

# Notatki z kursu Bazy Danych 2

Małgorzata Dymek

2019/20, semestr zimowy

## 1 Charakterystyki baz danych.

### 1.1 Bazy danych produkcyjne

- inaczej: **operacyjne**.
- wykorzystywana, gdy istnieje potrzeba nie tylko gromadzenia, ale też modyfikowania danych
- **dane dynamiczne**, tzn. ulegające zmianą i przedstawiające **aktualny stan rzeczy**
- **duża liczba prostych zapytań** od wielu użytkowników
- **zoptymalizowane** pod kątem **szybkiego wyszukiwania**
- np. baza inwentaryzacyjna, baza obsługi zamówień

#### OLTP

- **Online Transaction Processing**
- kategoria **aplikacji klient-serwer** dotyczących baz danych w ramach **bieżącego przetwarzania transakcji**
- klient współpracuje z **serwerem transakcji**, zamiast z serwerem bazy danych.
- np. systemy rezerwacji, obsługa punktów sprzedaży, systemy śledzące itp.

### 1.2 Hurtownie danych

- zorganizowana pod kątem pewnego **wycinka rzeczywistości** (tematycznie spójna)
- wyższy szczebel **abstrakcji** niż zwykła relacyjna baza danych
- dane często pochodzą z **wielu źródeł**
- **zoptymalizowane** pod kątem **szybkości wyszukiwania** i **efektywnej analizy** zawartości.
- wyróżniany jest **poziom danych detalicznych** oraz **warstwa agregatów/kostek** tematycznych.
- korzystanie z danych hurtowni poprzez różne systemy wyszukiwania danych (np. OLAP).
- **brak** zastosowania typowych **transakcji**
- eksploracja danych (**data mining**) wyszukuje **ogólne formy wiedzy** z olbrzymiej ilości danych
- **wyszukiwania** mają najczęściej **charakter wielowymiarowy** – korzystają z wielu relacji.
- **tematycznie hurtownie** danych nazywane minihurtowniami danych (z ang. **data mart**)
- Cele hurtowni

- przetwarzanie analityczne (OLAP)
- wspomaganie decyzji (DSS)
- archiwizacja danych
- analiza efektywności
- wsparcie dla systemów CRM (Customer Relationship Management)

Data warehouse vs Data mart	
Data warehouse	Data mart
Dla całego przedsiębiorstwa	Dla konkretnego działu
Wiele obszarów tematycznych	Jeden konkretny obszar tematyczny
Trudna i czasochłonna do zbudowania	Łatwa i szybka do zbudowania
Duże zapotrzebowanie na pamięć	Małe zapotrzebowanie na pamięć

Dwa **główne podejścia** wykorzystywane przy budowie hurtowni danych to:

- **ETL (Extract, Transform, Load)** - dane są wyciągane z różnych źródeł, następnie transformowane do formatu danych wymaganego przez hurtownię, a na końcu dopiero są ładowane do samej hurtowni
- **ELT (Extract, Load, Transform)** - dane są wyciągane z różnych źródeł, następnie ładowane od razu do hurtowni (do tabel roboczych) gdzie są dopiero transformowane i kopiowane do właściwych tabel hurtowni

#### Architektura

- **Źródło danych** – inne bazy danych (najczęściej relacyjne), różnego rodzaju pliki.
- **Obszar przejściowy** – dane pobrane z systemów źródłowych są oczyszczane i dostosowane do wymagań hurtowni danych (narzędzia ETL)
- **Warstwa metadanych**
  - metadane **biznesowe**: tabele wymiarów, data marta, agregaty, tabele faktów
  - metadane **techniczne**: mapowania i transformacje danych od systemu źródłowego do systemu docelowego
- **Warstwa prezentacji** – warstwa dostępna dla użytkowników końcowych w postaci raportów i analiz; reprezentowana w postaci **data martów**

#### 1.2.1 Znormalizowane vs. wielowymiarowe podejście do gromadzenia danych

**Podejście wielowymiarowe** - model gwiazdy.

- transakcje danych są podzielone albo na poszczególne "fakty", które są generalnie transakcjami numerycznymi, albo "wielowymiarowe", które odnoszą się do kontekstów tych "faktów"
- **zaleta**: hurtownia danych jest prostsza do zrozumienia i użytkowania
- **wady**:
  - skomplikowane utrzymanie porządku i integracji faktów wielowymiarowych
  - trudno jest zmodyfikować hurtownię danych jeżeli przyjmuje się podejście wielowymiarowe zmieniając sposób organizacji danych.

**Podejście znormalizowane** (model 3NF) - model **normalizacyjny** (E-R).

- tabele pogrupowane według ich **tematyki**
- dzieli dane na jednostki, które tworzą kilka tabel w relacyjnej bazie danych

- główną **zaletą** tego podejścia jest to, że **dodawanie** nowych informacji do bazy danych jest bardzo proste
- **wady**:
  - ogromna ilość tabel
  - łączenie danych z różnych źródeł w sensowne informacje jest trudne
  - dostęp do danych wymaga precyzyjnego zrozumienia źródeł danych i ich struktur w hurtowni

### 1.3 Bazy analityczne: OLAP

- **Kostki OLAP (Online Analytical Processing)**, bazy analityczne (archiwalne)
- dane często pochodzą z bazy operacyjnej, ale po wprowadzeniu do bazy analitycznej są **stałe, nie podlegają modyfikacjom**
- główne operacje: **wyszukiwanie**, sporządzanie **zestawień statystycznych**, przeprowadzanie **analiz i prognoz**
- **niewielka liczba zapytań** dotycząca **dużych ilości danych**
- dane przechowywane w sposób przypominający **wielowymiarowe arkusze**; więcej niż trzy wymiary = **hiperkostka**

#### 1.3.1 Budowa kostki

- **Fakty** - pojedyncze **zdarzenia**, które rejestrujemy (np. sprzedaż danego produktu)
  - Każdy fakt ma **miary** - są to **wskaźniki numeryczne** związane z danym faktem (ile?)
  - Oprócz miar w tabeli faktów są **klucze obce do wymiarów** opisujących dany fakt
- **Wymiary** - **cechy** opisujące dany fakt (np. kto, co kiedy?) używane przy filtrowaniu
  - Każdy wymiar ma **atrybuty** - są to po prostu **dane związane z danym wymiarem** (np. dla wymiaru daty sprzedaży będą to numer dnia, numer miesiąca i numer roku)

#### TYPOWE OPERACJE

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• <b>Zwijanie</b> – agregacja, uogólnienie danych,</li> <li>• <b>Rozwijanie</b> – uszczegółowienie danych,</li> <li>• <b>Selekcja</b> – wybór interesujących danych,</li> <li>• <b>Projekcja</b> – zmniejszenie liczby wymiarów,</li> </ul> | <ul style="list-style-type: none"> <li>• <b>Wycinanie</b> – połączenie selekcji z projekcją,</li> <li>• <b>Sortowanie</b> – tworzenie rankingów,</li> <li>• <b>Obracanie</b> – zmiana perspektywy oglądania danych.</li> </ul> |
|--|--|

#### RODZAJE KOSTEK

##### fizyczne

posiadają w swoich komórkach już przeliczone i zgregowane dane

##### wirtualne

powstają z kostek fizycznych, mogą mieć swoje własne miary, wymiary dziedziczą po kostkach fizycznych

#### SCHEMATY BUDOWY KOSTEK

##### gwiazda

- każda tabela wymiarów zawiera pełen zestaw atrybutów opisujących dany wymiar
- może prowadzić do redundancji

##### płatki śniegu

- każda tabela wymiarów zawiera tylko atrybuty specyficzne dla siebie
- może też zawierać klucze obce do kolejnych tabel wymiarów

## 1.4 Jeziora danych

- **ogromna ilość nieprzetworzonych danych w oryginalnym formacie** (strukturalne, częściowo lub zupełnie nie strukturalne)
- dane **łatwodostępne i modyfikowalne**
- każdy element znajdujący się w repozytorium ma przypisany **unikalny identyfikator** i jest oznaczany zestawem znaczników metadanych
- pozwala na maksymalnie szybką, zaawansowaną i kontekstową **analizę danych** nie tylko **historycznych**, ale także tych **generowanych** w czasie rzeczywistym
- dane mogą być np. wykorzystane do późniejszego zasilenia hurtowni danych lub w ogóle nie zostać nigdy użyte
- zazwyczaj wymagają zdecydowanie większej ilości **przestrzeni dyskowej** niż hurtownie danych

## 1.5 Bazy danych NoSQL

- **Not Only SQL**, non SQL, non relational
- łatwo **skalowalne horyzontalnie** (w klastrach i na wielu serwerach)
- przydatne w przypadku danych o **dużym wolumenie**
- celowo **rezygnuje się ze spójności** na rzecz większej wydajności i tolerancji na partycjonowanie
- nieunikanie **redundancji**, jest ona wręcz pożądana
- **główne typy**
  - Klucz-Wartość
  - Hierarchiczna struktura klucz-wartość
  - Dokumentowe
  - Grafowe
  - Kolumnowe
- pozwalają na **szybką analizę** danych niestrukturyzowanych i badanie korelacji pomiędzy nimi
- **brak łączenia (JOIN)** danych po stronie serwera - jeżeli zachodzi taka potrzeba musi to być wykonywane po stronie aplikacji
- obiekty różnego typu: JSON, **BSON**, XML lub inny zbliżony, **przekazywane są wraz z kompletnym ich schematem**.

### ZALETY

- są bardziej przystosowane i **wydajniejsze** przy przetwarzaniu **Big Data**,
- modele danych – **brak predefiniowanych schematów** powoduje ich większą **elastyczność**,
- potrafią przetwarzać **dane niestrukturalne**,
- **tańsze i prostsze** w utrzymaniu (szczególnie w przypadku prostych baz klucz-wartość),
- z natury **skalowalne** (łatwe skalowanie horyzontalne).

### WADY

- dla danych, w których występują **relacje** zalecane są nadal standardowe RDBMS,
- niemożność uniknięcia **redundancji**,
- mniejsza **integralność** i **spójność** danych,
- SQL jest szeroko znany,
- **brak mechanizmów transakcyjnych** – może nie spełniać zasady **ACID** (ang. Atomicity, Consistency, Isolation, Durability)

## 1.6 Big Data

- duże, zmienne i różnorodne **zbiory danych**, których przetwarzanie i analiza jest trudna
- dwie główne grupy:
  - **Ustrukturyzowane** - najczęściej dane wewnętrzne firmy (np. dane ze stacji pogodowych)
  - **Nieustrukturyzowane** - najczęściej dane pobierane z portali społecznościowych lub ogólniej Internetu
- ponieważ zapytania muszą być wykonywane szybko, wszystkie **analizy** wykonuje się **równolegle**

Trzy **podstawowe cechy (3xV)**:

- **Volume** (amount) of data - duża ilość danych
- **Velocity** (speed) of collecting - są one gromadzone z ogromną szybkością
- **Variety** of info - duża różnorodność gromadzonych danych

Najpopularniejszymi **narzędziami** do pomiaru Big Data są:

- platforma **Hadoop**,
- system Storm,
- **magazyny baz danych** – Cassandra, MongoDB czy Neo4j,
- algorytmy do **data-miningu** – RapidMiner i Mahout,

## 2 Poziomy izolacji transakcji.

**Problemy:**

- **P0 (Dirty Write)**:  $T_1$  modyfikuje daną.  $T_2$  modyfikuje tę samą daną zanim  $T_1$  zostanie zaakceptowana (lub anulowana).
- **A1 (Dirty Read)**:  $T_1$  modyfikuje daną.  $T_2$  czyta daną zanim  $T_1$  zostaje zaakceptowana. Jeżeli  $T_1$  zostanie wycofana,  $T_2$  ma odczyt danej która "nigdy nie istniała".
- **A2 (Non-repeatable or Fuzzy Read)**:  $T_1$  czyta daną. Następnie  $T_2$  modyfikuje albo usuwa tę daną i zostaje zatwierdzona. Gdy  $T_1$  próbuje powtórzyć odczyt, dostaje inną wartość albo okazuje się, że dana została usunięta.
- **A3 (Phantom)**:  $T_1$  odczytuje zestaw danych zaspokajających klauzulę WHERE. Następnie  $T_2$  dodaje rekordy które spełniają tę klauzulę i zostaje zaakceptowana. GDY  $T_1$  próbuje powtórzyć odczyt dostaje inny zestaw danych.
- **P4 Lost update**:  $T_1$  odczytuje daną i wylicza nową wartość.  $T_2$  odczytuje daną i wylicza nową wartość.  $T_1$  zapisuje wartość i zostaje zaakceptowana,  $T_2$  nadpisuje tę wartość i zostaje zaakceptowana.

A - oryginalna definicja, P - rozszerzona.

P0:  $w1[x]...w2[x]...((c1 \text{ lub } a1) \text{ i } (c2 \text{ lub } a2))$  w dowolnej kolejności)

A1:  $w1[x]...r2[x]...(a1 \text{ i } c2 \text{ w dowolnej kolejności})$

P1:  $w1[x]...r2[x]...((c1 \text{ lub } a1) \text{ i } (c2 \text{ lub } a2))$  w dowolnej kolejności)

A2:  $r1[x]...w2[x]...c2...r1[x]...c1$

P2:  $r1[x]...w2[x]...((c1 \text{ lub } a1) \text{ i } (c2 \text{ lub } a2))$  w dowolnej kolejności)

A3: r1[P]...w2[y in P]...c2...r1[P]...c1  
P3: r1[P]...w2[y in P]...((c1 lub a1) i (c2 lub a2) w dowolnej kolejności)

P4: r1[x]...w2[x]...c2 ...w1[x]...c1

Poziom izolacji	P0 Dirty Write	P1 Dirty Read	P2 Non-repeatable/Fuzzy Read	P3 Phantoms
<b>READ UNCOMMITTED</b>	NIE	TAK	TAK	TAK
<b>READ COMMITED</b>	NIE	NIE	TAK	TAK
<b>REPEATABLE READ</b>	NIE	NIE	NIE	TAK
<b>SERIALIZABLE</b>	NIE	NIE	NIE	NIE

## 2.1 Izolacja dla systemów z blokowaniem

**Blokada (lock)** jest **zmienną związaną z elementem danych**, która opisuje **stan** tego elementu pod względem **możliwości działań**, jakie mogą być na nim w danej chwili wykonywane.

**Dobrze sformowane zapisy** - przed zapisem **wymagane jest założenie blokady X** (ew. predykatowej).

**Dobrze sformowane odczyty** - do operacji odczytu **wymagane jest założenie blokady S** (ew. predykatowej).

**Blokada U** - gdy element danych **jest odczytywany i być może będzie potem aktualizowany**. Podnoszenie S na X może prowadzić do **zakleszczenia**. Jest zakładana **przy odczycie**, przed wykonaniem aktualizacji **jest konwertowana do X**.

przyznana v / przyznawana →	S	X	U
<b>S</b>	tak	nie	tak
<b>X</b>	nie	nie	nie
<b>U</b>	tak/nie	nie	nie

### 2.1.1 Podstawowe poziomy izolacji

Poziom izolacji	P0	P1	P2	P3	Blokady X	Blokady S
<b>Locking READ UNCOMMITTED</b>	NIE	TAK	TAK	TAK	długie	nie
<b>Locking READ COMMITED</b>	NIE	NIE	TAK	TAK	długie	krótkie
<b>Locking REPEATABLE READ</b>	NIE	NIE	NIE	TAK	długie	długie
<b>Locking SERIALIZABLE</b>	NIE	NIE	NIE	NIE	długie	długie predykatowe

### 2.1.2 Poziom izolacji Cursor Stability

**Rozszerzenie** sposobu blokowania w poziomie **Locking READ COMMITED**. Dodaje się **operację rc (czytaj kursor, pobierz wiersz)** dla instrukcji **FETCH**, blokada (**S** lub nowy typ blokady do dczytu **scroll lock**) będzie utrzymywana **do chwili przejścia** do innego wiersza lub do **zamknięcia** kursora.

**Aktualizacja wiersza przez kursor** – operacja **wc** powoduje założenie na ten wiersz **długiej blokady X**.

Dla operacji na kursorze można zdefiniować odmianę problemu **P4 Lost Update**:

**P4C: rc1[x]...w2[x]...c2...wc1[x]...c1**

Poziom izolacji Cursor Stability **eliminuje P4C**, w2[x] będzie wstrzymane do zdjęcia blokady (S, scroll lock) przez przejście do innego wiersza lub zamknięcie kursora.

Uwaga: READ COMMITTED « Cursor Stability « REPEATABLE READ

## 2.2 Poziom izolacji Snapshot i podobne

Transakcja czyta dane (zatwierdzone) z **chwili swojego początku**, Start-Timestamp.

### • Snapshot isolation (MS SQL Server SNAPSHOT)

- Są stosowane **blokady do zapisu**, ponadto przy każdym zapisie transakcja wykonuje podobne sprawdzenie jak wykonywane w propozycji 1 na końcu transakcji.
- Przechowywane są **różne wersje danych**. Transakcja **odczytuje dane aktualne** w momencie rozpoczęcia transakcji.
- **Nie ma blokad do odczytu**, operacja odczytu nie blokuje operacji zapisu ani innych operacji odczytu. Są jednak stosowane **długie blokady wyłączne do zapisu**.
- Transakcja  $T_1$  przy każdym zapisie sprawdza, czy istnieje inna transakcja  $T_2$ , która zmodyfikowała dane zapisywane i zakończyła się zatwierdzeniem. Jeśli istnieje, to  $T_1$  jest wycofywana.
- Stosowaną tu zasadę można określić jako **First-writer-wins**.

### • Read Committed Snapshot (Oracle READ COMMITTED)

- Operacja odczytu **czyta ostatnią zatwierdzoną wartość** elementu danych (niekoniecznie sprzed początku transakcji).
- W przyjętej implementacji wiersze kursora czytane są w momencie otwarcia kursora, a nie w momencie odczytu wiersza.

Poziom izolacji Snapshot **nie gwarantuje szeregowalności konfliktowej** harmonogramów.

**A5 Data Item Constraint Violation** Załóżmy, że na elementy danych x oraz y narzucono pewne ograniczenie C(). Każda transakcja z osobna dba o spełnienie C().

### A5A Skrzywiony odczyt (Read Skew)

$T_1$  odczytuje x, potem inna transakcja  $T_2$  aktualizuje x oraz y do nowych wartości i zostaje zatwierdzona. Jeśli następnie  $T_1$  odczyta y, to będzie miała niespójny obraz danych.

**A5A: r1[x]...w2[x]...w2[y]...c2...r1[y]...(c1 or a1)**

### A5B Skrzywiony zapis (Write Skew)

$T_1$  odczytuje x (ew. odczytuje też y). Następnie inna transakcja  $T_2$  odczytuje y (ew. odczytuje też x). Następnie  $T_1$  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.

**A5B: r1[x]...r2[y]...(w1[y] w2[x] c1 i c2 w dowolnej sensownej kolejności)**

**A5A oraz A5B nie wystąpią w harmonogramach, w których wykluczony jest P2.**

	P0	P1	A3	P3	A5A	A5B
Snapshot Isolation	nie	nie	nie	tak	nie	tak
Read Committed					tak	
Locking Repeatable Read				tak		nie

REPEATABLE READ »« Snapshot Isolation

### 3 Kursory.

- zmienne umożliwiające **sekwencyjny dostęp** do tabel
- pobranie wiersza: FETCH, zamknięcie kursora: CLOSE, zwolnienie pamięci: DEALLOCATE

#### 3.1 ISO

:

- **INSENSITIVE** – tworzona jest kopia statyczna danych w bazie tempdb. Nie można aktualizować.
- **SCROLL** – można wykorzystywać wszystkie opcje FETCH: FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE (domyślnie można używać tylko NEXT).
- **READ ONLY** – nie można wykonywać **pozycjonowanych zmian danych** (np. UPDATE ... WHERE CURRENT OF nazwakursora).
- **UPDATE OF** – można aktualizować **tylko wymienione kolumny** (samo UPDATE oznacza zgodę na aktualizacje wszystkich kolumn).

#### 3.2 T-SQL

:

- **LOCAL** – zakres (scope) kursora jest lokalny do wsadu (batch), procedury składowanej, wyzwalacza, gdzie kursor został zadeklarowany.
- **GLOBAL** – kursor będzie globalny dla połączenia. Zwolnienie pamięci będzie zrealizowane przy zamknięciu połączenia lub wskutek jawnie wykonanej operacji deallocate.
- **FORWARD\_ONLY** – określa, że kursor może być skrolowany tylko **od pierwszego do ostatniego wiersza** (można używać tylko FETCH NEXT).
  - Jeśli nie wyspecyfikowano ani FORWARD\_ONLY ani SCROLL, ani STATIC/KEYSET/DYNAMIC, FORWARD\_ONLY jest przyjmowany
  - Dla kursorów STATIC, KEYSET i DYNAMIC domyślną opcją jest SCROLL.
- **STATIC** – odpowiednik **INSENSITIVE**.
- **KEYSET** – w bazie tempdb tworzona jest tabela z wartościami klucza (kluczy) określającymi zestaw wierszy.
  - Jeśli w chociaż jednej tabeli, do której odwołuje się kursor, nie ma indeksu unikalnego, to typ kursora zmieniany jest na STATIC.
  - Zmiany na danych są widoczne poprzez kursor (w kolejnych operacjach FETCH), z wyjątkiem dodania nowego wiersza, skasowania wiersza lub aktualizacji klucza. Przy próbie pobrania nieistniejącego wiersza funkcja @@FETCH\_STATUS zwraca wartość -2.
- **READ\_ONLY** – zmiany pozycjonowane nie są możliwe (domyślnie są).
- **OPTIMISTIC** – zmiany pozycjonowane są możliwe, jest realizowane optymistyczne sterowanie współbieżnością.
- **SCROLL LOCK** – zmiany pozycjonowane są możliwe, jest realizowane pesymistyczne sterowanie współbieżnością z blokadami.
  - Jeśli nie wyspecyfikowano ani READ\_ONLY, ani OPTIMISTIC ani SCROLL LOCK, to będzie przyjęte ustawienie według poniższych reguł:
    - \* Jeśli zdanie SELECT nie wspiera aktualizacji (np. niewystarczające uprawnienia), kursor będzie READ\_ONLY.
    - \* Standardowo kursory STATIC i FAST\_FORWARD są READ\_ONLY.



\* Standardem dla kursorów DYNAMIC i KEYSET jest OPTIMISTIC.

### 3.3 Sterowanie współbieżnością

- **OPTIMISTIC WITH VALUES** - na początku wczytujemy wartości w wierszach, przed zapisaniem sprawdzamy czy się nie zmieniły
- **OPTIMISTIC WITH ROW VERSIONS** - mamy dodatkową kolumnę row version, którą sprawdzamy by się upewnić że nie było zmian
- **SCROLL\_LOCKS (pesimistic)** - zakładamy blokadę = gwarancja sukcesu naszej operacji

### 3.4 Przykład

Np. kiedy chcemy utworzyć logi kto co usunął - trigger dla usuwania i zapisanie poszczególnych wierszy wraz z user id do tabeli logów.

## 4 Obsługa transakcji w procedurach składowanych.

```
CREATE PROC [dbo].[usp_my_proc] (@arg1 INT, @arg2 VARCHAR(25))
AS
DECLARE @tran_count AS INT
SET @tran_count = @@TRANCOUNT

IF @tran_count > 0
SAVE TRAN save_point
ELSE
BEGIN TRAN

BEGIN TRY

-- SQL code goes here

IF @tran_count > 0
COMMIT TRAN save_point
ELSE
COMMIT
END TRY
BEGIN CATCH
IF @tran_count > 0
ROLLBACK TRAN save_point
ELSE
ROLLBACK
END CATCH;
GO
```

## 5 Odtwarzanie baz danych po awarii.

**LSN** - log sequence number.

**Checkpoint** - moment **zrzucenia buforowanych** w pamięci logów transakcji i stron (**dirty pages**) na dysk oraz ustawienia LSN checkpointu w boot page.

**Transaction logs** są **obcinane** w momencie checkpointu (przeważnie **automatycznie co pewien określony czas**). W zależności od ustawionego **recovery model**:

- **Simple Recovery Model** - transaction logs są **ucinane do momentu rozpoczęcia pierwszej niezatwierdzonej transakcji**. Serwer w razie awarii może:
  - **automatycznie przywrócić niezsynchronizowane** jeszcze z plikiem strony **dane**,
  - **wycofać** te, które **nie** zostały **zatwierdzone** a są już **zapisane** do pliku strony.
- **Full or Bulk-Logged Recovery Models** - transaction logs są **ucinane do momentu ostatniej kopii zapasowej** tych logów. Możliwe jest:
  - odtworzenie stanu danych aż do momentu samej awarii (w Simple Recovery Model jedynie do momentu ostatniej pełnej lub różnicowej kopii, ponieważ potem nie mamy logów).

## 5.1 Przywracanie bazy danych po awarii dysków (plików z danymi)

Zakładamy tu, że pliki z logami przetrwały!

1. Tworzymy **kopię zapasową logów** transakcji, które nie zostały zbackupowane wcześniej

```
BACKUP LOG <database_name> TO <backup_device>
WITH NORECOVERY, NO_TRUNCATE;
```

2. Odtwarzamy **pełną kopię zapasową**

```
RESTORE DATABASE <database_name> FROM <backup_device>
WITH NORECOVERY;
```

3. (Ewentualnie) Odtwarzamy **kopię różnicową** wykonaną po kopii pełnej z poprzedniego punktu

```
RESTORE DATABASE <database_name> FROM <backup_device>
WITH NORECOVERY;
```

4. **Aplikujemy każdy zbackupowany transaction log** (włącznie z tym zbackupowanym na samym początku) zawierający logi od ostatniej odtworzonej kopii zapasowej

```
RESTORE LOG <database_name> FROM <backup_device>
WITH NORECOVERY;
```

5. Na końcu wydajemy **komendę** mówiącą bazie danych, że **odtworzenie zostało już ukończone** i baza danych może być normalnie używana

```
RESTORE DATABASE <database_name>
WITH RECOVERY;
```

Podczas procesu odtwarzania z opcją **NORECOVERY** baza danych **nie** jest normalnie **dostępna** (nie możemy z niej korzystać). Ponadto:

- Operacje “**roll back**” **nie** są jeszcze **wykonywane**.
- Operacje “**roll forward**” się **wykonują**.
- Umożliwia nam to zaaplikowanie kolejnego logu transakcji czy backupu.
- Dopiero podczas aplikowania **ostatniego logu** transakcji możemy dać opcję **WITH RECOVERY** co spowoduje **wykonanie** wszystkich operacji “**roll back**” i uznanie tej bazy danych za **w pełni odtworzoną**, a przez to **dostępną** już normalnie dla wszystkich użytkowników.

Ustawienie opcji **WITH STANDBY** pozwala (administratorom) **wykonywać odczyty** na tej bazie danych. Wymaga podajania ścieżki do **pliku standby** (zostanie utworzony).

## 6 Mirroring i database snapshot.

### 6.1 Mirroring

- Zazwyczaj jest zestawiany **między odległymi** od siebie **maszynami**
- Mirroring odbywa się **na poziomie baz danych** a **nie całego serwera**
- Aby baza danych mogła być mirrorowana **na obu serwerach musi być** ustawiony dla tej bazy danych **full recovery model**
  - **mirroring opiera się na przesyłaniu logów** transakcji pomiędzy serwerami
- Jest tylko **jeden principale** i **jeden mirror** oraz opcjonalnie **jeden witness** (przynajmniej w MSSQL serverze):
  - **Principale** - jest to **serwer produkcyjny**. Udostępnia on swoją bazę do odczytu i zapisu.
  - **Mirror** - jest to serwer, który utrzymuje **kopię danych** z principala aby móc w razie potrzeby przejąć jego rolę. Dane na tym serwerze nie są dostępne ani do odczytu, ani do zapisu.
  - **Witness** - jest to serwer, który **nie przechowuje żadnych danych**, a jedynie **utrzymuje kontakt** z principalem i mirrorem.
  - **Automatic failover** Jeśli:
    - \* mirror nie ma kontaktu z principalem,
    - \* mirror ma kontakt z witnessem,
    - \* witness również stracił kontakt z principalem

Wtedy uznaje się **principala za niedostępnego** i **mirror staje się principalem**. Aby automatic failover działał, **mirroring** musi być skonfigurowany w trybie **high-safety**
- Mirroring może być skonfigurowany w dwóch trybach:
  - **High-safety mode (synchroniczny)** - w tym trybie aby **transakcja** przeprowadzana **na principale** mogła zostać uznana za **zatwierdzoną**, principal musi poczekać na **potwierdzenie od mirrora** odnośnie wpisania danych związanych z tą transakcją do jego dziennika transakcji.
  - **High-performance (asynchroniczny)** - w tym trybie aby **transakcja na principalu** została **zatwierdzona** **nie musi on czekać** na potwierdzenie od mirrora. **Dane do mirrora** odnośnie wykonanych operacji są **wysyłane niezależnie** (jak najszybciej).
- **Połączenie między serwerami** może być **zabezpieczane** przy pomocy **Windows authentication** albo **Certificate-based authentication**.

### 6.2 Database snapshot

- Snapshot jest pełnym obrazem bazy danych z momentu jego utworzenia
- Dostępny tylko do odczytu
- Używa mechanizmu “copy-on-write”, co w tym przypadku oznacza, że tylko strony danych, które zostały zmienione od czasu utworzenia snapshotu są kopiowane do specjalnych plików (sparse files). Z tego wynika że w najgorszym przypadku utrzymywanie snapshotu może zająć dodatkowo tyle miejsca na dysku ile zajmowała oryginalna baza danych w momencie jego tworzenia.
- Zastosowania:
  - Posiadanie stanu danych z pewnego okresu, np. w celu utworzenia raportu
  - Swego rodzaju zabezpieczenie przed utratą danych na wypadek błędu po stronie administratora lub użytkownika - ze snapshotu można zawsze przekopiować z powrotem dane do właściwej

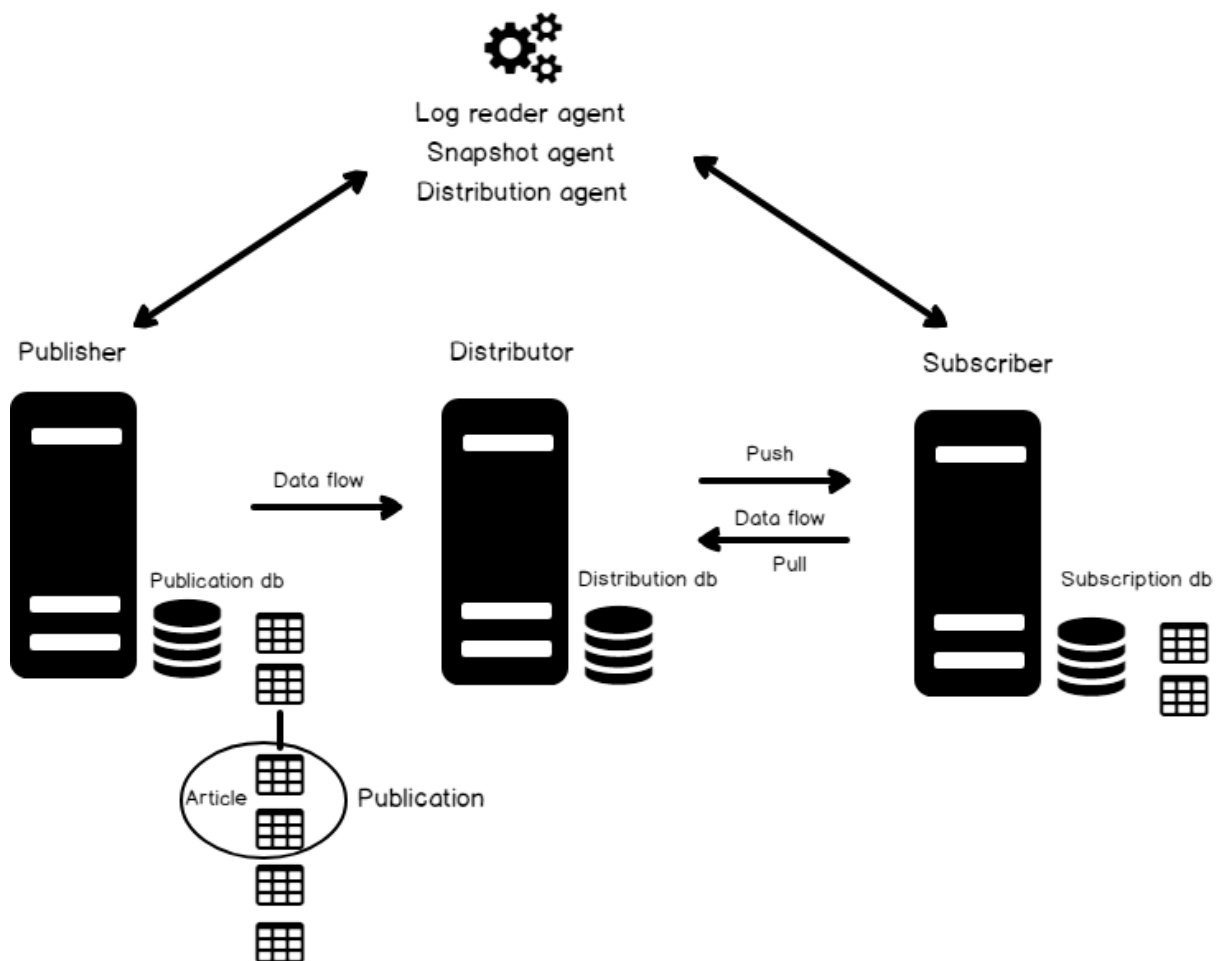
bazy danych lub przywrócić ją wprost ze snapshota przy użyciu komendy RESTORE DATABASE. Snapshoty **nie zastępują jednak kopii zapasowych**, ponieważ nie można na przykład odtworzyć z nich uszkodzonej bazy danych lub bazy, która jest offline

- Trzymanie szeregu snapshotów z różnych okresów czasu może być używane do analizy zmian jakie zaszły w tych danych

- Tworzenie snapshotu:

```
CREATE DATABASE <database_snapshot_name>  
ON (NAME = <logical_file_name>, FILENAME = <file_path>)  
AS SNAPSHOT OF <source_database_name>
```

## 7 Replikacja.



### 7.1 Definicja

Replikacja to proces powielania informacji pomiędzy różnymi serwerami baz danych. Ze wszystkich kopii można korzystać, w przeciwieństwie do mirroringu.

- **Article** - **obiekt** bazy danych (tabela, procedura składowana, itp.), który jest **opublikowany** (będzie dostępny dla Subscriberów)
- **Publication** - **logiczna kolekcja** opublikowanych **obiektów** (Articlów) umożliwiająca również **konfigurację** ich **atrybutów** (np. co ma się stać gdy dana tabela lub procedura składowana już istnieje w docelowej bazie danych)

- **Publisher** - baza danych zawierająca publikowane zasoby
- **Distributor** - pośrednik w wymianie danych między Publisherem a Subskrybentem. **Przechowuje metadane** odnośnie publikacji, w niektórych konfiguracjach jest także **buforem** dla przekazywanych danych. Na dystrybutorze mamy co najmniej jedną **Distribution database**, która przechowuje wspomniane metadane. Każdy Publisher ma swoją Distribution database na dystrybutorze.
- **Subscriber** - instancja bazy danych, która **pobiera dane** od Publishera. Subscriber może pobierać dane **z wielu publikacji**. W zależności od konfiguracji może również **przekazywać** te dane do innych baz (działać jako Publisher), lub **odsyłać modyfikacje** przeprowadzone po jego stronie na synchronizowanych danych z powrotem do Publishera
- **Subscription database** - docelowa baza danych dla replikowanych obiektów

## 7.2 Podstawowe metody replikacji w systemie Microsoft SQL Server

Najważniejsze typy publikacji:

- **Snapshot publication** - jest to **jednorazowy transfer** z Publishera do Subskrybenta. **Po dokonaniu tego transferu** danych żadne **zmiany** wykonane po którejkolwiek ze stron **nie są** dalej śledzone.
- **Transactional publication** - umożliwia **prawie natychmiastowe aktualizowanie** danych zmienionych na Publisherze do Subskrybenta. **Przeważnie** i domyślnie jest to **replikacja tylko w jedną stronę**, aczkolwiek istnieje możliwość obustronnego jej skonfigurowania.  
Dwa główne sposoby aktualizowania zmienionych danych:
  - **Push - Distributor** bezpośrednio **aktualizuje** dane na subskrybencie. Wtedy Distribution Agent działa na Distributorze i “wypycha” dane do Subskrybenta
  - **Pull - Subskrybent** sam co jakiś czas **pobiera** zmodyfikowane dane z Distributora. W tym przypadku Distribution Agent działa na Subskrybencie i “zaciąga” dane z Distributora (na dystrybutorze musi być **udostępniony udział Windows** z danymi do pobrania - przez to ze względu na uwierzytelnienie, dostępne jest to tylko w domenie Windows?)
- **Peer-to-peer replication** - w tym typie replikacji **zmiany mogą być wprowadzane na każdej z baz** danych.
  - Baza na której została wprowadzona zmiana wysyła ją jak najszybciej do pozostałych tak aby wszystkie z nich miały spójną kopię.
  - **Mogą wystąpić niespójności danych** w przypadku, gdy ta sama porcja danych zostanie w tym samym czasie zmodyfikowana na wielu węzłach.
  - Niespójności muszą być **rozwiązywane ręcznie**. (?)
- **Merge replication** - ten tryb zakłada **synchronizację danych** z pozostałymi węzłami **co określony czas**.
  - W momencie synchronizacji danych **zmiany są mergowane**, a ewentualne konflikty (automatycznie?) rozwiązywane.
  - Węzły **nie muszą być cały czas podłączone** do sieci.
  - **Łączność sieciowa** z Distributorem jest **wymagana tylko w momencie synchronizacji** danych.

## 8 Widoki rozproszone.

Ideą tych widoków jest **rozłożenie obciążenia** na dwa lub więcej serwerów poprzez podzielenie poziome całego zestawu danych (wszystkich wierszy) według zadanych kryteriów:

1. Na obu serwerach tworzymy odpowiednie tabele do przetrzymywania naszych danych. Założmy że będziemy chcieli partycjonować nasze dane ze względu na nazwisko, tak aby nazwiska od A-M włącznie były na jednym serwerze a reszta na drugim:

- Na tworzonych tabelach musimy założyć **CONSTRAINTy** na kolumny, po których będziemy partycjonować! To dzięki niemu serwer podczas wykonywania kwerendy będzie wiedział do którego serwera skierować zapytanie, oraz będziemy mieli pewność, że nie wstawimy rekordu na nieodpowiedni serwer.
- Na pierwszym serwerze mogłoby wyglądać to tak:

```
CREATE TABLE UsersAM(  
  first_name varchar(40) NOT NULL,  
  surname varchar(20) NULL,  
  CONSTRAINT CHK_UsersAM CHECK (surname < 'N')  
)
```

- Natomiast na drugim serwerze mogłoby wyglądać to tak:

```
CREATE TABLE UsersNZ(  
  first_name varchar(40) NOT NULL,  
  surname varchar(20) NULL,  
  CONSTRAINT CHK_UsersNZ CHECK (surname >= 'N')  
)
```

2. TNa każdym z serwerów musimy skonfigurować **linked servers** czyli serwery do których będziemy mogli **zdalnie zadawać zapytania**. W naszym przypadku na serwerze pierwszym naszym linked serwerem będzie serwer drugi, a na drugim pierwszy.

- Konfiguruując Linked server podajemy jego **alias**, **sterownik OLE DB**, którego będziemy używać do komunikacji z linked serwerem, oraz właściwą **ścieżkę do serwera** (w przypadku MSSQL Servera jest to NazwaKomputera\NazwaInstancji, lub gdy jest to instancja domyślna wystarczy sama nazwa komputera)
- Dodatkowo musimy **skonfigurować uwierzytelnienie** między tymi serwerami. W domenie Windows sprowadza się to do **nadania odpowiednich uprawnień** użytkownikowi (wykonującemu zapytanie?). W przeciwnym przypadku musimy skorzystać z **uwierzytelnienia SQL** tworząc odpowiedniego użytkownika na jednym i drugim serwerze oraz konfiguruując Linked server login na obu serwerach.
- Gdy będziemy chcieli na serwerze o aliasie “Server2” z bazy “Database” wybrać wszystkie rekordy z tabeli “UsersNZ” będziemy musieli użyć następującego zapytania:

```
SELECT *  
FROM Server2.Database.dbo.UsersNZ
```

3. Teraz na każdym z serwerów tworzymy **widok rozproszony**, który będzie **wybierał dane z obu serwerów**. W naszym przypadku taki widok na serwerze pierwszym może wyglądać następująco:

```
CREATE VIEW AllUsers  
AS  
SELECT *  
FROM UsersAM  
  
UNION ALL  
  
SELECT *  
FROM Server2.Database.dbo.UsersNZ  
  
GO
```

4. Teraz gdy wydany **zapytanie** do widoku **erwer** na podstawie **constraintów** zdefiniowanych dla tabel **zdecyduje** gdzie skierować zapytanie.

## 9 Podstawowa charakterystyka transakcji rozproszonych.

**Transakcja rozproszona** to transakcja, której **polecenia DML** (insert, update, delete) i select **odwołują się do tabel** znajdujących się co najmniej w dwóch **węzłach rozproszonej bazy danych**. Transakcja rozproszona jest często nazywana **transakcją globalną**.

- Transakcja rozproszona jest reprezentowana przez **zbiór transakcji lokalnych**.
- **W każdej z baz danych**, do której odwołuje się transakcja rozproszona tworzona jest jedna **transakcja lokalna**.
- **Cechy transakcji rozproszonej**
  - **Trwałość**
  - **Spójność**
  - **Izolacja**
  - **Atomowość** - wszystkie transakcje lokalne **zatwierdzone** lub wszystkie **wycofane** (2PC)

### 9.1 Aktorzy

- Każdy węzeł do którego odwołuje się transakcja rozproszona pełni **ściśle określoną funkcję**.
- Rodzaje węzłów:
  - **Koordynator globalny** (ang. global coordinator) jest tym węzłem z którego **zainicjowano transakcję** rozproszoną. Zadaniem koordynatora globalnego jest **zarządzanie** całą **transakcją**, tj. doprowadzenie jej w całości do zatwierdzenia lub wycofania.
  - **Koordynator lokalny** (ang. local coordinator) jest węzłem który **otrzymuje żądanie, przetwarza je i wysyła** kolejne żądania do innych podległych mu węzłów. KL nie **koordynuje** całej transakcji globalnej a jedynie **te żądania, które sam wysłał**.
  - **Uczestnik** (ang. node) jest węzłem, **do którego jest kierowane żądanie** transakcji rozproszonej, a sam węzeł **nie wysyła** żadnych **żądań**.
  - **Węzeł zatwierdzania** (WZ):
    - \* **inicjowanie zatwierdzania lub wycofywania** transakcji **zgodnie z komunikatem od KG** (jako pierwszy je wykonuje)
    - \* zawiera **status zatwierdzania** transakcji rozproszonej odczytywany przez transakcje lokalne
    - \* wybierany przez administratora systemu
      - parametr konfiguracyjny **COMMIT\_POINT\_STRENGTH**: 0-255
      - odzwierciedla **ilość danych krytycznych** w węźle i jego **niezawodność**
    - \* węzeł o najwyższej wartości COMMIT\_POINT\_STRENGTH jest węzłem zatwierdzania
    - \* **transakcja rozproszona jest uznawana za zatwierdzoną** jeżeli zostanie **zatwierdzona w węźle zatwierdzania**, nawet jeśli pozostałe węzły jeszcze nie zatwierdziły swoich transakcji lokalnych

## 9.2 Protokół zatwierdzania dwu-fazowego (2PC)

- Gwarantuje atomowość zatwierdzania lub wycofywania transakcji rozproszonej.
- Fazy realizacji:
  - **przygotowanie** (prepare) transakcji lokalnych do zatwierdzania lub wycofywania.
  - **zatwierdzanie** (commit) lub wycofywanie transakcji lokalnych.
  - **zakończenie** (forget) - usuwanie z systemu informacji o transakcji rozproszonej i zwalnianie niezwolnionych w fazie zatwierdzania zasobów.

### 9.2.1 Faza przygotowania

1. **KG wybiera węzeł zatwierdzania** spośród wszystkich węzłów biorących udział w transakcji rozproszonej.
2. KG wysyła do wszystkich węzłów z wyjątkiem węzła zatwierdzania **komunikat PREPARE**, wymuszający przygotowanie transakcji lokalnych do zatwierdzenia.
3. Po zakończeniu operacji przygotowania do zatwierdzenia transakcji, **każdy węzeł przesyła** do KG **komunikat PREPARED** jeżeli się przygotował do zatwierdzenia swojej transakcji lokalnej.
4. W ostatnim kroku tej fazy **KG odbiera komunikaty** od wszystkich węzłów, z wyjątkiem węzła zatwierdzania. Po ich odebraniu następuje **przejsie do fazy zatwierdzania**.

Faza przygotowania w **węźle uczestnika**:

1. **Odbiór komunikatu PREPARE** od KG.
2. Dokonanie **zapisów** w bieżących plikach **dziennika powtórzeń** (ang. online redo logs). W plikach tych znajdują się wszystkie zmiany wykonane w bazie danych wprowadzone zarówno przez zatwierdzone, jak i niezatwierdzone transakcje. Informacje z plików dziennika powtórzeń są wykorzystywane w czasie odtwarzania bazy danych po awarii.
3. Wysłanie komunikatu **PREPARE** do węzłów podległych.
4. **Odbiór** komunikatów od węzłów podległych.
5. **Wysłanie** komunikatu do KG.
  - Jeżeli w węźle **nie dokonano modyfikacji** danych wysyła on komunikat **READ-ONLY**.
  - Jeśli uczestnik i wszystkie węzły mu podległe są przygotowane do zatwierdzania, to wysyła on do KG komunikat **PREPARED**.
  - Jeśli choć jeden węzeł nie mógł się przygotować, to uczestnik **wycofuje swoją transakcję lokalną** i wysyła do koordynatora globalnego komunikat **ABORT**.

### 9.2.2 Faza zatwierdzania

1. **KG odbiera potwierdzenia** od uczestników.
2. Jeśli wszyscy odpowiedzieli komunikatem **PREPARED**, wówczas KG wysyła **do węzła zatwierdzania komunikat COMMIT**, czyli żądanie zatwierdzenia transakcji rozproszonej.
3. **WZ zatwierdza transakcję** i wysyła do KG komunikat potwierdzający zatwierdzenie (**COMMITTED**).
4. Po otrzymaniu od WZ komunikatu **COMMITTED**, **KG wysyła komunikat COMMIT** do pozostałych węzłów (uczestników).

**Wycofanie** w fazie zatwierdzania:

1. KG odbiera potwierdzenia od uczestników.
2. Przynajmniej jeden węzeł odpowiedział komunikatem **ABORT**, więc **KG wysyła do WZ komunikat ABORT**, czyli żądanie wycofania transakcji rozproszonej.



3. **WZ wycofuje transakcję** i wysyła do KG komunikat potwierdzający wycofanie (**ABORTED**).
4. Po otrzymaniu od WZ komunikatu ABORTED, **KG wysyła** komunikat **ABORT** do pozostałych węzłów (uczestników).

Faza zatwierdzania w węźle **uczestnika**:

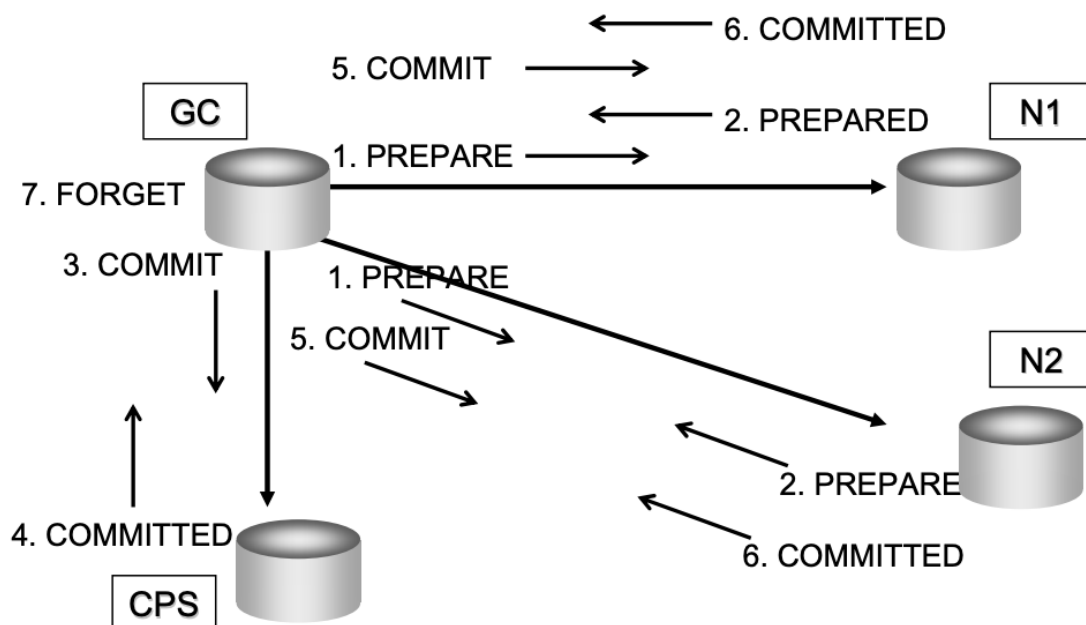
1. Odbiór COMMIT od KG.
2. Zatwierdzenie lokalnej transakcji.
3. Zwolnienie blokad.
4. Zapis informacji o zatwierdzeniu w pliku dziennika powtórzeń.

### 9.2.3 Faza zakończenia

1. **Usunięcie z systemu informacji o zakończonej transakcji** rozproszonej.
2. **Zwolnienie** wszystkich niezwolnionych jeszcze **zasobów** systemowych.



## 2PC - podsumowanie



### 9.3 Graf wywołań transakcji

- Transakcje lokalne tworzą tzw. graf wywołań transakcji, zwany również **drzewem sesji**.
- Wierzchołki to **węzły** (bazy).
- Łuki (krawędzie) stanowią **zadania** wykonania poleceń z jednej bazy w innej.

#### Problemy sprzętowo-programowe

- W czasie fazy COMMIT (ROLLBACK) następuje awaria sieci, węzła lub zdalnej bazy danych
  - nie wszystkie węzły zatwierdziły (wycofały)

- nie wszystkie węzły potwierdziły zakończenie operacji
- transakcja rozproszona w stanie zawieszenia “in-doubt” do momentu usunięcia awarii, nawiązania połączenia ze zdalnym węzłem i doprowadzenia transakcji do końca.
- transakcja w stanie “in-doubt” **blokuje dane**.
- Automatyczne odtwarzanie transakcji rozproszonej w stanie “in-doubt” po usunięciu awarii
  - wynik: wszystkie węzły zatwierdzą lub wszystkie wycofają

## 10 MongoDB.

- **otwarty, nierelacyjny** system zarządzania bazą danych
- duża **skalowalność i wydajność**
- **brak ściśle zdefiniowanej struktury** obsługiwanych baz danych
- dane składowane są jako **dokumenty JSON**, co umożliwia aplikacjom bardziej naturalne ich przetwarzanie zachowując możliwości tworzenia **hierarchii** oraz **indeksowania**.
- dokumenty umieszczone są w **kolekcjach** przechowywanych w bazach danych
- **model działania:**
  - model asynchronicznych zapisów i eventual consistency,
  - klient otrzymuje jedynie obietnicę, że dane zostaną zapisane w przyszłości
- **możliwości:**
  - jednorodne wsparcie dla standardu Unicode,
  - obsługa danych w innych kodowaniach w formacie binarnym,
  - duża liczba obsługiwanych typów danych,
  - obsługa kursorów,
  - zapytania ad-hoc,
  - zapytania do zagnieżdżonych pól dokumentów,
  - indeksowanie,
  - wsparcie dla agregacji danych,
  - możliwość składowania plików w bazie,
  - architektura zaprojektowana z myślą o łatwej replikacji,
  - łatwa skalowalność,
  - brak joinów,
  - brak ograniczenia schematami danych
- wewnętrznym językiem do definiowania zapytań oraz funkcji agregujących jest JavaScript wykonywany bezpośrednio przez serwer MongoDB.
- **wady:**
  - w domyślnych ustawieniach bazy MongoDB dostępne są **publicznie**, bez hasła.
  - problemy ze spójnością danych wynikające z błędów implementacyjnych modelu (naprawione),
  - problemy z kompletnością zwracanych dokumentów przy wyszukiwaniach z użyciem indeksów (pomijanie aktualnie aktualizowanych)

– ograniczone wsparcie kodowania UTF-8

## 10.1 Podstawowe operacje

### 10.1.1 Insert

```
>db.COLLECTION_NAME.insert(document)
```

Jeśli nie wyspecyfikujemy `_id` w dokumencie, MongoDB wygeneruje dla niego unikalny identyfikator.

Analogicznie działa metoda `save()`, jednak dla wyspecyfikowanego `_id` może podmienić dane na nowe (insert wyrzuci błąd).

### 10.1.2 Find

```
>db.COLLECTION_NAME.find(query)
```

Wykonuje kwerendę na kolekcji, zwraca dokumenty w sposób nieustrukturyzowany (można je sformatować używając `pretty()`).

Wywołany **bez argumentu** zwraca **wszystkie** dokumenty w kolekcji. Argumentem jest **kwerenda**:

Op	Syntax	Example	RDBMS Equivalent
==	{<key>:<value>}	db.mycol.find({"by":"tutorials point"})	where by = 'tutorials point'
<	{<key>:{\$lt:<value>}}	db.mycol.find({"likes":{\$lt:50}})	where likes < 50
≤	{<key>:{\$lte:<value>}}	db.mycol.find({"likes":{\$lte:50}})	where likes ≤ 50
>	{<key>:{\$gt:<value>}}	db.mycol.find({"likes":{\$gt:50}})	where likes > 50
≥	{<key>:{\$gte:<value>}}	db.mycol.find({"likes":{\$gte:50}})	where likes ≥ 50
≠	{<key>:{\$ne:<value>}}	db.mycol.find({"likes":{\$ne:50}})	where likes != 50

Można używać również **AND** jako `db.mycol.find({$and: [ {key1: value1}, {key2: value2} ]})` lub po prostu `db.mycol.find({key1: value1}, {key2: value2})`.

**OR** analogicznie: `db.mycol.find({$or: [ {key1: value1}, {key2: value2} ]})`.

## 11 XML w systemie Microsoft SQL Server.

- W SQL Serverze 2000 dodano klauzulę **FOR XML**, która pozwala zwrócić wynik kwerendy jako XML. **AUTO** zapisuje informacje jako atrybuty, dodanie **ELEMENTS** - jako podelementy.
- W SQL Serverze 2005 dodano **typ danych XML**. Pozwala on natywnie **przechowywać** dokumenty i fragmenty XML w **kolumnach i zmiennych**.
- Typ danych XML zapewnia **odpowiednie formatowanie** danych, tj. **zgodne ze standardami ISO**.
- Obiekty typu XML mają narzucone pewne **ograniczenia**:
  - Instancja XML nie może przekroczyć **2GB**.
  - Kolumna typu XML **nie** może być **kluczem głównym** tabeli.
  - **Nie** można używać obiektów XML w klauzuli **GROUP BY**.
  - **Nie** można **porównywać** ani **sortować** danych typu XML.
- Użycie typu XML nie zawsze jest **potrzebne** - jeśli zamierzamy je tylko przechować, bez przeszukiwania czy zmieniania, można użyć np. `VARCHAR(MAX)`. Unikniemy w ten sposób wywoływania parsera, który jest obciążający obliczeniowo.

- **Schemat kolekcji XML** (XML schema collection) składa się z jednego lub więcej **schematów definicji XML** (XML Schema Definition, XSD), używanych do walidacji danych znajdujących się w obiekcie XML. Zawierają faktyczne formatowanie informacji, definiują ich strukturę i typy danych; na jego podstawie jest parsowany XML.
- SQL Server rozróżnia **dokumenty i fragmenty XML**. Jeśli obiekt XML jest typowany, to można określić czy przyjmuje oba, czy tylko pełne dokumenty.
- Metoda **query()** zwraca **instancję** nietypowanego XMLa będącą **podzbiorem** większego obiektu.
- Metoda **value()** zwraca **element** lub **wartość argumentu** (typ danych, nie fragment XML).
- Metoda **exist()** **testuje istnienie** elementu podanego argumentem. Zwraca:
  - **1 (BIT)** jeśli element szukany **istnieje**,
  - **0 (BIT)** jeśli element szukany **nie istnieje**,
  - **NULL** jeśli XML podany do przeszukania jest **nullem**.
- Metoda **nodes()** zwraca **tabełę** (wiersze) z pojedynczą kolumną, gdzie każdy wiersz zawiera pojedynczą instancję XMLa z wyniku zapytania. Używany w celu **podzielenia instancji XMLa na relacyjne dane**.
- Metoda **modify()** jest jedyną metodą pozwalającą **modyfikować** dane typu XML. Bierze jako argument wyrażenie w XML Data Modification Language (XML DML). Pozwala **wstawiać** (insert), **edytować** (replace) i **usuwać** (delete) dane w obiekcie XML.

## 12 Przetwarzanie grafów.

- **Wierzchołki** reprezentują **obiekty** (rekordy, wiersze, dokumenty).
- Kolekcja wierzchołków reprezentuje powiązane/podobne obiekty.
- Trzy ważne **cechy** węzłów:
  - **Identyfikator** – łatwo się domyślić po co
  - **Etykietę** - mówi nam czym jest dany węzeł
  - **Zestaw właściwości** w postaci **klucz-wartość**.
- **Krawędzie** (skierowane lub nie) reprezentują **relacje** pomiędzy wierzchołkami.
- kolekcja krawędzi reprezentuje powiązane/podobne relacje.
- Dwie najważniejsze cechy:
  - **Typ** relacji. Podobnie jak **etykieta** węzła, możemy nazwać go dowolnie.
  - **Zestaw właściwości**, zupełnie jak w węzłach

Kiedy używać grafowych baz danych?

1. Dane hierarchiczne
2. Skomplikowane relacje many-to-many
3. Skomplikowane relacje między obiektami

Tworzenie grafowych baz danych

1. CREATE DATABASE
2. CREATE TABLE ... AS NODE
  - automatyczne dodanie dwóch kolumn do tabeli oraz stworzenie unique, non-clustered index na jednej z tych kolumn

- graph\_id
  - node\_id
3. sys.tables - posiada kolumny is\_\_node oraz is\_\_edge ustawiane na wartości 0 lub 1
  4. sys.columns - posiada kolumny graph\_type oraz graph\_type\_\_desc