stronach LOB.



1. Zarządzanie miejscem na dysku

W pliku mamy dwa rodzaje stron:

strony z wolnym miejscem do zapisu nowych rekordów, oraz

strony całkiem albo prawie całkiem zapełnione.

Zakładamy także, że system ma możliwość rozszerzania pliku rekordów i alokacji nowych, pustych stron oraz że system ma możliwość zwrotu (dealokacji) nieużywanych stron.

Moduł zarządzania miejscem na dysku (ang. disk space manager) realizuje następujące funkcje:

Alokacja ciągu stron (położonych spójnie na dysku).

Zwrot (dealokacja strony).

Wyznaczenie strony do zapisu nowego rekordu.

Aktualizacja struktur danych na dysku związanych z przechowywanymi stronami.

Rozmiarem strony jest zwykle rozmiar bloku dyskowego i strony są przechowywane jako bloki dyskowe, więc odczyt/zapis strony wykonuje się jako pojedyncza operacja We/Wy.

Moduł zarządzający miejscem na dysku ukrywa szczegóły sprzętu i systemu operacyjnego i umożliwia pozostałym modułom traktowanie danych na dysku jako zbioru stron.

1. Zarządzanie buforami (w RAM)
	1. *Dane muszą być w RAM aby SZBD mógł na nich operować!*
	2. *Pomocnicza struktura danych: tablica par <nr\_bufora, id\_strony>.*



* O szybkości działania aplikacji decyduje liczba operacji odczytu i zapisu stron na dysku.
* Wzrastają rozmiary pamięci RAM i w związku z tym wzrasta liczba buforów danych.
* Wielokrotnie używane strony są przechowywane w buforach.
	+ Gdy baza danych jest używana przez aplikację biznesową, przyjmuje się, że średnio ponad 90% potrzebnych do wykonania zapytania danych powinno znajdować się w buforach.
* Zarówno skompilowane instrukcje SQL jak i wyniki zapytań są zapisywane w buforach i używane wielokrotnie.
	+ Wyniki zapytań mogą też być zapisywane w buforach serwera aplikacyjnego pośredniczącego między klientem a serwerem bazy danych w architekturze trójwarstwowej.
1. Formaty rekordów i stron
	1. **4. *Formaty rekordów: stała długość***
	2. Rozmiary pól takie same dla wszystkich rekordów w pliku; zapisane w *katalogu systemowym.*
	3. 
	4. ***Formaty rekordów: zmienna długość***
	5. Dwa alternatywne formaty (#*pól* jest stała):

W drugim przypadku:

* bezpośredni dostęp do wartości *i*-tego pola;
* efektywne przechowywanie pseudo-wartości *null*
* 

***Format mieszany rekordu***

Każdy rekord składa się z dwóch części: stałego i zmiennego rozmiaru zarządzanych odpowiednio przy użyciu przedstawionych poprzednio metod (tak jest w SQL Server).

***Formaty stron: rekordy stałej długości***

* *W pierwszym przypadku, przesuwanie rekordów powoduje zmianę miejsca rekordu, co komplikuje odwołania do rekordu.*

**

***Formaty stron: rekordy zmiennej długości***

* *Można przesuwać rekordy po stronie bez konieczności zmiany identyfikatora rekordu:* ***rid = (nr strony, nr w tablicy pozycji)***

**

1. Podstawowe organizacje plików bazy danych

Plik rekordów:

* PLIK (segment): kolekcja stron, każda zawierająca zbiór rekordów:
	+ wstawianie/usuwanie/modyfikowanie rekordu,
	+ odczytywanie konkretnego rekordu (o podanym *rid*),
	+ wyszukiwanie wszystkich rekordów spełniających podane warunki.

***5.1 Plik nieuporządkowany (sterta, heap)***

* *Rekordy* są przechowywane na *stronach* w dowolnym porządku.
* Nowy rekord jest wstawiany do pierwszej strony, na której jest wolne miejsce.
* Efektywne wyszukiwanie rekordów, gdy dotyczy sporej części całej tabeli np. 20%.
* Przy wyszukiwaniu rekordów po wartościach kluczy wyszukiwania stosowana łącznie z indeksami.





***Plik uporządkowany (posortowany)***

*Rekordy są zapisywane na kolejnych stronach zgodnie z porządkiem względem pewnego klucza wyszukiwania rekordu. Taka reprezentacja jest wygodna gdy rekordy są przetwarzane zawsze w tym samym, ustalonym porządku lub tylko pewien ich zakres względem tego porządku np.
SELECT \* FROM Emp e ORDER BY e.Sal lub
SELECT \* FROM Emp e WHERE e.Sal BETWEEN 1000 and 2000*

*Wyszukanie rekordu mając daną wartość jego klucza wyszukiwania zależy od dodatkowych szczegółów implementacji.*

*Bardziej skomplikowane stają się operacje wstawienia nowego rekordu do pliku jak i usunięcia rekordu z pliku.*

***Plik haszowany (Oracle)***

Plik jest kolekcją “***segmentów****” (ang. bucket).* ***Segment*** = *strona główna* plus zero lub więcej *stron nadmiarowych*.

*Funkcja haszująca***h**: **h**(*r*) = “segment” do którego wpada rekord *r*. Funkcja **h** bierze pod uwagę tylko klucz wyszukiwania rekordu *r*.

Organizacja pliku haszowanego jest użyteczna przy wyborze rekordu z pliku w oparciu o wartość lub wartości pewnych pól rekordu np. przy wykonywaniu zapytania:
 SELECT \* FROM Emp e WHERE e.Ename=:Nazwisko



***Wielowymiarowa tablica (DB2)***

*Komórka – zbiór rekordów z tym samym układem wartości wymiarów. Realizowana jako lista ekstentów.*

Zastosowania:

1. hurtownia danych np. tabela sprzedaży – alternatywa dla indeksów bitmapowych i haszowania,
2. dane przestrzenne np. zbiór punktów w przestrzeni kartezjańskiej wielowymiarowej.



1. Realizacja wyszukiwania rekordów po kluczu wyszukiwania
	1. Skorzystanie z organizacji pliku rekordów:
	2. Gdy plik posortowany i *klucz wyszukiwania* = *klucz sortowania,* skorzystanie z indeksu,
	3. Inaczej, przejście sekwencyjne pliku w poszukiwaniu rekordu.
	4. Skorzystanie ze zbudowanego wcześniej (obok pliku rekordów) osobnego indeksu dla klucza wyszukiwania.
	5. **CREATE INDEX Ind\_Ename\_Emp ON Emp(Ename);**
2. Indeksy pogrupowane i niepogrupowane
	1. *Indeks pogrupowany*
		1. **Uporządkowany plik rekordów** jest oparty na indeksie drzewowym. Indeks ten nazywa się *pogrupowanym* (ang. *clustered*) - w liściach znajdują się **strony z rekordami** **danych**.
		2. Rekordy w liściach drzewa są posortowane według wartości klucza indeksu.
		3. Dla jednej tabeli może być zbudowany tylko jeden indeks pogrupowany.
		4. Pozostałe indeksy - *niepogrupowane* – nie zawierają w sobie stron z rekordami danych.
	2. Indeks niepogrupowany
		1. Jeśli na tabeli nie jest założony indeks pogrupowany, wszystkie indeksy są niepogrupowane i wskaźnikami w liściach są **adresy do rekordów**.
		2. Jeśli na tabeli jest założony indeks pogrupowany, wskaźnikami w liściach indeksu niepogrupowanego są **wartości klucza indeksu pogrupowanego** a nie adresy do rekordów.
		3. Aby znaleźć rekord po wartości klucza indeksu niepogrupowanego:
		4. najpierw przechodzimy indeks niepogrupowany znajdując odpowiadającą mu wartość klucza indeksu pogrupowanego,
		5. następnie przechodzimy indeks pogrupowany aby znaleźć sam rekord z tą wartością klucza indeksu pogrupowanego.
3. Sortowanie w bazie danychZnaczenie pola działania (workload) aplikacji bazy danych.Ulepszanie schematu tabel a postacie normalnIndeks wewnętrzny w Oracle.Indeks haszowany w Oracle.
	1. ORDER BY - dane są wymagane w pewnym porządku.
	2. Budowa indeksu - początkowego drzewa wyszukiwań dla wczytywanego zbioru rekordów.
	3. Złączanie tabel metodą *Sort-merge*.
	4. Realizacja DISTINCT, GROUP BY, UNION, EXCEPT.
	5. ***Sortowanie zewnętrzne - wielofazowe przez scalanie***
		1. Faza 0 – sortowanie rekordów w ramach stron: wczytaj stronę, posortuj, zapisz na dysku.
		2. Faza 1,2,3 …, itd: scalaj uporządkowane fragmenty w większe uporządkowane fragmenty, aż cały plik zostanie uporządkowany.
	6. 
	7. ***Sortowanie za pomocą indeksu***
		1. Scenariusz: Tabela ma indeks względem kolumn sortowania.
		2. Idea: Przejść po liściach indeksu.
		3. ***Czy jest to dobra metoda?***
		4. Przypadki:
			1. Indeks *pogrupowany* ***Bardzo dobra!***
			2. Indeks *niepogrupowany*  Może być ***bardzo zła!***
	8. Indeks pobrupowany
		1. Od korzenia przejdź do skrajnie lewego liścia a następnie sekwencyjnie w prawo po liściach.
		2. 
		3. ***Zawsze lepsze od sortowania zewnętrznego!***
	9. ***Indeks niepogrupowany***
		1. Ogólnie, jedna operacja We/Wy na rekord danych!
		2. 
4. Realizacja selekcji
	1. Bez indeksu Rozważ kolejno wszystkie rekordy pliku.
	2. Jeśli jest indeks
		1. na atrybucie selekcji E.Ename: Użyj go, wyznacz pozycje danych w indeksie, przejdź do rekordów.
		2. obejmujący wszystkie atrybuty w zapytaniu (E.Ename, E.Sal): Przejdź przez indeks (strategia *tylko-indeks*), odczytaj dane z indeksu nie przechodząc w ogóle do rekordów w pliku.
5. Strategia „tylko-indeks”
	1. ***Kolumny INCLUDE w indeksie***
		1. Umożliwiają korzystanie z metody optymalizacyjnej „Tylko indeks”.
		2. Klauzula INCLUDE dodaje dodatkowe kolumny do pozycji danych w liściach indeksu niepogrupowanego.
		3. SELECT Imie, Nazwisko, Data\_ur
		FROM Osoby
		WHERE Nazwisko = 'Kowalski'
		4. Indeks: Osoby(Nazwisko), kolumny INCLUDE: Imie i Data\_ur. Wyszukiwanie po *nazwisku*, w pozycji danych odczytanie danych *imienia* i *daty urodzenia* bez sięgania do stron z rekordami.
		5. CREATE NONCLUSTERED INDEX Osoby\_po\_nazwisku
		ON Osoby(Nazwisko) INCLUDE (Imie, Data\_ur)
	2. **PROJEKCJA** Bez DISTINCT – przepisanie.
	3. Z DISTINCT – wymagane jest wyeliminowanie powtórzeń:
	4. posortowanie i eliminacja powtarzających się wartości, które po sortowaniu znajdują się obok siebie.
	5. Gdy jest indeks, który obejmuje wszystkie atrybuty występujące w klauzuli SELECT – wystarczy przejść tylko ten indeks (***strategia tylko indeks***).
6. Realizacja złączenia
7. Użycie perspektyw zmaterializowanych
	1. *Perspektywa zmaterializowana* – perspektywa, której wiersze wynikowe są zapisywane tak jak wiersze tabeli.
	2. Można na niej założyć indeksy tak jak na tabeli.
	3. Można raz wyliczyć pracochłonne złączenia i statystyki, a następnie z nich wielokrotnie korzystać.
	4. Potrzebne jest uaktualnianie zawartości po operacjach na tabelach źródłowych perspektywy.
	5. CREATE MATERIALIZED VIEW Stat
	AS SELECT Dname, COUNT(Empno) No, AVG(Sal) AvSal
	FROM Emp INNER JOIN Dept ON Emp.Deptno=Dept.Deptno
	GROUP BY Dept.Deptno, Dept.Dname;
8. Optymalizacja zapytań, drzewo zapytania, plan wykonania zapytania
	1. **Optymalizacja zapytań**
		1. *Budowa drzewa zapytania* (przez parser)
		2. *Plan*: *Plan* (*algorytm) wykonania zapytania w postaci drzewa operatorów +*
		3. sposób dostępu do rekordów na dysku,
		4. metody realizacji operatorów
		5. *Optymalizator generuje plany, szacuje ich koszt i wybiera do realizacji ten z najmniejszym kosztem.*
9. Aksjomaty ACID współbieżnego wykonywania transakcji
10. *Atomowość (niepodzielność)* - albo wszystkie akcje wchodzące w skład transakcji są wykonywane albo żadna.
11. *Spójność* - po wykonaniu transakcji stan bazy danych powinien być spójny (pod warunkiem, że przy rozpoczynaniu transakcji stan bazy danych był spójny).
12. *Izolacja* - wynik działania transakcji powinien być taki sam, jakby od chwili rozpoczęcia transakcji nie działała na wspólnych danych żadna inna transakcja. Każdy użytkownik powinien mieć poczucie, że sam korzysta z bazy danych.
13. *Trwałość* - dane zatwierdzone przez transakcję powinny być dostępne nawet w sytuacji awarii oprogramowania lub sprzętu.

W praktyce, podniesienie wyjątku (np. w Oracle ***no\_data\_found*** lub ***too\_many\_rows***)przy wykonywaniu pojedynczej instrukcji SQL powoduje tylko nie wykonanie tej jednej instrukcji i samo nie powoduje jeszcze automatycznego wycofania całej transakcji i powinno być odpowiednio obsłużone w aplikacji, która zgłasza transakcję do wykonania – i w miarę potrzeby z zastosowaniem przez nią explicite instrukcji ROLLBACK.

1. Mechanizmy współdzielenia zasobów bazy danych
* *blokady (zamki, locks)* – ograniczają działanie innych transakcji na zablokowanym obiekcie;
* *dzienniki transakcji (logi)* - opisują działanie transakcji:
	+ w oparciu o ten opis można wycofać transakcję jak i odtworzyć stan bazy danych sprzed awarii;
* *kopia zabezpieczająca* bazy danych (backup) - wykonywana w regularnych odstępach czasu, np. raz na dzień;
* mechanizm *wielowersyjności* – dostęp do różnych wersji tych samych danych,
	+ umożliwia odczytywanie danych zmienianych równocześnie przez inne transakcje w takiej postaci, w jakiej istniały w chwili rozpoczynania się danej instrukcji lub transakcji.
1. Protokół ścisłego blokowania dwufazowego (2PL)

***Dwufazowość protokołu Strict 2PL***

Dwie fazy działania transakcji:

zakłada blokady i dokonuje wymaganych odczytów i zapisów na obiektach, na których założyła blokadę;

wykonuje COMMIT/ROLLBACK i zwalnia wszystkie blokady.

***Słabsza wersja protokołu 2PL***

* Protokół *Weak 2PL*:
	+ transakcja nie może przed swoim końcem zwolnić żadnej blokady X, ale w każdej chwili może zwolnić założoną do odczytu blokadę S (np. w chwili zakończenia odczytu);
	+ nie gwarantuje powtarzalności odczytów w ramach jednej transakcji;
	+ używany przez system jako domyślny: zwiększa wydajność systemu, ale programista musi dodatkowo zadbać o poprawność swojej transakcji w sytuacji niepowtarzalności dokonywanych odczytów!
* Każda transakcja może wybrać własny poziom izolacji od innych transakcji:
	+ *Strict-2PL* (Serializable, szeregowalny – realizacja równoważna szeregowemu wykonaniu transakcji jedna po drugiej),
	+ *Weak-2PL* (Read Committed).
1. Problem fantomów
* Protokół *Strict 2PL* (w dotychczasowej postaci) jest poprawny pod warunkiem, że do tabeli w trakcie wykonywania transakcji nie są wstawiane nowe wiersze.
* W trakcie realizacji transakcji mogą się pojawić nowe wiersze wstawione przez inne transakcje, co powoduje, że wyniki poprzednio wykonanych zapytań mogą się zmienić i, że transakcja nie jest w pełni izolowana od innych współbieżnych transakcji.
* Trzeba zablokować albo całą tabelę w trybie S albo jej fragment – używając w tym celu blokady na węźle indeksu.
1. Poziomy izolacji
	1. **Read Committed** - ochrona tylko przed odczytem niezatwierdzonych danych.
	2. **Serializable** - ochrona przed odczytem niezatwierdzonych danych, niepowtarzalnym odczytem i fantomami.
		1. realizacja transakcji równoważna szeregowemu ich wykonaniu jedna po drugiej.

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE (Read committed);**

 **.......**

 **COMMIT;**

1. Wielowersyjność w bazie danych
	1. **Idea:** Procesy zapisujące tworzą nową kopię obiektu podczas gdy procesy odczytujące korzystają ciągle ze starej wersji (np. SQL Server):
	2. Procesy odczytujące (np. przygotowujące raporty) mogą działać bez zakładania blokad mając zagwarantowany spójny widok na dane w bazie.
	3. 
2. Dzienniki transakcji
	1. Dziennik transakcji - rejestruje wszystkie zmiany zachodzące w bazie danych. Jest on z założenia trzymany na innym nośniku danych niż pliki z danymi w bazie danych.
	2. Na ogół, dokonuje się regularnie jego archiwizacji (BACKUPu) przepisując go na obszerniejszy nośnik danych.
	3. Zmiana danych na stronie i informacja o zatwierdzeniu są najpierw zapisywane do *dziennika transakcji,* dopiero potem uwzględniane w pliku danych na dysku (zasada Write-Ahead-Logging).
		1. Po zapisie do dziennika nawet awaria serwera lub dysku z danymi nie spowoduje utraty danych, bo można je odtworzyć (własność trwałości).
3. Realizacja odtwarzania po awarii serwera
	1. *Punkt kontrolny (checkpoint)*
		1. Synchronizacja zawartości pamięci i dysku.
		2. Stanowi punkt początkowy dla odtwarzania po awarii serwera bazy danych.
	2. *Ucinanie dziennika*
		1. Dziennik transakcji na dysku jest strukturą cykliczną. Miejsce zajmowane przez najdawniej zapisane rekordy jest nadpisywane przez nowe rekordy.
		2. Do odtworzenia po awarii serwera są potrzebne rekordy dziennika *od chwili rozpoczęcia się najstarszej nie zatwierdzonej w chwili punktu kontrolnego transakcji*.
		3. Można więc w chwili punktu kontrolnego „odciąć” starsze rekordy dziennika (po ich zbackupowaniu).
		4. Po odtworzeniu po awarii (lub restartowaniu) serwera można uciąć cały stary dziennik (po jego zbackupowaniu).
		5. Do odtwarzania po awarii mediów trzeba mieć pełny backup dziennika.
4. Realizacja odtwarzania po awarii dysku
	1. Gdy nastąpi utrata *zawartości pamięci RAM*, analiza samego dziennika transakcji (z części on-line na dysku) pozwala na odtworzenie stanu bazy danych w chwili awarii.
	2. Gdy nastąpi *awaria dysku z danymi*, pozycje dziennika transakcji (z części on-line na dysku i z jego backupu) zastosowane do backupu bazy danych pozwalają odtworzyć stan bazy danych w chwili awarii.
	3. Jest to proces nazywany ***odtwarzaniem do przodu*.**
	4. Ponieważ w dzienniku transakcji zapisane są również pozycje potrzebne do wycofania transakcji, można wycofać nie zatwierdzone transakcje, których działanie zostało przerwane w chwili awarii.
	5. Jest to proces nazywany ***odtwarzaniem do tyłu***.
	6. W rezultacie obu odtwarzań następuje doprowadzenie stanu bazy danych do spójnego stanu.
5. Rezerwowa baza danych (standby)
	1. Dodatkowa instalacja bazy danych na osobnym komputerze utrzymywana stale w specjalnym trybie *standby* – z ciągle dokonywanym odtwarzaniem w oparciu o kopie dziennika transakcji generowane przez główną, operacyjną bazę danych.
	2. W przypadku awarii dysku lub katastrofy w rodzaju trzęsienia ziemi, pożaru czy kradzieży dotyczących głównej bazy danych, rezerwowa baza danych przechodzi z trybu *standby* w tryb *read write* i przejmuje obowiązki głównej bazy danych.
		1. Również gdy trzeba dokonać zmian w sprzęcie i oprogramowaniu na komputerze, gdzie znajduje się główna baza danych
	3. Można jej używać na bieżąco do realizacji zapytań SELECT, tworzenia raportów, obliczeń analitycznych i backupów odciążając główną bazę danych.
6. Dwu-fazowe zatwierdzanie (2PC)
* Węzeł inicjujący transakcję – koordynator.
* Wykonanie instrukcji COMMIT na sieci węzłów:
	+ Koordynator wysyła komunikat **prepare**.
	+ Węzły zapisują w swoim dzienniku rekord **abort** lub **prepare** a następnie wysyłają do koordynatora komunikat **no** lub **yes**.
	+ Gdy koordynator uzyska jednomyślną odpowiedź **yes**, zapisuje do swojego dziennika rekord **commit** i wysyła komunikat **commit**. Wpp. zapisuje do swojego dziennika rekord **abort** i wysyła komunikat **abort**.
	+ Węzły zapisują w swoim dzienniku odpowiedni rekord **abort/commit** i **end** a następnie wysyłają do koordynatora komunikat **ack**. Kończą swoją część transakcji.
	+ Po otrzymaniu wszystkich potwierdzeń **ack** koordynator zapisuje do swojego dziennika rekord **end** i kończy całą transakcję (usuwając o niej informacje ze swoich buforów w RAM).
* ***Komentarz na temat 2PC***
* Dwie rundy komunikacji: pierwsza - głosowanie; druga - kończenie. Obie inicjowane przez koordynatora.
* Każdy węzeł może zadecydować o wycofaniu (*abort*) transakcji.
* Każdy komunikat odzwierciedla decyzję nadawcy; aby mieć pewność odporności na awarie, decyzja jest najpierw zapisywana do dziennika transakcji (*logu*).

***Dwu-fazowe zatwierdzanie (2PC) z domyślnym wycofaniem***

 **W przypadku podjęcia decyzji o wycofaniu zarówno koordynator (po wysłaniu komunikatu abort) jak i węzeł lokalny od razu dokonują wycofania transakcji.**

* **Brak rekordu transakcji w pamięci RAM oznacza, że została ona wycofana.**
* **Natomiast gdy koordynator podjął decyzję o zatwierdzeniu transakcji rozproszonej, utrzymuje o niej informacje dopóki nie uzyska potwierdzeń ack od wszystkich lokalnych węzłów.**

1. Rozwiązania problemu integracji danych
2. Realizacja rozproszonych baz danych w Oracle

Oracle dostarcza oprogramowania sieciowego umożliwiającego komunikację między bazami danych Oracle oraz obsługę transakcji działających na więcej niż jednej bazie danych – w tym zatwierdzanie takich transakcji i ich wycofywanie.

* **CREATE DATABASE LINK *nazwa\_powiązania***

**CONNECT TO *użytkownik***

**IDENTIFIED BY *hasło***

**USING ’*nazwa\_usługi*’;**

gdzie:

* + *użytkownik/hasło –* dotyczą konta, na które ma zostać dokonane logowanie w odległej bazie danych,
	+ *nazwa\_usługi* – nazwa usługi (aliasu bazy danych) sieciowej Oracle zdefiniowanej w pliku konfiguracyjnym TNSNAMES.ORA
1. Architektura serwera bazy danych Oracle