

# Bazy Danych

## Pytania egzaminacyjne

Hubert Jaremko

2019/2020

# Spis treści

1	Architektury systemów baz danych	3
2	Relacyjny model danych, normalizacja relacji	3
3	Model ER	10
4	Transakcje	11
4.1	Właściwości (ACID)	11
4.2	Harmonogramy, szeregowalność konfliktowa i perspektywiczna	14
4.3	Poziomy izolacji transakcji	18
4.4	Sterowanie współbieżnymi transakcjami w oparciu o blokady	20
4.5	Sterowanie współbieżnymi transakcjami z wykorzystaniem wielo-wersyjności i blokad	20
4.6	Zakleszczenia	24
4.7	Kursory, sterowanie współbieżnością w kursorach	25
5	Język SQL	27
6	Procedury, funkcje i wyzwalacze	27
7	Indeksy, typy indeksów, statystyki, wykorzystanie przez optymalizato-ry kwerend	29
8	Budowa fizyczna baz danych - macierze RAID	30
9	Charakterystyka baz danych NoSQL	35

# 1 Architektury systemów baz danych

## 2 Relacyjny model danych, normalizacja relacji

---

Proszę podać przykład tabeli (relacji), która **jest w trzeciej** postaci normalnej, ale **nie jest** w postaci normalnej Boyce'a Codd'a.

---

Relacja {Miasto, Ulica, Kod}

Klucze:

1. {Miasto, Ulica}
2. {Ulica, Kod}

Dodatkowe zależności:

1. {Kod}  $\rightarrow$  {Miasto}

**Tylko 3NF** - Kod nie jest nadkluczem.

---

Dla tabeli (relacji) z podanymi zależnościami funkcyjnymi proszę powiedzieć w której **postaci normalnej** jest ta tabela.

---

Relacja {CustomerID, Firstname, Surname, Telephone}

Klucze: {CustomerID, Telephone}

Dodatkowe zależności:

1. {CustomerID}  $\rightarrow$  {Firstname, Surname}
2. {Telephone}  $\rightarrow$  {CustomerID}

**Tylko 1NF** - Częściowa funkcyjna zależność atrybutu wtórnego od podzbioru właściwego klucza w zależności nr 2.

Relacja {Tournament, Year, Winner, Winner Date of Birth}

Klucze: {Tournament, Year}

Dodatkowe zależności:

1. {Winner}  $\rightarrow$  {Winner Date of Birth}

**Tylko 2NF** - Atrybut niekluczowy funkcyjnie zależny od innego atrybutu niekluczowego.

Relacja {PESEL, NrPłyty, Od, Do, NrDowodu}

Klucze:

1. {PESEL, NrPłyty, Od}
2. {NrDowodu, NrPłyty, Od}

Dodatkowe zależności:

1. {NrDowodu}  $\rightarrow$  {PESEL}

**Tylko 3NF** - Lewa strona zależności nr 1 nie jest nadkluczem.

Relacja {Restaurant, Pizza Type, Delivery Area}

Klucze: {Restaurant, Pizza Type, Delivery Area}

Dodatkowe zależności:

1. {Restaurant}  $\rightarrow$  {Pizza Type}
2. {Restaurant}  $\rightarrow$  {Delivery Area}

**Tylko BCNF** - Jeśli założymy, że wszystkie typy piz są dowożone na każdy obszar to występują nietrywialne zależności wielowartościowe, których lewa strona nie zawiera klucza.

---

Podaną tabelę należy doprowadzić do postaci normalnej Boyce'a Codd'a.

---

### PRZYKŁAD 1.

Tytuł	Rok	Długość	TypFilmu	NazwaStudia	AdresStudia
Gwiezdne Wojny	1977	124	kolor	Fox	Hollywood
Poteżne Kaczory	1991	104	kolor	Disney	Buena Vista
Świat Wayna	1992	95	kolor	Paramount	Hollywood
Rodzina Adamsów	1991	102	kolor	Paramount	Hollywood

Klucz: {Tytuł, Rok}

Zależność funkcyjna: {NazwaStudia} → {AdresStudia}

Dekomponujemy schemat na dwa zbiory:

{NazwaStudia, AdresStudia}

{NazwaStudia, Tytuł, Rok, Długość, TypFilmu}

### PRZYKŁAD 2.

Person	Shop Type	Nearest Shop
Davidson	Optician	Eagle Eye
Davidson	Hairdresser	Snippets
Wright	Bookshop	Merlin Books
Fuller	Bakery	Doughy's
Fuller	Hairdresser	Sweeney Todd's
Fuller	Optician	Eagle Eye

Klucze: {Person, Shop Type}; {Person, Nearest Shop}

Zależność funkcyjna: {Nearest Shop} → {Shop Type}

Dekomponujemy schemat na dwa zbiory:

{Nearest Shop, Shop Type}

{Nearest Shop, Person}

---

Proszę podać przykład uzasadniający **denormalizację**.

---

### Adresy:







{NrPracownika, Ulica, Miasto, Województwo, KodPocztowy}

Zakładamy następujące zależności funkcyjne:

1. {Ulica, Miasto, Województwo}  $\rightarrow$  {KodPocztowy}
2. {KodPocztowy}  $\rightarrow$  {Miasto, Województwo}

Relacja **Adresy** jest w 2NF, ale nie jest w 3NF.

**Rozwiązanie 1** (wynikające z zależności funkcyjnej 1.)

{Ulica, Miasto, Województwo, KodPocztowy} (\*)  
{NrPracownika, Ulica, Miasto, Województwo} (\*\*)









Relacja (\*) jest w 3NF, ale nie jest BCNF.

Relacja (\*\*) jest w BCNF.




Zostały zachowane zależności funkcyjne, ale jest redundancja.

Tabele będą łączone ze sobą aż przez **trzy** pola (klucz obcy złożony).





Można tego uniknąć poprzez dodanie atrybutu **IdAdresu** do relacji (\*) z zależnością funkcyjną {**IdAdresu**}  $\rightarrow$  {Ulica, Miasto, Województwo} oraz wstawienie **IdAdresu** zamiast {Ulica, Miasto, Województwo} w relacji (\*). Mielibyśmy:

{IdAdresu, Ulica, Miasto, Województwo, KodPocztowy} (tylko 3NF)  
{NrPracownika, IdAdresu} (BCNF)

Wróćmy jednak do rozwiązania bez **IdAdresu**. W pierwszej relacji jest redundancja. Można dekomponować pierwszą relację na dwie, doprowadzając do BCNF:


{Miasto, Województwo, KodPocztowy} oraz  
{Ulica, KodPocztowy}




Ostatecznie otrzymamy trzy relacje:


{Miasto, Województwo, KodPocztowy}  
{Ulica, KodPocztowy}  
{NrPracownika, Ulica, Miasto, Województwo}

"Zgubiliśmy" zależność funkcyjną nr 1. Ponadto uzyskanie kodu pocztowego pracownika wymaga złączenia trzech tabel.

W przypadku wersji z atrybutem **IdAdresu** otrzymalibyśmy:


{Miasto, Województwo, KodPocztowy}


{IdAdresu, Ulica, KodPocztowy}

{NrPracownika, IdAdresu}

"Zgubiliśmy" zależność funkcyjną nr 1. Uzyskanie miasta i województwa pracownika wymaga złączenia trzech tabel.

**Rozwiązanie 2** (wynikające z zależności funkcyjnej 1.)

{KodPocztowy, Miasto, Województwo}

{NrPracownika, Ulica, KodPocztowy}

Obie relacje są w BCNF.

Obydwie wcześniej przedstawione **zależności funkcyjne zostały "zgubione"**. Połączenie między tabelami będzie realizowane przez jedno pole.

Obydwa rozwiązania mają swoje wady i zalety. Zalety i wady ma również wyjściowa relacja **Adresy** (choć jest tylko w 2NF, nie jest w 3NF).

---

Proszę podać przykład tabeli, która jest w postaci normalnej **Boyce'a Codd'a**, a w której jest **redundancja**.

---

Nr Pracownika	Język Programowania	Język Obcy
1	C	niemiecki
1	Java	angielski
2	C	francuski
2	Java	niemiecki
2		hiszpański

Klucz:{Nr Pracownika, Język Programowania, Język Obcy}

Zależności:

1. {Nr Pracownika}  $\rightarrow$  {Język Programowania}
2. {Nr Pracownika}  $\rightarrow$  {Język Obcy}

**Tylko BCNF** - Występują zależności wielowartościowe, których lewa strona nie zawiera klucza.

---

Proszę podać przykład tabeli, która jest w **5NF**, ale jest w niej **redundancja**.

---

Producent	Wyrób	Część
1	1	1
1	2	1

Klucz: {Producent, Wyrób, Część}

**5CNF** - Zakładamy, że **Producent** może produkować daną część tylko do jednego wyrobu, nawet jeśli mogłaby być użyta w innym. Zatem **brak zależności wielowartościowych oraz klucz obejmuje wszystkie atrybuty** a wciąż jest redundancja (powtarzamy, że **Producent** 1 produkuje Część 1).



---

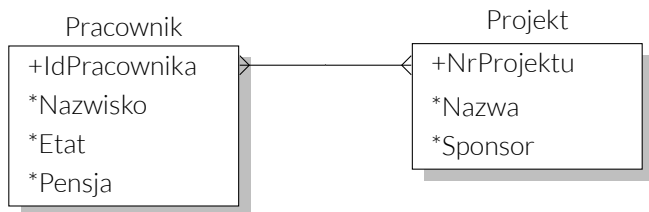
Proszę przedstawić algorytm doprowadzenia relacji do **postaci normalnej Boyce'a Codd'a**.

---

- Szukamy wszystkich nietrywialnych, pełnych zależności funkcyjnych, które naruszają warunek BCNF, tzn. **lewa strona** zależności funkcyjnej **nie jest nadkluczem**.
- Bierzemy jedną z takich zależności funkcyjnych  $A \rightarrow B$  (obojętnie którą, algorytm jest niedeterministyczny).
- Dzielimy schemat relacji na dwa nierozłączne podzbiory: jeden zawierający wszystkie atrybuty występujące w zależności (\*) naruszającej BCNF, drugi zawierający atrybuty z lewej strony rozważanej zależności (\*) oraz atrybuty nie występujące ani z lewej ani z prawej strony tej zależności.
- Strategie stosujemy do relacji powstałych w wyniku dekompozycji do chwili, gdy wszystkie relacje są w BCNF.

### 3 Model ER

Proszę przedstawić przykład diagramu **ER** w notacji **Barkera**, zawierającego dwie encje i związek między nimi (związek bez własnych atrybutów). Diagram powinien być taki, że po jego transformacji do modelu relacyjnego **otrzymamy trzy relacje** (trzy tabele).



Po transformacji diagramu zostaną utworzone trzy tabele.

Pracownicy
# IdPracownika
* Nazwisko
* Etat
* Pensja

Projekty
# NrProjektu
* Nazwa
* Sponsor

Pracownicy_Projekty
# IdPracownika REFERENCES Pracownicy(IdPracownika)
# NrProjektu REFERENCES Projekty(NrProjektu)
PRIMARY KEY (IdPracownika, NrProjektu)

Proszę omówić trzy schematy **transformacji hierarchii** encji do modelu relacyjnego.

1. Utworzenie **jednej tabeli ze wszystkimi** atrybutami i kluczami obcymi, tj. wspólnymi i specyficznymi dla podencji.
2. Utworzenie **osobnej tabeli dla każdej podencji**.
3. Utworzenie **osobnej tabeli na atrybuty wspólne i osobnej tabeli dla każdej podencji**.

Tabele powstałe z podencji zawierają klucz podstawowy i atrybuty specyficzne, tabela wspólna i tabele powstałe z podencji są połączone ograniczeniami referencyjnymi.

## 4 Transakcje

### 4.1 Właściwości (ACID)

---

Proszę omówić własności **ACID** transakcji. W jaki sposób implementowane są własności **A i D**? Proszę podać jak wykorzystywany jest **dziennik transakcji** oraz co to jest strategia **No-Fix/No-Flush** i jak wpływa ona na sposoby odtwarzania systemu po awarii.

---

#### WŁASNOŚCI ACID

- **Atomicity** (*atomowość, niepodzielność*)
  - Transakcja jest niepodzielną jednostką przetwarzania, musi być wykonywana w całości lub wcale.
- **Consistency** (*spójność*)
  - Po wykonaniu transakcji baza danych musi być w stanie spójnym, tj. muszą zostać zachowane wszystkie więzy narzucone na dane.
- **Isolation** (*separacja, izolacja*)
  - Transakcja powinna wyglądać tak, jakby była wykonywana w izolacji od innych transakcji.
- **Durability** (*trwałość*)
  - Po zatwierdzeniu transakcji jej efekty muszą być trwałe w systemie, nawet jeśli nastąpi uszkodzenie systemu natychmiast po zatwierdzeniu.

#### IMPLEMENTACJA A I D

- Za odtwarzanie i w pewnym sensie za **atomowość, trwałość** oraz spójność, jest odpowiedzialny **moduł zarządzania odtwarzaniem** (*recovery-management component*).
- Modyfikacja danych następuje w buforze w pamięci RAM. Buforem zarządza specjalny menadżer (*zarządcą*). W pewnym momencie zarządcą bufora musi skopiować nową zawartość bloku z powrotem na dysk.

## STRATEGIA NO-FIX/NO-FLUSH

- Stosowana w większości relacyjnych systemów baz danych wykorzystujących dzienniki transakcji.
- **NO-FIX**
  - Blok skopiowany do RAM **może** być skopiowany lub przeniesiony z powrotem na dysk zanim transakcja, która ten blok zmodyfikowała się skończy.
- **NO-FLUSH**
  - Na końcu transakcji **nie ma obowiązku** zsynchronizowania zmienionych przez tę transakcję bloków z dyskiem.
  - Synchronizacja może być wykonana później.
  - Zwiększa wydajność.
  - Na tradycyjnych dyskach HDD operacje kopiowania bloków mogą być grupowane.

## WPŁYW NA SPOSOBY ODTWARZANIA SYSTEMU PO AWARII

- **No-Flush** oznacza, że nie mamy gwarancji, że natychmiast po zakończeniu transakcji na dysku znajdują się zmienione dane.
- Trwałość transakcji w większości systemów zapewniają **dzienniki transakcji**.
- **No-Fix** oznacza, że może się zdarzyć, że blok zmieniony przez transakcję zostanie skopiowany na dysk, zanim transakcja zakończy się.
- Synchronizacja buforów z RAM z dyskiem może być realizowane cyklicznie w ramach **punktów kontrolnych** (*control point, check point*).
- **Punkty kontrolne** ułatwiają **odtworzenie systemu po awarii**, kiedy dyski z danymi i z dziennikiem transakcji nie zostały uszkodzone.

- Odtwarzanie po awarii systemu (RECOVERY)
  - redo
  - undo
- Odtwarzanie po awarii dysków z danymi
  - RESTORE (*przywracanie plików z kopii zapasowych*)
  - RECOVERY

## STOSOWANIE DZIENNIKA TRANSAKCJI

- Dziennik transakcji jest plikiem na dysku, zawierającym **informację o wszystkich wprowadzonych przez transakcję zmianach**.
- Transakcja **nie jest uznana** za zakończoną, jeśli fizycznie na dysku w pliku dziennika nie znajdują się **wpisy opisujące wszystkie zmiany oraz informacja o zatwierdzeniu** transakcji.
- Wpis (rekord) w dzienniku może zawierać znacznik transakcji, starą i nową wartość zmienianego elementu, informację o rodzaju operacji, może zawierać informację o tzw. operacji kompensującej. Są też wpisy dotyczące rozpoczęcia transakcji i jej zatwierdzenia, w pewnych systemach także wpisy dotyczące operacji odczytu.
- Dzięki dziennikowi można powtórzyć te operacje, których efekty nie zostały jeszcze zapisane na dysku, mimo że operacja została zatwierdzona (no-flush).
- W przypadku odtwarzania po awarii może być wymagane **powtórzenie** (redo) niektórych operacji (zatwierdzone, ze względu na **No-Flush** mogły jeszcze nie zostać zapisane) i **wycofanie** (undo) innych (niezatwierdzone, ze względu na **No-Fix** mogły już się zapisać).

## 4.2 Harmonogramy, szeregowalność konfliktowa i perspektywiczna

---

Proszę podać definicje **harmonogramu szeregowalnego, szeregowalnego konfliktowo i szeregowalnego perspektywicznie**. Jakie znaczenie w praktyce ma pojęcie **szeregowalności konfliktowej**?

---

### HARMONOGRAM SZEREGOWALNY

- Jeśli jego wpływ na stan bazy danych jest taki sam jak pewnego harmonogramu szeregowego, **niezależnie od stanu początkowego tej bazy danych**.
- Harmonogramy są **równoważne co do wyniku** (*result equivalent*), jeżeli dają ten sam stan bazy danych bez względu na początkowy stan bazy.

### HARMONOGRAM SZEREGOWALNY KONFLIKTOWO (*conflict serializable*)

- Harmonogramy są **równoważne konfliktowo** (*conflict equivalent*) jeżeli kolejność wszystkich operacji konfliktowych jest w nich taka sama.
- Dwie operacje są w stanie **konfliktu**, jeśli:
  - Należą do różnych transakcji.
  - Uzyskują dostęp do tego samego elementu.
  - Przynajmniej jedna z nich jest operacją zapisu.
- Harmonogram **S** jest szeregowalny konfliktowo, jeżeli jest on równoważny konfliktowo z pewnym szeregowym harmonogramem **S'**.
- W takim przypadku możemy zamieniać kolejność niekonfliktowych operacji w **S** do momentu, aż utworzony zostanie równoważny harmonogram szeregowy **S'**.

## HARMONOGRAM SZEREGOWALNY PERSPEKTYWICZNIE

(view serializability)

- Jest on **równoważny perspektywicznie** pewnemu harmonogramowi szeregowemu.
- **Równoważność perspektywiczna:**
  - Harmonogram **S** i **S'** zawierają te same instrukcje i dla każdego elementu danych **Q**:

S	S'
$T_k$ jest pierwszą transakcją, która czyta <b>Q</b>	$T_k$ musi być pierwszą transakcją, która czyta <b>Q</b>
$T_i$ czyta <b>Q</b> zapisany przez $T_j$	$T_i$ czyta <b>Q</b> zapisany przez $T_j$
$T_m$ jest ostatnią transakcją, która zapisuje <b>Q</b>	$T_m$ jest ostatnią transakcją, która zapisuje <b>Q</b>

- Oznacza to samo co **szeregowalność konfliktowa**, jeśli założymy ograniczenie co do operacji **zapisów** we wszystkich transakcjach harmonogramu.
  - Każda operacja **WRITE[x]** jest poprzedzona operacją **READ[x]**
  - Wartość zapisana przez **WRITE[x]** zależy tylko od wartości elementu **x** odczytanej przez operację **READ[x]** (*jest pewną nie stałą funkcją tylko elementu x, nie zależy od wartości innych elementów*).
- Szeregowalność perspektywiczna zapewnia **spójność** bazy danych, ponieważ powoduje, że wyniki harmonogramu są takie same jak wyniki pewnego harmonogramu szeregowego.

## ZNACZENIE W PRAKTYCE SZEREGOWALNOŚCI KONFLIKTOWEJ

- Zapewnia **spójność** bazy danych.

---

Proszę podać przykłady harmonogramów **szeregowalnych** nie szeregowych.

---

$T_1$	$T_2$
<pre>read(A) A := A - 50 write(A)  read(B) B := B + 50 write(B)</pre>	<pre>read(A) temp := A * 0.1 A := A - temp write(A)  read(B) B := B + temp write(B)</pre>



Proszę podać przykłady harmonogramów **szeregowalnych konfliktowo** i takich, które **nie są** szeregowalne konfliktowo.

### HARMONOGRAMY SZERGOWALNE KONFLIKTOWO

$T_1$	$T_2$
<code>read(A)</code> <code>write(A)</code>  <code>read(B)</code> <code>write(B)</code>	<code>read(A)</code> <code>write(A)</code>  <code>read(B)</code> <code>write(B)</code>

### HARMONOGRAMY NIESZERGOWALNE KONFLIKTOWO

$T_1$	$T_2$
<code>read(A)</code> <code>A := A - 50</code> <code>write(A)</code>  <code>read(B)</code> <code>B := 5 + 50</code> <code>write(B)</code>	<code>read(B)</code> <code>B := 5 - 10</code> <code>write(B)</code>  <code>read(A)</code> <code>A := A + 10</code> <code>write(A)</code>

## 4.3 Poziomy izolacji transakcji

Proszę omówić **poziom izolacji** transakcji wybrany przez egzaminatora.

Poziom izolacji	P0 Dirty Write	P1 Dirty Read	P2 Non-repeatable read	P3 Phantoms	Blokady X	Blokady S
Locking READ UNCOMMITTED	NIE	TAK	TAK	TAK	TAK, długie	Nie ma
Locking READ COMMITTED	NIE	NIE	TAK	TAK	TAK, długie	TAK, krótkie
Locking REPEATABLE READ	NIE	NIE	NIE	TAK	TAK, długie	TAK, długie
Locking SERIALIZABLE	NIE	NIE	NIE	NIE	TAK, długie	TAK, długie, predykatowe

### CURSOR STABILITY

- READ COMMITTED « Cursor Stability « REPEATABLE READ
- Pewne rozszerzenie Locking READ COMMITTED.
- Dodaje się operacje **READ\_CURSOR** (*pobierz wiersz*) dla instrukcji **FETCH**.
- Blokada (**s** lub nowy typ do odczytu *scroll lock*) będzie utrzymywana do chwili przejścia do innego wiersza lub zamknięcia kursora.
- Aktualizacja wiersza przez kursor - operacja **WRITE\_CURSOR** powoduje założenie na ten wiersz **blokady x** utrzymywanej do końca transakcji.
- Eliminuje problem P4C.

### SNAPSHOT ISOLATION (*First-commiter-wins*)

- Transakcja czyta dane (zatwierdzone) z chwili swojego początku, *Start-Timestamp*.
- Wszelkie zmiany wykonywane są na lokalnych kopiach i zapisywane na końcu transakcji.

- Aktualizacje wykonywane przez inne transakcje nie są odczytywane.
- Jeśli  $T_1$  jest gotowa do zatwierdzenia, otrzymuje  $Commit-Timestamp(T_1)$ , większy od wszystkich znaczników  $Start-Timestamp$  i  $Commit-Timestamp$  rozpoczętych i zakończonych już transakcji.
- Transakcja zostaje zatwierdzona tylko wówczas, jeśli żadna inna  $T_2$  z czasem zakończenia  $Commit-Timestamp(T_2)$  zawartym w przedziale  $[Start-Timestamp(T_1), Commit-Timestamp(T_1)]$  nie zapisała danych, które zapisała również  $T_1$ .
- W przeciwnym wypadku  $T_1$  zostaje wycofana.
- **First-commiter-wins** zapobiega lost update (P4).
- Po zatwierdzeniu  $T_1$ , zmiany wykonane przez nią są widoczne przez wszystkie inne transakcje o czasie rozpoczęcia większym od  $Commit-Timestamp$  transakcji  $T_1$ .

#### SNAPSHOT ISOLATION (*First-writer-wins*)

- Podobnie jak wcześniej, ale są stosowane **blokady do zapisu**.
- Przy każdym zapisie transakcja wykonuje podobne sprawdzenie jak *First-commiter-wins* na końcu transakcji.
- Przechowywane są różne wersje danych.
- Transakcja czyta dane zatwierdzone przed początkiem transakcji.
- **Brak blokad do odczytu**, operacja odczytu nie blokuje zapisu ani innych operacji odczytu.
- **Długotrwałe blokady wyłącznie do zapisu**.
- $T_1$  przy każdym zapisie sprawdza, czy istnieje  $T_2$ , która zmodyfikowała dane zapisywane i zakończyła się powodzeniem.
- Jeśli tak,  $T_1$  jest wycofywana.
- W systemie MS SQL Server tak działa poziom SNAPSHOT.

- 4.4 Sterowanie współbieżnymi transakcjami w oparciu o blokady
- 4.5 Sterowanie współbieżnymi transakcjami z wykorzystaniem wielowersyjności i blokad

---

Dla przedstawionego harmonogramu proszę podać jak będzie wyglądać sterowanie współbieżnością w wybranym przez egzaminatora poziomie izolacji transakcji.

---

---

Proszę omówić wybrane przez egzaminatora **problemy** związane ze współbieżnym wykonaniem transakcji.

---

### P0 Dirty Write (*nadpisanie brudnopisu*)

- Transakcja  $T_1$  modyfikuje daną.
- Transakcja  $T_2$  dalej modyfikuje daną **zanim**  $T_1$  zostanie **zatwierdzona lub wycofana**.
- WRITE\_1[x]
- WRITE\_2[x]
- (COMMIT\_1[x] lub ABORT\_1[x]) i (COMMIT\_2[x] lub ABORT\_2[x]) w dowolnej kolejności.

### P1 Dirty Read (*czytanie brudnopisu*)

- Transakcja  $T_1$  modyfikuje daną.
- Transakcja  $T_2$  czyta daną **zanim**  $T_1$  zostanie zatwierdzona lub wycofana.
- **Jeśli**  $T_1$  zostaje **wycofana**  $T_2$  przeczytało daną, która nie została zatwierdzona czyli w sumie nigdy nie istniała.
- WRITE\_1[x]
- READ\_2[x]
- (COMMIT\_1[x] lub ABORT\_1[x]) i (COMMIT\_2[x] lub ABORT\_2[x]) w dowolnej kolejności.

### P2 Non-repeatable read

- Transakcja  $T_1$  czyta daną.
- Transakcja  $T_2$  modyfikuje lub usuwa daną oraz zostaje **zatwierdzona**.
- Jeśli  $T_1$  spróbuje znowu przeczytać daną, otrzyma zmodyfikowaną wartość albo odkryje, że dana została skasowana.
- READ\_1[x]
- WRITE\_2[x]

- (COMMIT\_1[x] lub ABORT\_1[x]) i (COMMIT\_2[x] lub ABORT\_2[x]) w dowolnej kolejności.

### P3 Phantoms

- Transakcja  $T_1$  czyta zestaw danych spełniający jakiś *predykat*.
- Transakcja  $T_2$  **tworzy daną** spełniającą ten *predykat* i zostaje zatwierdzona.
- Jeśli  $T_1$  spróbuje znowu przeczytać zestaw danych z tym samym *predykatem* otrzyma zestaw danych **inny od pierwotnego**.
- READ\_1[P]
- WRITE\_2[y in P]
- (COMMIT\_1[x] lub ABORT\_1[x]) i (COMMIT\_2[x] lub ABORT\_2[x]) w dowolnej kolejności.

### P4 Lost Update

- Transakcja  $T_1$  czyta element danych.
- Transakcja  $T_2$  aktualizuje ten element i **zostaje zatwierdzona**.
- Transakcja  $T_1$  aktualizuje ten sam element i **zostaje zatwierdzona**.
- READ\_1[x]
- WRITE\_2[x]
- COMMIT\_2
- WRITE\_1[x]
- COMMIT\_1

### P4C Lost Update (dla operacji na kursorze)

- READ\_CURSOR\_1[x]
- WRITE\_2[x]
- COMMIT\_2
- WRITE\_CURSOR\_1[x]
- COMMIT\_1

### A5A Read Skew (*skrzywiony odczyt*)

- Transakcja  $T_1$  odczytuje  $x$ .
- Transakcja  $T_2$  aktualizuje  $x$  oraz  $y$  i zostaje **zatwierdzona**.
- Jeśli  $T_1$  odczyta  $y$  to będzie miała niespójny obraz danych.
- READ\_1[ $x$ ]
- WRITE\_2[ $x$ ]
- WRITE\_2[ $y$ ]
- COMMIT\_2
- READ\_1[ $y$ ]
- (COMMIT\_1 lub ABORT\_1)

### A5B Write Skew (*skrzywiony zapis*)

Załóżmy, że na elementy danych  $x$  oraz  $y$  narzucono pewne ograniczenie  $C()$ . Każda transakcja z osobna dba o spełnienie  $C()$ .

- Transakcja  $T_1$  odczytuje  $x$  (ew. też  $y$ ).
- Transakcja  $T_2$  odczytuje  $y$  (ew. też  $x$ ).
- $T_1$  zapisuje  $y$  a  $T_2$  zapisuje  $x$  i **obydwie zostają zatwierdzone**.
- Ostatnie cztery operacje mogą być zrealizowane w dowolnej (sensownej) kolejności.
- Każda transakcja przy zapisie dba o spełnienie ograniczenia  $C()$ , jednak w wyniku przeplatanego wykonania ograniczenie  $C()$  może nie być spełnione po zatwierdzeniu obydwu transakcji.
- READ\_1[ $x$ ]
- READ\_2[ $y$ ]
- WRITE\_1[ $y$ ], WRITE\_2[ $x$ ], COMMIT\_1, COMMIT\_2 w dowolnej sensownej kolejności.

## 4.6 Zakleszczenia

Proszę napisać przykładowy harmonogram, który doprowadzi do zakleszczenia. Jak mogą być wykrywane zakleszczenia?

$T_1$	$T_2$
read(K) (zakłada blokadę S na K)	
write(K) (żąda blokady X na K)	read(K) (zakłada blokadę S na K)
wait	write(K) (żąda blokady X na K)
wait	wait
wait	wait

### WYKRYWANIE ZAKLESZCZEŃ

- **GRAF OCZEKIWANIA** (*wait-for graph*)
  - Każda wykonywana transakcja jest wierzchołkiem w grafie.
  - Jeśli transakcja  $T_i$  próbuje zablokować element danych, który **jest już blokowany** przez inną transakcję  $T_k$  z użyciem **konfliktowej blokady**, w grafie tworzona jest krawędź skierowana z  $T_i$  do  $T_k$ .
  - Po zwolnieniu blokady krawędź jest usuwana.
  - **Cykl w grafie oznacza zakleszczenie.**
  - Wybór ofiary - na ofiarę można wybrać transakcję młodszą, lub tę, która mniej zmodyfikowała (tę, której wycofanie jest prostsze).
- **LIMITY CZASU** (*timeouts*)
  - Jeśli transakcja czeka na zasób **dłużej niż** przyjęta **wartość progowa**, to system przyjmuje, że uległa zakleszczeniu i **anuluje ją**, bez względu na to czy zakleszczenie rzeczywiście wystąpiło, czy nie.



## 4.7 Kursory, sterowanie współbieżnością w kursorach

Proszę omówić, jak wygląda sterowanie współbieżnością w kursorach w systemie Microsoft SQL Server.

### OPCJE PRZY OTWIERANIU KURSORA

- READ\_ONLY
  - Nie można wykonywać pozycjonowanych zmian wierszy przez kursor.  
**UPDATE ... WHERE CURRENT OF** CursorName
  - Blokady nie są zakładane.
- SCROLL LOCKS
  - Kursor jest otwarty w transakcji **jawnej**:
    - \* Zakładane długotrwałe blokady u (*update*) i blokady kursora (*scroll locks*).
    - \* Blokady są zwalniane w momencie przejścia do innego wiersza.
  - Kursor jest otwarty **poza transakcją**:
    - \* Zakładane są tylko blokady kursora.
  - Niewłączona opcja **automatycznego zamykania** kursorów na końcu transakcji może sprawić, że **blokady są trzymane** nadal po zakończeniu transakcji.  
**SET** CURSOR\_CLOSE\_ON\_COMMIT **ON**  
**ALTER DATABASE SET** CURSOR\_CLOSE\_ON\_COMMIT
- OPTIMISTIC (WITH VALUES)
  - Przy **odczyt**ie wiersza **nie są zakładane** blokady.
  - Przy próbie modyfikacji wiersza następuje **sprawdzenie, czy inna transakcja tego nie zrobiła** (*już po odczycie przez kursor, ale przed próbą modyfikacji*).
  - Wiersz wczytywany jest jeszcze raz i **porównywane są wartości** w kolumnach.
  - Jeśli się **nie zmieniły**, to aktualizacja następuje, jeśli nie, zgłaszany jest **błąd**.

- OPTIMISTIC (WITH ROW VERSIONING)

- Podobnie, ale w tabeliu musi być kolumna typu **rowversion** (w MSSQL 2000 i 2005 *timestamp*).
- Wartość w takiej kolumnie jest zawsze automatycznie modyfikowana przy modyfikacji wiersza, nawet jeśli jest to modyfikacja typu **pole1 = pole1**.

## 5 Język SQL

---

Proszę napisać zdanie w języku SQL, które zrealizuje cel podany przez egzaminatora.

---

**Ważne:** kolejność wykonania w klauzul w zapytaniu

5. **SELECT**
1. **FROM**
2. **WHERE**
3. **GROUP BY**
4. **HAVING**
6. **ORDER BY**

## 6 Procedury, funkcje i wyzwalacze

---

Proszę napisać **procedurę** w języku Transact SQL, funkcjonalność procedury poda egzaminator.

---

```
CREATE PROCEDURE ProcName (@Arg TYPE, @OutArg TYPE OUTPUT)  
AS  
BEGIN  
    ...  
END
```

---

Proszę napisać **funkcję skalarną** w języku Transact SQL, opisaną przez egzaminatora.

---

```
CREATE FUNCTION FuncName (@Arg TYPE, @OutArg TYPE OUTPUT)  
RETURNS TYPE  
AS  
BEGIN  
    ...  
END
```

---

Proszę napisać w języku Transact SQL opisaną przez egzaminatora **funkcję** zwracającą zestaw rekordów.

---

```
CREATE FUNCTION ProcName (@Arg TYPE, @OutArg TYPE OUTPUT)  
RETURNS @TableName TABLE (  
    ColumnName1 TYPE,  
    ColumnName2 TYPE  
)  
AS  
BEGIN  
    ...  
END
```

```
-----  
CREATE FUNCTION ProcName (@Arg TYPE, @OutArg TYPE OUTPUT)  
RETURNS TABLE  
AS  
BEGIN  
    RETURN (SELECT ... )  
END
```

---

Proszę napisać w języku Transact SQL **wyzwalacz**, który zrealizuje zadaną przez egzaminatora funkcjonalność.

---

```
CREATE TRIGGER TriggerName  
ON TableName  
AFTER INSERT, UPDATE, DELETE  
--INSTEAD OF  
AS  
BEGIN  
    ...  
END
```

## 7 Indeksy, typy indeksów, statystyki, wykorzystanie przez optymalizatory kwerend

---

Proszę omówić budowę indeksu typu **drzewo B+**. Proszę podać wersje tego indeksu (w systemie Microsoft SQL Server: clustered i non-clustered, w Oracle IOT).

---

## 8 Budowa fizyczna baz danych - macierze RAID

---

Proszę omówić macierze **RAID** (wybrany przez egzaminatora poziom).

---

### RAID 0

- Dane umieszczane są równomiernie na dwóch lub więcej dyskach.
- Jeden dysk: B1, B2, B3, B4, B5, B6, B7, B8, ...
- Cztery dyski (RAID 0) - przykład.

B1	B2	B3	B4
B5	B6	B7	B8
...	...	...	...

- **ZALETY**

- **Szybszy odczyt i zapis** w porównaniu z pojedynczym dyskiem dzięki operacjom równoległym.
- Był używany do połączenia dwóch lub więcej mniejszych dysków w jeden większy logiczny dysk.

- **WADY**

- **Brak odporności na błędy** - nie ma nadmiarowości!
- Niezawodność jest odwrotnie proporcjonalna do liczby dysków w systemie RAID 0.  
Niezawodność dwóch dysków jest połowę mniejsza od niezawodności jednego dysku.
- RAID 0 nie jest polecany w środowiskach podwyższonej odporności na błędy (*mission-critical environments*).

## RAID 1

- *Mirroring* lub *duplexing* (gdy każdy dysk ma osobny kontroler).
- Na ogół zawiera dwa dyski, czasem więcej.
- Obydwa dyski mają taką samą zawartość.

B1	B1
B2	B2
...	...

- **ZALETY**

- **Odczyt** jest prawie dwukrotnie **szybszy** w porównaniu z pojedynczym dyskiem.
- **Odporny na awarie** -  $N$  dysków może przetrwać jednoczesną awarię  $N - 1$  dysków.

- **WADY**

- **Wolniejszy zapis** w porównaniu z pojedynczym dyskiem.  
Dane muszą być zapisane na obydwu dyskach, a na początku operacji głowice są na ogół nad innymi ścieżkami i sektorami.
- Pamięć podręczna powinna być włączona w celu przyspieszenia operacji zapisu.
- Używa efektywnie jedynie 50% całkowitej pojemności.

- **TYPOWE ZASTOSOWANIE**

- **Dziennik transakcji.**
- **System operacyjny.**
- Dziennik transakcji jest zapisywany sekwencyjnie, najlepiej stosować jeden system RAID 1 dla każdego dziennika.
- Dziennik jest jednym z najważniejszych komponentów systemu baz danych, dlatego dysk z dziennikiem powinien być odporny na awarie.

## RAID 5

- Zawiera **trzy** lub więcej dysków.
- Zapisy są wykonywane blokami na wielu dyskach z wykorzystaniem bloków parzystości.
- Bloki mogą być większe niż sektory (np. 256 sektorów).

B1	B2	B3	Parity 1, 2, 3
B5	B6	Parity 5,6,7	B7
B8	Parity 8,9,10	B9	B10
Parity 11,12,13	B11	B12	B13

- **ZALETY**

- **Szybki odczyt.**
- Względnie efektywne wykorzystanie przestrzeni dyskowej.
- **Odporność na błędy.**
  - \* Bloki parzystości są odczytywane jeśli przy odczycie wykryty jest błąd sumy kontrolnej (*CRC error*).
  - \* W takim przypadku pozostałe bloki z paska są automatycznie użyte do odtworzenia informacji w bloku uszkodzonym.
  - \* Podobnie dzieje się przy uszkodzeniu całego dysku.
  - \* Komputer może być "nieświadomy" awarii dysku. System RAID pracuje, chociaż nieco wolniej.

- **WADY**

- **Wolne operacje zapisu.**
  - \* Jeśli blok ma być zapisany, system musi wykonać dwa odczyty i dwa zapisy zamiast jednego zapisu.
  - \* Zmieniany blok i odpowiedni blok parzystości musi być odczytany, trzeba zapisać nową wartość w bloku parzystości.
  - \* Wystarczy znać różnice między starą i nową wartością zmienianego bloku i starą wartość bloku parzystości.
  - \* Na końcu nowy blok i zmieniony blok parzystości muszą być zapisane na dysk.
- Należy włączyć pamięć podręczną jeśli jest podtrzymywanie baterijne. Operacje zapisu mogą być wówczas przyspieszone.



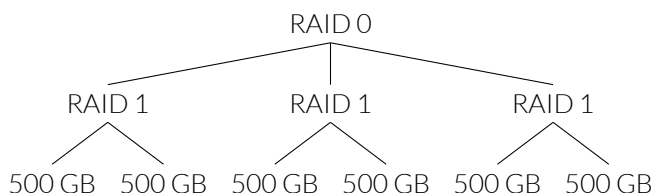
- Wykorzystuje  $\frac{1}{n}$  pojemności dysku na bloki parzystości, gdzie  $n$  oznacza liczbę dysków w systemie.

## • TYPOWE ZASTOSOWANIE

- Systemy, w których **większość operacji to operacje odczytu**.
- **Tablice lub indeksy**, które są tylko do odczytu lub są rzadko modyfikowane.
- RAID 5 nie jest na ogół dobrym rozwiązaniem, jeśli więcej niż 10 procent operacji to operacje zapisu.  
Należy jednak znać wydajność systemu po włączeniu buforowania.

## RAID 10 (1+0)

- Poziomy RAID mogą być zagnieżdżane.  
Jeden system RAID może użyć innego systemu RAID jako elementu składowego zamiast pojedynczego dysku.
- RAID 10 jest systemem z przeplotem z systemami RAID 1 jako elementami składowymi (*a stripe of mirrors*).
- $N - 1$  z **każdego systemu** RAID 1 może ulec awarii bez całkowitej utraty danych.

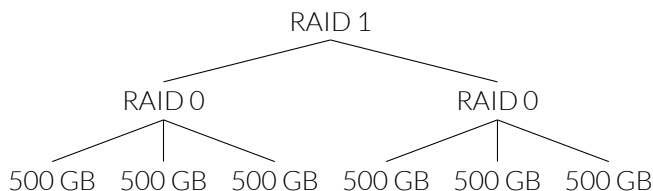


- Ten system może przetwać równoczesną awarię **do trzech** dysków.
- CECHY

- **Najszybszy zapis i odczyt.**
- Odporność na błędy.
- Wykorzystuje efektywnie 50% pojemności całkowitej.
- Najlepszy, ale najdroższy.
- Polecany w systemach baz danych, gdy **liczba operacji zapisu** jest większa niż 10% liczby wszystkich operacji.

## RAID 01 (0+1)

- Różnica między RAID 0+1 i RAID 1+0 stanowi położenie każdego z systemów RAID.
- Jest uważany za **gorszy niż RAID 10**.
- **Nie przetrwa dwóch równoczesnych awarii** jeśli nie są to dyski z tego samego układu RAID.
- Po awarii jednego dysku, wszystkie dyski w drugim pasku stanowią krytyczne punkty. Połowa dysków przestaje być wykorzystywana.  
W systemie RAID 10 jeśli uszkodzeniu ulegnie jeden dysk, tylko jeden staje się krytycznym punktem układu.



## WAŻNE UWAGI

- RAID nie powinien zastąpić odpowiedniej strategii robienia kopii zapasowych.
- Administrator musi wykonywać okresowo kopie zapasowe (pełne, różnicowe, plików, dziennika transakcji).

## 9 Charakterystyka baz danych NoSQL

---

Proszę powiedzieć co oznacza termin skalowanie **poziome** w systemach baz danych i **jak realizowane jest** skalowanie poziome w bazach danych NoSQL.

---

### SKALOWANIE POZIOME

- Jest to dodawanie dodatkowych "węzłów" do systemu, na przykład dodawanie nowego komputera do systemu.

### SKALOWANIE PIONOWE

- Jest to dodawanie dodatkowych zasobów istniejącym "węzłom" w systemie, na przykład dodawanie RAM komputerowi w systemie.

### REALIZACJA SKALOWANIA POZIOMEGO

- Rezygnacja z części funkcjonalności relacyjnych baz danych, brak relacji, transakcji, nie są zachowane własności ACID.

---

Proszę podać główne typy baz danych **NoSQL**.

Proszę podać przykłady dokumentów w formacie **JSON**.

Jak dokumenty są przechowywane w systemach **NoSQL** zawierających wiele węzłów (komputerów) połączonych siecią komputerową?

Czy łączenie danych z różnych dokumentów odbywa się na ogół w bazie danych (jak operacja **JOIN** w bazach relacyjnych), czy w aplikacji?

Czy w bazach **dokumentowych** należy unikać redundancji tak, jak w systemach relacyjnych?

---

### GŁÓWNE TYPY

- Klucz-Wartość
- Hierarchiczna struktura klucz-wartość
- Dokumentowe
- Grafowe

## PRZYKŁAD JSON

---

```
1
2 menu":
3
4   "id": "file",
5   "value": "File",
6   "popup":
7   {
8     "menuitem":
9     [
10      {"value": "New", "onclick": "CreateNewDoc()"},
11      {"value": "Open", "onclick": "OpenDoc()"},
12      {"value": "Close", "onclick": "CloseDoc()"}
13    ]
14  }
15
16
```

---

## PRZECHOWYWANIE DOKUMENTÓW

- Dokumenty rozrzucone po węzłach, każdy z węzłów czyta i zapisuje równolegle.

## ŁĄCZENIE DOKUMENTÓW

- Odbywa się w aplikacji.

## UNIKANIE REDUNDANCJI

- Nie należy unikać redundancji.
- Liczy się szybkość dostępu do danych, często przez wielu użytkowników na raz.