Pomorska Fundacja
Inicjatyw Gospodarczych

Relacyjne i nierelacyjne bazy danych cz.4

Michał Szymański

Agenda

Dowiem się:

- Co to są indeksy i do czego służą
- Dowiecie się co to jest ‚query optimizer' i plan zapytania
- Poznacie zasadę działania triggerów
- Dowiecie się co to są transakcje bazodanowe
- Będziecie wiedzieć jak tworzyć i używać widoki

Indeksy – właściwie po co?

Znajdowanie wiersza:

imie=‚Jan’

Przeszukiwanie sekwencyjne przy założeniu, że przeglądamy od góry odczytujemy 5 wierszy.

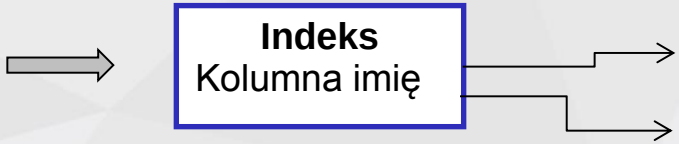
Co by było jakby tabela miała 40 mln wierszy?

ID	Imię	Nazwisko
1	Michał	Szymański
2	Jan	Kowalski
3	Jan	Nowak
4	Stanisław	Wyspiański
5	Jacek	Malczewski

Indeks

Indeksy jest pewną strukturą trzymaną przy bazie danych w celu usprawnienia wyszukiwania określonych danych.

```
SELECT *
FROM klient
WHERE imię=,Jan'
```



ID	Imię	Nazwisko
1	Michał	Szymański
2	Jan	Kowalski
3	Jan	Nowak
4	Stanisław	Wyspiański
…	….	….

https://goo.gl/SmEocG - wprowadzenie do indeksów
https://www.youtube.com/watch?v=C_q5ccN84C8

Indeksy

Indeks może być założony na jedną albo więcej kolumn

```
CREATE INDEX pasazer_nazwisko_idx      ON kontrola.pasazer  (nazwisko);

CREATE INDEX pasazer_imie_nazwisko_idx ON kontrola.pasazer  (imie,nazwisko);

DROP INDEX pasazer_nazwisko_idx ;
```

Zapytania w których używany jest indeks:

- kolumnaX=wartość
- kolumnaX=wartość AND/OR kolumnaY=wartość
- kolumnaX>wartość

Indeksy - typy

Dostępne typy indeksów (tylko B-Tree działa w InnoDB):

- B-Tree - <,>,=, <=,>= (również bierze pod uwage NULL)
- Hahs – tylko =
- (Postgres) GiST – operacje na typach geometrycznych
- (Postgres) GIN - Generalized Inverted Index – przeszukiwania tekstów
- (Postgres,Oracle) Bitmap – dla kolumn z małą licznością

Indeksy – O ile szybciej?

To zależy:

- Rozkładu wartości
- Wielkości tabeli

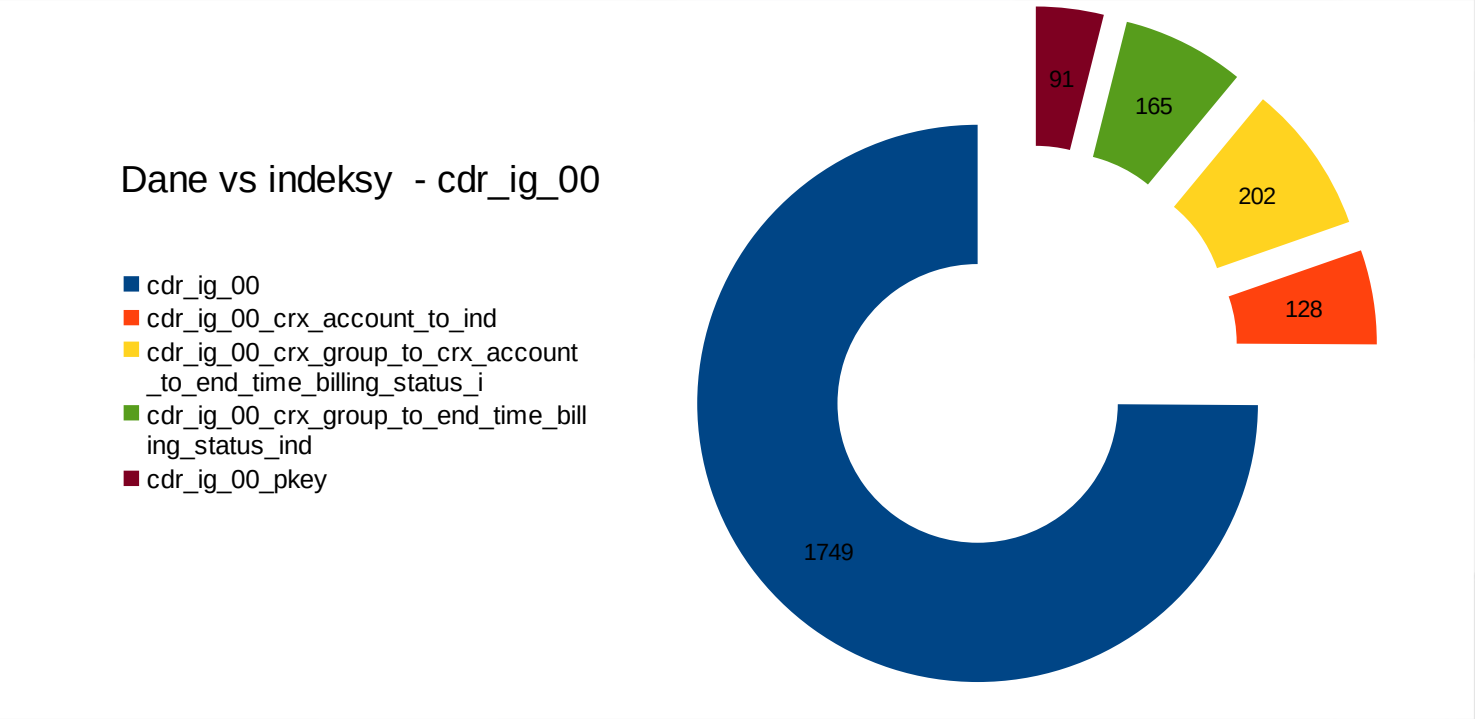
Jednak można się spodziewać wzrostu wydajności nawet o parę rzędów!!

Indeksy – minusy

Dlaczego indeksy nie są doskonałe:

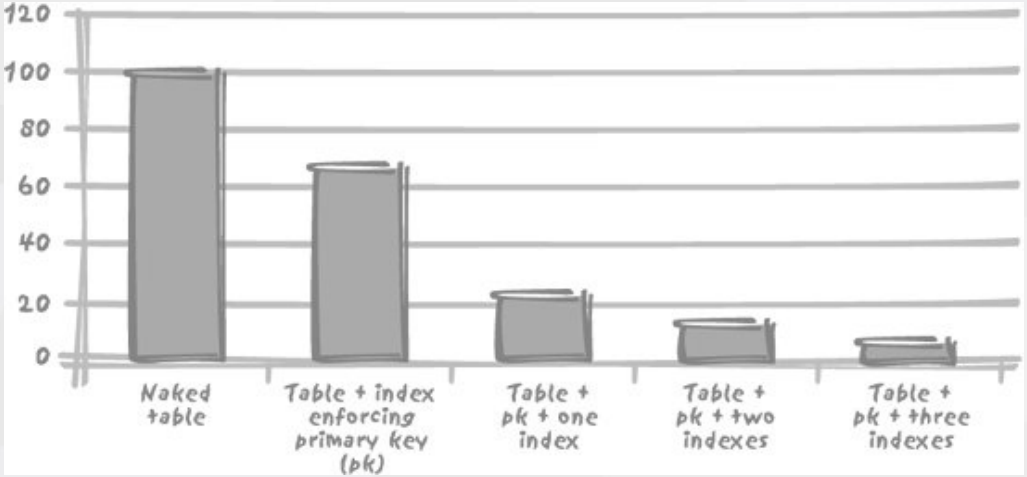
- Zajmują dodatkowe miejsce na dysku
- Przy operacjach modyfikujących wymagają przebudowania
- Tworzenie indeksu jest operacją czasochłonną

Indeksy – minusy



Indeksy – minusy

Wydajność operacji INSERT



Indeksy – Kiedy stosować ?

Porady

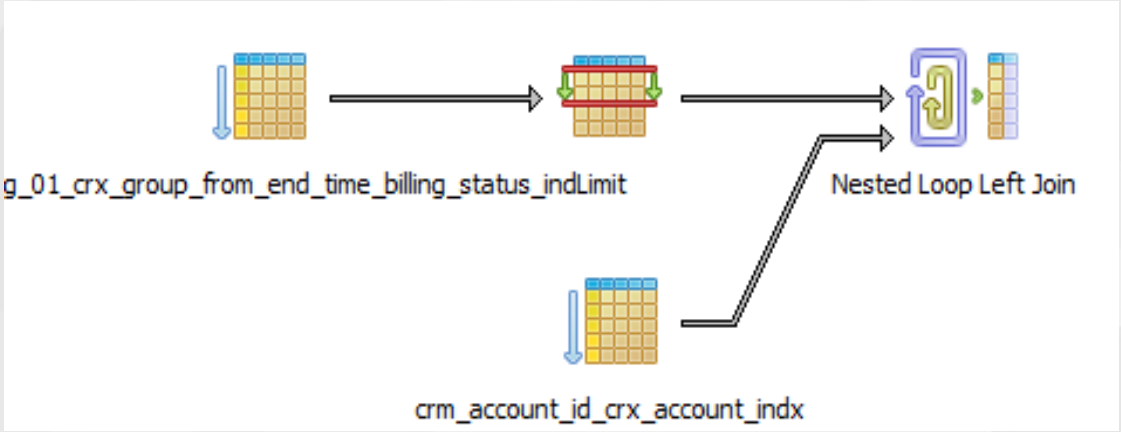
- Indeksy dodajemy z czasem dużo zależy od ilości, rozkładu danych
- Warto posiłkować się statystykami baz danych
- Nie zawsze indeksy to dobry pomysł i nie zawsze rozwiązują problemy wydajności

Indeksy – Query optimizer i plan zapytań

Zapytanie SQL

DB Query optimizer

Plan zapytania



Indeksy – ćwiczenia

Proszę zaproponować i utworzyć indeksy dla tabel z systemu ‚Kontrola’

Dla następujących założeń:
•Chcemy wyszukiwać pasażerów w oparciu o imię i nazwisko
•Chcemy pobierać klientów skontrolowanych przez danego strażnika (lista posortowana)
•Chcemy wyświetlać listę wszystkich portów lotniczych
•Chcemy wyświetlać listę wszystkich kontroli dla wybranego numeru stanowiska dla danego przedziału czasowego (chcemy wyświetlić informacje o pasażerach i strażnikach)

Ilość szacowanych danych:
•Lotniska – 10
•Liczba stanowisk – 100
•Liczba pasażerów - 3mln
•Liczba strażników - 150 / lotniska, suma 1500
•Liczba kontroli - 3 mln/lotnisko, suma 30 mln

Triggery - wyzwalacze

Mechanizm wyzwalający akcje na bazie danych. Wyzwalany jeśli zostanie spełniony zdefiniowany warunek.

Kiedy stosujemy:

- Przenosimy logikę z programu do samej bazy
- Dodatkowe ograniczenia integralności

Triggery - wyzwalacze

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
      trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }
```

FOR EACH – wyzwalane dla każdego modyfikowanego wiersza

Triggery - wyzwalacze

Przykład (sumowanie wartości w zmiennej @sum):

```
CREATE TRIGGER ins_sum BEFORE INSERT ON account
       FOR EACH ROW SET @sum = @sum + NEW.amount;
```

lub

```
DROP TRIGGER set_data_zatrudnienia;
delimiter //
CREATE TRIGGER set_data_zatrudnienia BEFORE UPDATE ON straznik
FOR EACH ROW
 BEGIN
   SET NEW.data_zatrudnienia = now();
 END;
//delimiter ;
```

Triggery – wady

- Wprowadzają działanie, którego nie widać na pierwszy rzut oka
- Ustalenie kolejności wywołania triggerów może być utrudnione
- Mogą być niewydajne

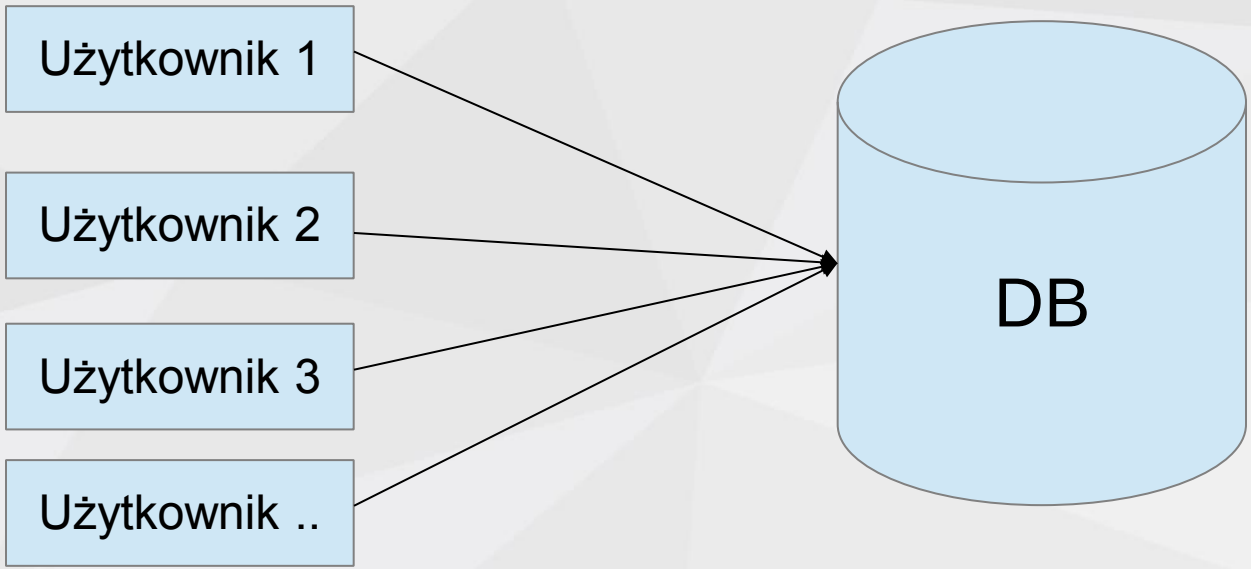
Wybieram stored procedures zamiast triggerów.

Triggery

Ćwiczenie

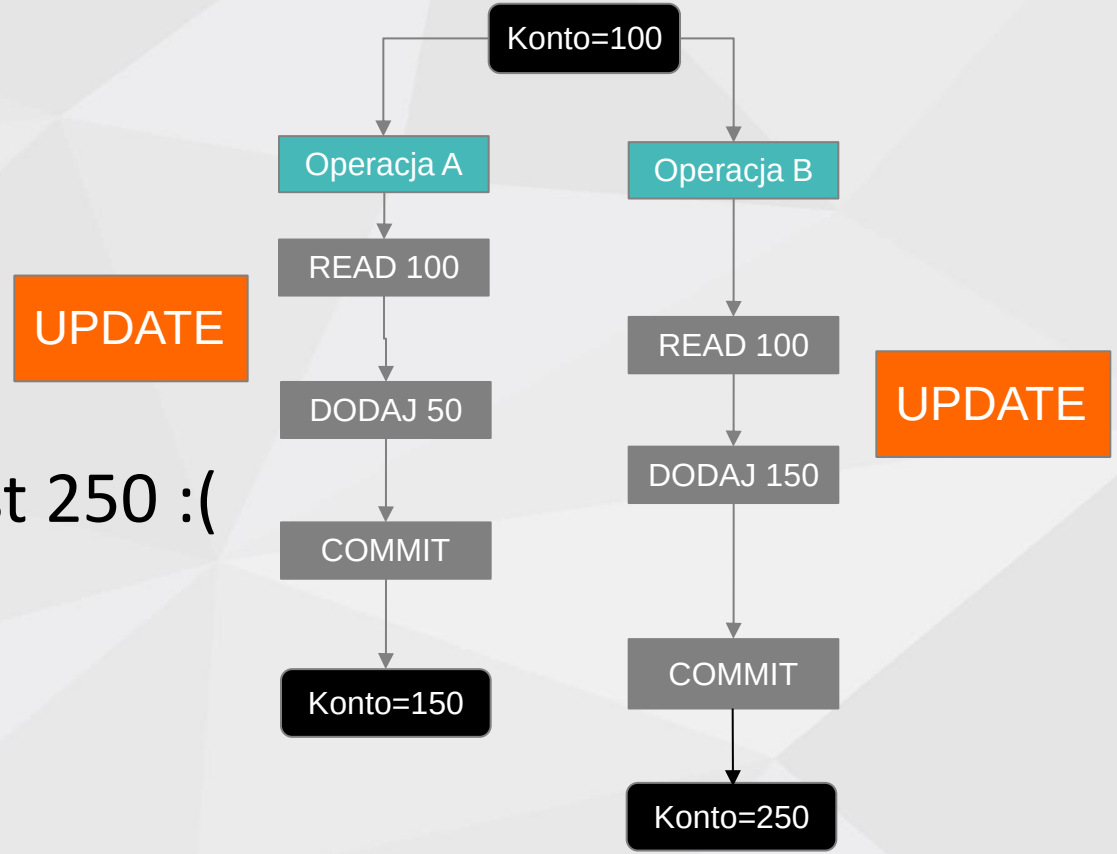
Transakcje

Transakcje



Co może się zdarzyć...

Powinno być 300zł a jest 250 :(



Przykład cd..

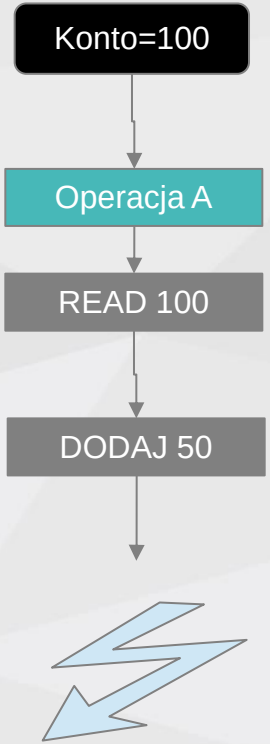
UPDATE kontrola.straznik SET pensja=pensja+10

..w tym samym czasie

*UPDATE kontrola.straznik SET pensja=pensja+pensja*0.10*

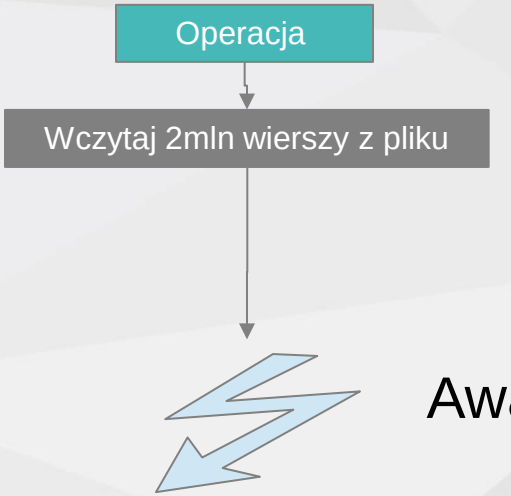
Jaki będzie wynik, czy kolejność wykonania operacji taka sama dla wszystkich wierszy?

Jaka jest wartość konta??



Awaria

Czy dane zostały wczytane?



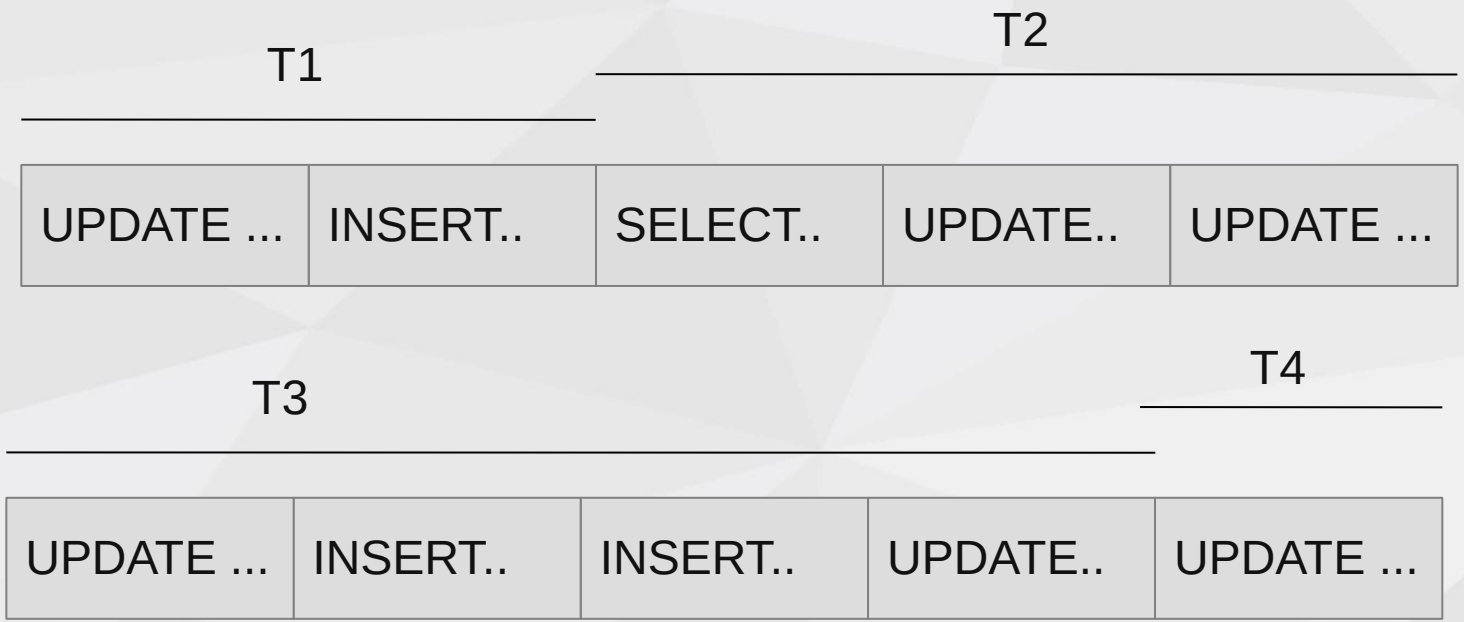
Awaria

Jak zdefiniować transakcje?

Definicja

Transakcja - zbiór operacji na systemie , które stanowią w istocie pewną całość i jako takie powinny być wykonane wszystkie lub żadna z nich. Warunki jakie powinny spełniać transakcje bardziej szczegółowo opisują zasady **ACID**

Definicja

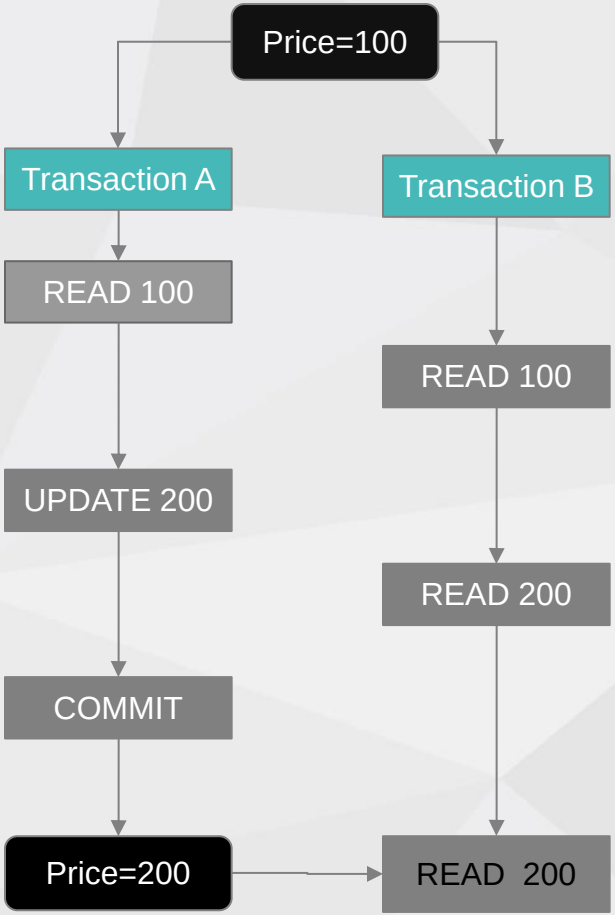

T1
T2
UPDATE ...
INSERT..
SELECT..
UPDATE..
UPDATE ...
T3
T4
UPDATE ...
INSERT..
INSERT..
UPDATE..
UPDATE ...

Jak odszyfrować skrót ACID?

Transaction - ACID

- **Atomicity** - każda transakcja albo wykona się w całości, albo w ogóle
- **Consistency** - oznacza, że po wykonaniu transakcji system będzie spójny, czyli nie zostaną naruszone żadne zasady integralności np.zapewnienie unikalności klucza unikalnego
- **Isolation** - gwarantuje, że dwie współbieżne operacje nie widzą efektów swojego działania do czasu zatwierdzenia transakcji (czyli operacji COMMIT)
- **Durability** - oznacza, że system potrafi udostępnić spójne i nienaruszone dane zapisane w ramach zatwierdzonych transakcji po awarii spowodowanej np. zanikiem napięcia

Dirty read - sytuacja kiedy transakcja może przeczytać dane zapisane przez inną niezakończoną transakcje



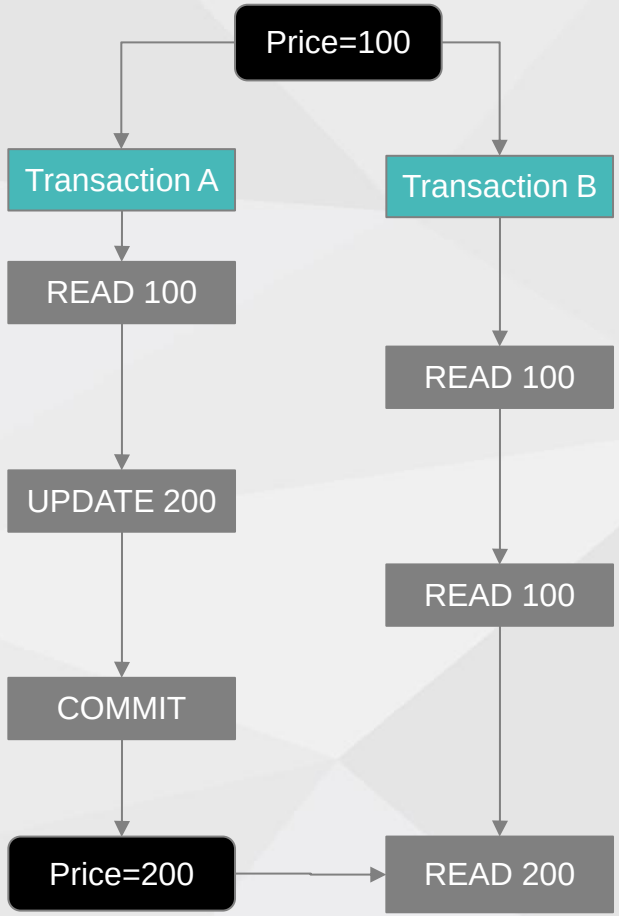
Tranzkacje – Dirty read

UPDATE kontrola.straznik SET pensja=pensja+10

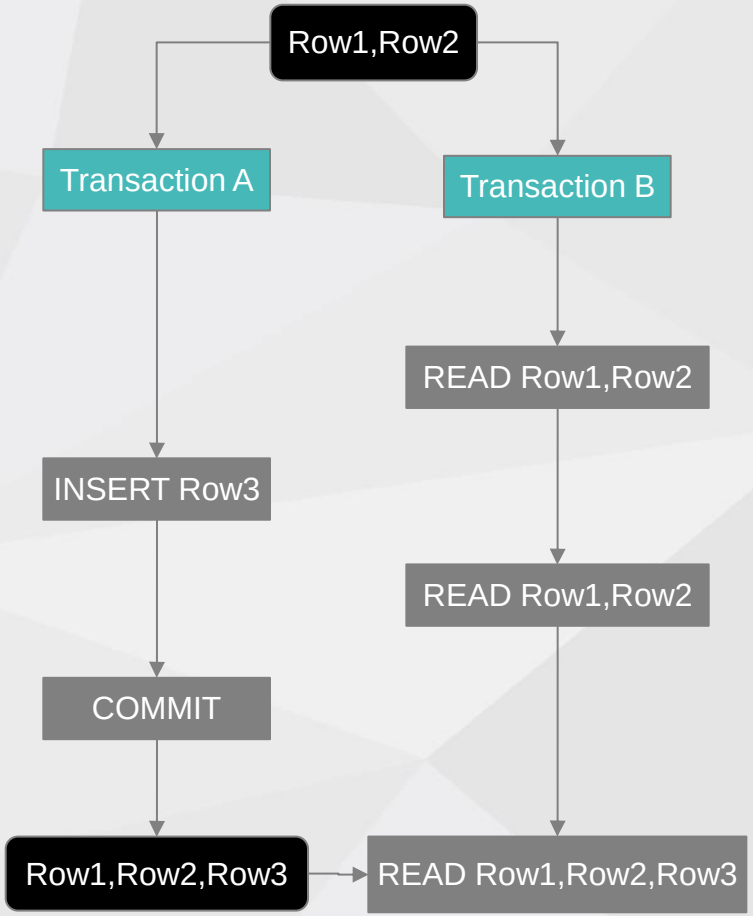
.. i równolegle

SELECT SUM(pensja) FROM kontrola.straznik

Nonrepeatable read - w transakcji ponownie odczytujemy dane, które wcześniej były odczytane w tej samej transakcji i okazuje się, że dane uległy modyfikacji w wyniku innej zakończonej transakcji (czyli były zmodyfikowane między kolejnymi odczytami)



Phantom read - ponowne wykonanie zapytanie z tymi samymi warunkami wyszukiwania zwraca inny zbiór wierszy - inna zakończona transakcja dodała nowe wiersze.



Transakcja – Poziomy izolacji

Poziom	Dirty read	Nonrepeatable read	Phantom read
Read uncommited	+	+	+
Read commited	-	+	+
Repeatable read	-	-	+
Serializable	-	-	-

Dostępne poziomy izolacji w Oracle i Postgres to *serializable, repeatable read* i *read commited* . W MySQL dostępne są wszystkie poziomy przynajmniej w InnoD.B https://blog.pythian.com/understanding-mysql-isolation-levels-repeatable-read/

Transakcja

SET [GLOBAL | SESSION] TRANSACTION

transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic: ISOLATION LEVEL level | READ WRITE | READ ONLY

level: REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE

Blokowanie tabel:

LOCK TABLES <nazwa> <tryb> , ...
Np. LOCK TABLES t1 READ

Blokowanie wiersza

SELECT <kolumny> FROM <tabela> WHERE <warunek> FOR UPDATE

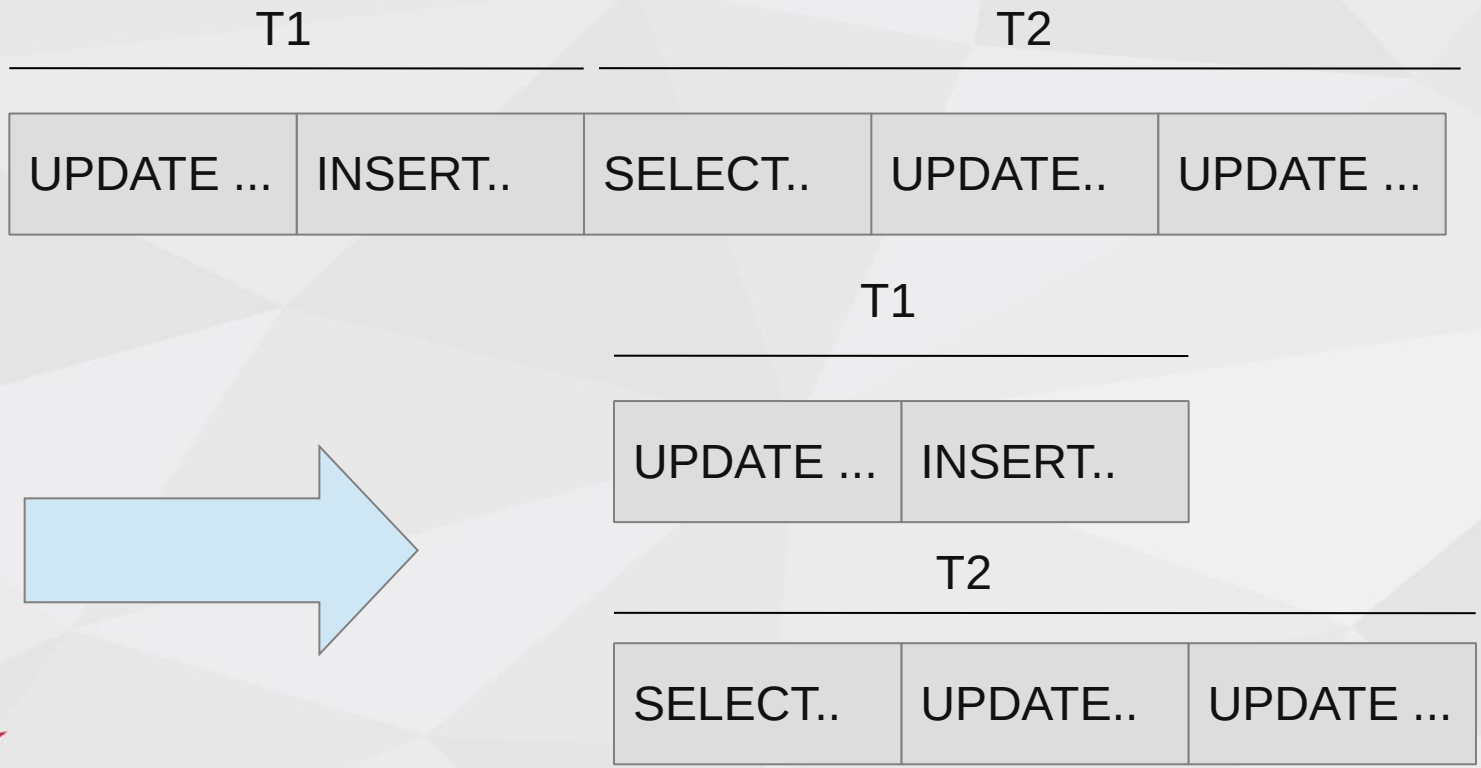
SELECT <kolumny> FROM <tabela> WHERE <warunek> LOCK IN SHARE MODE

Transakcje – Poziomy izolacji

To jaki poziom izolacji jest najlepszy?

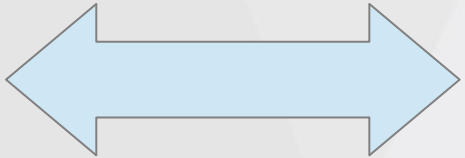
To zależy...

Transakcje – Poziomy izolacji



Transakcje- Poziomy izolacji

Spójność



Wydajność

+ poprawny rezultat odpowiedź
- dużo blokad (lock)
- szeregowe wykonywanie zadań

- nie zawsze poprawny rezultat
+ brak blokad (lock)
+ równoległe przetwarzania - skalowalność

Transakcje – Poziomy izolacji

Dobre praktyki

- Wybierać najmniej restrykcyjny (ale akceptowalny biznesowo) poziom izolacji
- Blokować jak najmniejszą ilość danych np. wiersze zamiast tabeli. Jak nie można używać blokad na wierszach używać semaforów.
- Zakładać blokady na jak najkrótszy czas.
- Używać jak najprostszych metod blokowania zasobów.

Transakcje - SQL

COMMIT - potwierdzenie

ROLLBACK - anulowanie

Transakcje – zarządzenie

Local Transaction Model – programista zarządza operacjami zatwierdzenia (COMMIT) i wycofywania (ROLLBACK). To jest bardziej zarządzanie połączeniami a nie transakcjami np. JDBC.

```
……….
Connection conn = ds.getConnection();
conn.setAuoCommit(false);
try {
     OPERACJA NA BAZIE
     con.commit();
} catch (Exception e) {
     conn.rollback();
throw e;
} finally {
     stmt.close();
     conn.close();
}
```

Transakcje - PHP

```
<?php
try {   $dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2',
         array(PDO::ATTR_PERSISTENT => true));
        echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $dbh->beginTransaction();
  $dbh->exec("insert into staff (id, first, last) values (23, 'Joe', 'Bloggs')");
  $dbh->exec("insert into salarychange (id, amount, changedate)
      values (23, 50000, NOW())");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

Transakcje – zarządzenie

Programmatic Transaction Model – Pozwala on na wywołanie kilkunastu operacji na różnych źródłach danych w obrębie jednej transakcji. Zarządzamy transakcją przy użyciu np. JTA. Przykładowa implementacja Atomikos

```
UserTransactionManager
utm=new UserTransactionManager();
utm.init();
utm.begin();
Connection conn = ds.getConnection();
OPERACJA NA BAZIE
conn.close();
utm.commit();
```

Widok bazodanowy

Widok - logiczny byt udostępniający dane z tabel bazodanowych

Kiedy używać:

- Udostępnienie tylko wybranych kolumn z wybranych tabel
- Uproszczenie zapytań SQL – w widokach 'chowamy' część logiki

Widok bazodanowy

Tworzenie widoku:

CREATE VIEW NazwaWidoku AS <Zapytanie>

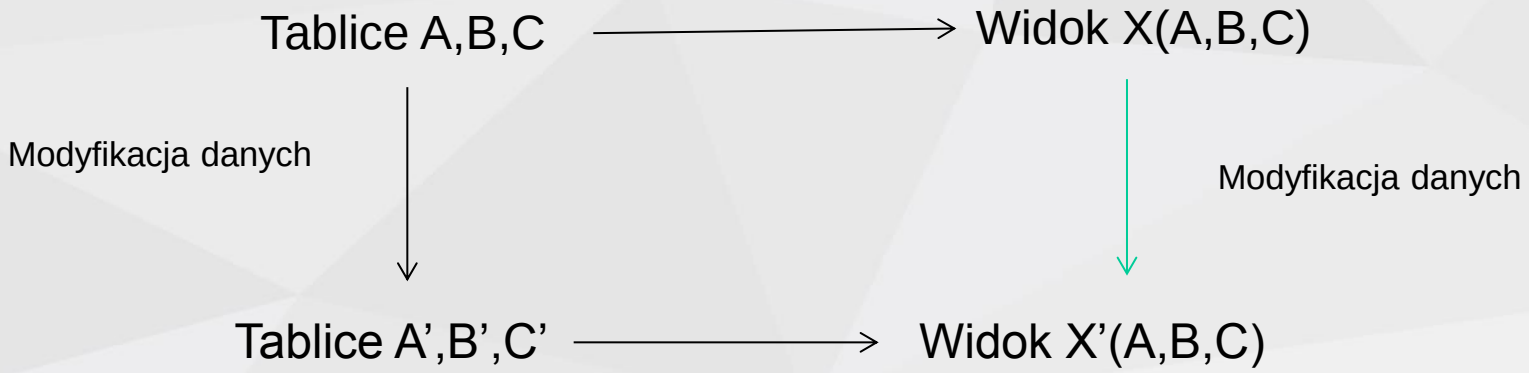
Tworzenie

```
CREATE VIEW KontrolaPasazer AS
  SELECT * FROM kontrola.kontrola k JOIN kontrola.pasazer p
    ON (k.id_pasazer=p.id)
```

Pobieranie danych

*SELECT * FROM KontrolaPasazer WHERE id_pasazer=1*

Widok bazodanowy - modyfikacja



Czyli modyfikując widok musimy znaleźć metodę modyfikujące tablice.
Czy takie przekształcenie jest możliwe?

Widok bazodanowy - modyfikacja

Zwykle jest to możliwe ale może być dużo możliwych rozwiązań !

Przykład:
Tabela Osoba(Imię,Nazwisko) → Widok OsobaSkrot(Imię)

Dodajemy wiersz do widoku z wartością ‚Jan'

Jakie wartości ma przyjąć pole nazwisko w tabeli Osoba ?!

Widok bazodanowy - modyfikacja

Dwie możliwości ‚translacji' :

- Użytkownik definiuje jak będą modyfikowane dane brane do widoku gdy widok zostaje zmodyfikowany → triggers. rules
- Oparcie się na mechanizmach bazy danych → Standard SQL

Widok zmaterializowany

X0	X1	X2
…	…	…

Y0	Y1
…	…

Skomplikowany/
wolny join

X0	X1	X2	Y0	Y1

Widok bazodanowy – zmaterializowany (Postgres, Oracle)

Zalety:

- Możemy uniknąć wykonywania skomplikowanej logiki na etapie korzystania z widoku

Wada

- Dane trzeba odświeżać – trzeba skonstruować mechanizm odświeżania
- Widok może być bardzo duży

CREATE **MATERIALIZED** *VIEW kontrola.PasazerPodroze AS*
SELECT imie, nazwisko, czas_kontroli, nazwa_portu, id_straznik
FROM kontrola.kontrola k JOIN kontrola.pasazer p ON (k.id_pasazer=p.id) JOIN kontrola.numer_stanowiska ns ON (k.id_numer_stanowiska=ns.id);

REFRESH MATERIALIZED VIEW *kontrola.PasazerPodrozeMat*