1. Modele i języki relacyjnych baz danych

Definicje

Atrybuty - symbol, nazwa, na przykład: Nazwisko, NrEwid, DataUr, Cena.
Dziedzina atrybutu - zbiór dopuszczalnych wartości atrybutu A.
Krotka typu U (U - zbiór atrybutów; wiersz, rekord, ang. *tuple*).
Tabela - skończony zbiór krotek samego samego typu.

Algebra relacji:

- Suma mnogościowa (*union*):
  - R, S - tabele tego samego typu U;
  - Wynik - tabela typu U;
  - $R \cup S = \{t \mid t \in R \vee t \in S\}$;
  - W SQL - UNION (usuwa duplikaty), UNION ALL - pozostawia duplikaty.
- Różnica mnogościowa (*difference*):
  - R, S - tabele tego samego typu U;
  - Wynik - tabela typu U;
  - $R - S = \{t \mid t \in R \wedge t \notin S\}$;
  - W SQL - except (usuwa duplikaty).
- Przekrój (*intersection*):
  - R, S - tabele tego samego typu U;
  - Wynik - tabela typu U;
  - $R \cap S = \{t \mid t \in R \wedge t \in S\}$;
  - W SQL - intersect (usuwa duplikaty).
- Projekcja, rzutowanie - z relacji wybieramy tylko podane kolumny, symbol π. Przykład $\pi_{\text{kolumna1,...,kolumna n}}$.
- Selekcja, wybór - zawiera tylko te krotki, dla których jest spełniony podany warunek, symbol σ. Przykład $\sigma_{\text{warunek}}(R)$, gdzie warunek to jakiś warunek, który musi zostać spełniony np. gatunek='Papuga', a R to tabela.
- Przemianowanie - służy do: zmiany nazwy relacji, zmiany nazw atrybutów relacji, tego i tego.
- Złączenie (naturalne) - w złączeniu naturalnym dwóch tabel R i S typów odpowiednio X i Y:
  - atrybuty występujące w X mogą występować także w Y,
  - relacja wynikowa zawiera sumę atrybutów z X i Y (bez powtórzeń),
  - łączone są te krotki, które na wspólnych atrybutach mają jednakowe wartości.
- Iloczyn kartezjański - jeżeli R i S są tabelami rozłącznych typów, to ich złączenie naturalne nazywamy iloczynem kartezjańskim.

- Podzielenie - wyznacza zbiór tych krotek typu U – X, których złączenie z każdą krotką relacji S należy do relacji R.

2. Zależności funkcyjne (ZF):

3. Język SQL

Trigger

```
CRATE TRIGGER SomeName
ON SomeTable
AFTER UPDATE, INSERT
AS
      IF EXISTS (
            SELECT * FROM inserted i
            WHERE i.SomeColumn = 'SOME STRING')
      BEGIN
            RAISERROR('Error Message', 16, 1)
            ROLLBACK TRANSACTION
            RETURN
      END
```

Procedura

```
-- Tworzenie
CREATE PROCEDURE InsertSomeData
      @StringParam NVARCHAR(50),
      @IntParam INT
AS
      SET NOCOUNT ON;
      INSERT INTO SomeTable(SomeStr, SomeInt)
      VALUES (@StringParam, @IntParam)

GO
-- Wywołanie
EXEC InsertSomeData N'String', 42
```

Funkcja

```
CREATE FUNCTION MySum(@a INT, @b INT)
RETURNS INT
AS
BEGIN
      RETURN @a+@b;
END;
```

```
GO
-- Wywołanie
-- Funkcje mogą być wywoływane z użyciem SELECT
SELECT dbo.MySum(20,22)
```

4. Zarządzanie transakcjami współbieżnymi:

Transakcja

Transakcja - transakcją T nazywamy ciąg następujących operacji na wspólnej bazie danych:

- $r_T[x]$ - czytanie (read) danej x przez transakcję T;
- $w_T[x]$ - zapisanie (write) (aktualizacja) danej x przez transakcję T;
- a_t - odrzucenie (abort, rollback) (wycofanie) transakcji T;
- c_T - zatwierdzenie (commit) transakcji T.

Istotą transakcji jest integrowanie kilku operacji w jedną niepodzielną całość.

ACID

Współbieżne wykonywanie transakcji wymaga zachowania własności ACID (Atomicity, Consistency, Isolation, Durability)

Postulaty ACID

- **Atomowość** (atomicity) – każda transakcja stanowi pojedynczą i niepodzielną jednostkę przetwarzania (a także odtwarzania) – w transakcji nie ma więc podtransakcji. Każda transakcja jest bądź wykonana w całości, bądź też żaden jej efekt nie jest widoczny w bazie danych.
- **Spójność** (consistency) – transakcja rozpoczynając się w spójnym stanie bazy danych pozostawia bazę danych w stanie spójnym (tym samym lub innym). Jeśli transakcja narusza warunki spójności bazy danych, to zostaje odrzucona
- **Odizolowanie** (isolation) – zmiany wykonywane przez transakcję nie zatwierdzoną nie są widziane przez inne transakcje (chyba, że przyjęty poziom izolacji na to zezwala).
- **Trwałość** (durability) – zmiany w bazie danych dokonane przez transakcję zatwierdzoną są trwałe w bazie danych, tzn. nawet w przypadku awarii systemu musi istnieć możliwość ich odtworzenia.

Poziomy izolacji

Poziom 0 (READ UNCOMMITED) – najniższy

- dopuszcza czytanie danych niezatwierdzonych tj. danych, które zostały zamienione przez transakcję jeszcze aktywną tzw. “dirty read”. Poziom ten cechuje się wysokim współczynnikiem współbieżności.
- konflikty/problemy:
    - brak odtwarzalności, kaskada odrzuceń, anomalnie powtórnego czytania

Poziom 1 (READ COMMITED)

- dopuszcza czytanie jedynie danych zatwierdzonych, ale pozwala na zapisywanie danych w transakcjach niezatwierdzonych

Poziom 2 (REPEATABLE READ)

- wprowadza zakaz zapisywania w transakcjach niezatwierdzonych
- eliminuje anomalię powtórnego czytania

Poziom 3 (SERIALIZABLE)– najwyższy

- wymaga uwzględnienia warunków, które definiują bazę danych, na których działają transakcje.
- eliminuje wszystkie anomalie

Blokowanie dwufazowe B2F

Każda historia przetwarzania transakcji utworzona przez planistę B2F jest poprawna, ale stosowanie tej metody może powodować zakleszczenia.

Planista

- wyspecjalizowany moduł odpowiedzialny za zarządzanie transakcjami;
- związany jest z menadżerem danych (MD);
- tylu jest planistów ile lokalnych baz danych w systemie.

Działania planisty

- przekazywanie operacji do wykonania menadżerowi danych (MD);
- umieszczanie operacji w kolejce, jeżeli ze względu na konflikt należy poczekać na zakończenie innej operacji;
- odrzucenie operacji, wraz z całą transakcją, jeśli jest to na przykład konieczne z punktu widzenia zakleszczeń

Reguły planisty B2F

- Jeśli operacja $p_i[x]$ może być wykonana, to planista zakłada blokadę:
  - SHARED do odczytu
  - EXCLUSIVE do zapisu

  na daną x dla transakcji T_i i operację tę przekazuje *MD* do wykonania. Jeśli nie może być wykonana, to jest umieszczana w kolejce
- Zdjęcie założonej blokady może nastąpić najwcześniej wtedy, gdy *MD* powiadomi planistę o zakończeniu wykonywania operacji.
- Jeśli nastąpiło zdjęcie jakiejkolwiek blokady założonej dla transakcji T, to dla T nie można już założyć żadnej innej blokady (reguła B2F)
- Trzecia reguła B2F uzasadnia nazwę "blokowanie dwufazowe" ⇒ w procesie wykonywania transakcji można wyróżnić dwie fazy:
  - faza zakładania blokad
  - faza zdejmowania blokad

  Sposób zdejmowania blokad związany jest z realizacją przetwarzania odpowiadającego różnym poziomom izolacji.

Założenia B2F

- Blokadę S *SHARED* dla tej samej danej może uzyskać dowolna liczba transakcji.
- Blokadę X *EXCLUSIVE* dla konkretnej danej może uzyskać tylko jedna transakcja.
- S i X nie mogą być jednocześnie założone na tę samą daną dla dwóch różnych transakcji.
- Uzyskanie blokady S jest konieczne dla odczytania danej.
- Uzyskanie blokady X jest konieczne dla zapisania (modyfikacji, usunięcia) danej.

Poprawność B2F

Podstawową cechą planisty B2F jest, to, że każda historia przetwarzania jest poprawna.

Niech $H = (o_1, o_2, ..., o_m)$ będzie historią przetwarzania dla zbioru transakcji $\Sigma = \{T_1, T_2, ..., T_n\}$.

Definicja

Mówimy, że operacja o_j *poprzedza w H* operację o_k, jeśli $j < k$ oraz operacje te są konfliktowe – stosujemy wówczas zapis $o_j < o_k$.

Definicja

Mówimy, że transakcja *T poprzedza w H* transakcję *T'*, co zapisujemy $T < T'$, jeśli zachodzi jeden z dwóch warunków:

- istnieją w *H* operacje *o* i *o'* pochodzące odpowiednio z *T* i *T'*, takie że $o < o'$, lub
- istnieje transakcja *T''*, taka że $T < T'' < T'$.

Definicja

Historię *H* nazywamy *poprawną*, jeśli dla każdej pary transakcji $T, T' \in \Sigma$ spełniony jest warunek:

- jeśli $T < T'$, to nieprawda, że $T' < T$.

Jeśli historia *H* jest poprawna, to określona na niej relacja „<„ wprowadza częściowy porządek w zbiorze transakcji. Zbiór transakcji możemy wówczas przedstawić w postaci acyklicznego grafu skierowanego zwanego *grafem uszeregowalności (GU) transakcji*.

Twierdzenie

Każda historia przetwarzania utworzona przez planistę B2F jest poprawna.

Dowód poprawności B2F

Pokażemy, że w każdej historii przetwarzania *H* utworzonej przez planistę B2F, dla każdej pary transakcji *T* i *T'* zachodzi warunek:
jeśli $T < T'$, to nieprawda, że $T' < T$.

1. Zauważmy, że jeśli $T < T'$, to założenie jakiejś blokady na rzecz transakcji *T'* musi być poprzedzone zdjęciem jakiejś blokady założonej na rzecz transakcji *T*.
2. Dowód poprowadzimy nie wprost. Przypuśćmy, że w *H* zachodzą: $T < T'$ oraz $T' < T$. Wówczas, uwzględniając p. 1 widzimy, że założenie jakiejś blokady na rzecz transakcji *T* musi być poprzedzone zdjęciem jakiejś blokady założonej na rzecz tej transakcji. Jest to sprzeczne z trzecią regułą B2F.
3. Z 2. wynika zatem, że rozważana historia przetwarzania *H* nie mogła by utworzona przez planistę B2F. Dowodzi to prawdziwości twierdzenia.

Algorytm UNDO/REDO

- Algorytm UNDO/REDO opisuje proces przetwarzania transakcji z uwzględnieniem możliwości odtworzenia bazy danych w przypadku awarii.
- Odtwarzanie bazy danych następuje w chwili restartu systemu po jego awarii.
- Algorytm UNDO/REDO jest stosowany wówczas, gdy w procesie odtwarzania konieczne jest wykonanie operacji UNDO dotyczącej przerwanych transakcji oraz operacji REDO dotyczącej zatwierdzonych transakcji.

Pojęcia:

- **PCh** ⇒ pamięć chwilowa
- **MPCh** ⇒ Mena

UNDO

- **UNDO** oznacza przywrócenie poprzednich (zatwierdzonych) wartości tym danym, które były zmieniane przez transakcje przerwane przed ich zatwierdzeniem.

- Operacja jest konieczna wtedy, gdy strategia wymiany stron w PCh prowadzona jest przez MPCh niezależnie od MO.
- Może więc się zdarzyć, że blok danych z PCh został zapamiętany w bazie danych w momencie, gdy transakcja zapisująca w nim dane nie została jeszcze zatwierdzona. W przypadku awarii należy więc te zmiany wycofać.

REDO

- REDO oznacza ponowne wykonanie operacji aktualizacji danych wykonanych przez transakcje zatwierdzone.
- Operacja ta jest konieczna wtedy, gdy MPCh prowadzi własną strategię wymiany stron.
- Może więc się zdarzyć, że mimo iż transakcja została zatwierdzona, to wykonane przez nią zmiany nie zostały jeszcze zrzucone z PCh do bazy danych. W przypadku awarii nastąpi utrata efektów pracy transakcji.

Optymalizacja

Metody dostępu do danych
Operacja selekcji

S1. *Iteracja*: Odczytuj kolejno wszystkie rekordy pliku i sprawdzaj czy wartości ich atrybutów spełniają warunek selekcji.

S2. *Połowienie binarne*: Jeżeli warunek logiczny operacji selekcji odwołuje się do atrybutu porządkującego plik można zastosować bardziej efektywną metodę wyszukiwania. Za pomocą algorytmu połowienia binarnego znajdź pierwszy rekord, który spełnia warunek zapytania. Następnie odczytaj kolejne rekordy spełniające warunek zapytania.

S3. *Zastosowanie indeksu*: Jeżeli na atrybucie występującym w warunku logicznym selekcji jest założony indeks, to korzystając z tego indeksu znajdź wskaźnik na pierwszy rekord spełniający warunek zapytania. Następnie trawersując po liściach indeksu pobieraj kolejne wskaźniki rekordów spełniających warunek zapytania.

S4. *Zastosowanie funkcji mieszającej*: Jeżeli zapytanie zawiera warunek logiczny zawierający atrybut, który jest argumentem funkcji mieszającej i operatorem relacyjnym jest operator "=" zastosuj funkcję mieszającą i wyznacz wskaźnik na szukany rekord.

S5. *Zastosowanie identyfikatora rekordu*: Odczytaj rekord o danym identyfikatorze.

Estymacja kosztu metod dostępu dla operacji selekcji

Koszty będą określane liczbą operacji I/O

Liczba rekordów relacji – `r`, liczba bloków zajmowanych przez relację – `b`, selektywność zapytania – `s`, wysokość indeksu – `h`, rząd indeksu – `q`.

S1	**b/2** (średnio) – pomyślne odnalezienie rekordu spełniającego warunek *atr* = *x*, gdzie *atr* jest kluczem relacji **b** – wszystkie pozostałe przypadki, niezależnie od selektywności zapytania
S2	$\approx \lceil \log_2 b \rceil + s*b$
S3	$\approx h + s*q^{h-1} + s*r$ lub $\approx h + s*r/q + s*r$ – dla indeksu drugorzędnego $\approx h + s*b$ – dla indeksu podstawowego lub zgrupowanego
S4	≈ 1
S5	**1**

5. Model danych JSON i przykładowe zapytania do bazy dokumentów JSON

JavaScript Object Notation, JSON

Składnia:

- *obiekt* jest nieuporządkowanym zbiorem (może być pustym) par (pól) *nazwa: wartość* ujętymi w nawiasy {},
- *nazwa* jest Stringiem, a wartość jest *skalarem, obiektem* lub *wektorem,*

- *wektor* jest uporządkowanym zbiorem (może być pustym) *wartości* ujętym w nawiasy [],
- *skalar* jest *stringiem, liczbą* lub stałą `true, false, null`.

Przykład:

Zapytania w języku DocumentDB SQL

- **Zwraca cały dokument JSON, Families - nazwa kolekcji dokumentów**

```
SELECT *
FROM Families f
WHERE f.id = "AndersenFamily"
```

- **Zwraca rodziny, które mieszkają w mieście o tej samej nazwie co nazwa stanu, w którym miasto się znajduje**

```
SELECT {
      "Name":f.id,
      "City":f.address.city}
AS Family FROM Families f
WHERE f.address.city = f.address.state
```

- **SELECT tworzy obiekty JSON. Ponieważ nie mamy nazw dla tworzonych wartości (obiektów), to używane są nazwy domyślnych argumentów (implicit argument variable names), a więc $1**

```
SELECT {
      "state": f.address.state,
      "city": f.address.city,
      "name": f.id }
FROM Families f
WHERE f.id = "AndersenFamily"
```

Reprezentacja danych w postaci dokumentu JSON

Przyjmij, że *Student* opisany jest przez: *Nazwisko* i *Kierunek*, a zdawane przez niego *Egzaminy* przez nazwę przedmiotu (*NazPrzed*) i ocenę (*Ocena*). Dla przykładowych studentów (dwóch), z których każdy zdał egzaminy z dwóch przedmiotów podaj sposób reprezentacji tych danych za pomocą Dokumentu JSON (omów składnię)

```
{
   "Studenci":[{
         "Nazwisko":"Mikulski",
         "Kierunek":"Informatyka",
         "Egzaminy":[{
               "NazPrzed":"Bazy Danych",
               "Ocena":5.0
            },{
               "NazPrzed":"Grafika Komputerowa",
               "Ocena":5.0
            }]
      },{
         "Nazwisko":"Włodarczyk",
         "Kierunek":"Informatyka",
         "Egzaminy":[{
               "NazPrzed":"Podstawy Ochrony Danych",
               "Ocena":5.0
            },{
               "NazPrzed":"Inżynieria Oprogramowania",
               "Ocena":4.5
            }]
      }]
}
```

Dokument JSON składa się z pola *Studenci* przechowującego tablicę obiektów JSONowych. Każdy obiekt w tej tablicy posiada pola: *Nazwisko* typu *String*, *Kierunek* typu *String* oraz *Egzaminy* typu *tablica*. Tablica *Egzaminy* obiekty z polami: *NazPrzed* typu *String* oraz pole *Ocena* typu *Number.* Jak widać przedmioty zdane pomiędzy studentami są różne i to jest jedyna informacja o tych przedmiotach w całym dokumencie.

6. Entity Framework i LINQ

Blokowanie optymistycznie w EF

- Domyślnie EF wspiera współbieżność optymistyczną.
- EF zapamiętuje krotki w bazie danych zakładając, że nie zostały zmienione przez inną transakcję od chwili ich wczytania.
- Jeśli stwierdzi, że dane jednak zostały zmienione, to zgłasza wyjątek i należy wtedy rozwiązać konflikt i ponownie spróbować zapisać.

- Współbieżność optymistyczna wymaga istnienia w tabeli kolumny **RowVersion** typu **Timestamp** (lub lepiej użyć typu **RowVersion**).
- Wartość w RowVersion jest automatycznie generowana jako jednoznaczna wartość binarna, gdy wykonywana jest komenda INSERT lub UPDATE.
- Po modyfikacji struktury tabeli aktualizujemy Entity Data Model:
  designer → Update Model From Database → Refresh Student Table.
- Ustaw model współbieżności na ***fixed***, prawy klawisz na własności RowVersion.

Przykładowe zapytania w języku LINQ

Przykładowe klasy na których będę operował:

```
class Student
{
    public int StudentId { get; set; }
    public string Nazwisko { get; set; }
    public string Kierunek { get; set; }
}

class Egzamin
{
    public int StudentId { get; set; }
    public string Przedmiot { get; set; }
    public double Ocena { get; set; }
}
```

Przykładowe dane:

```
var studenci = new List<Student>()
{
    new Student { StudentId = 1, Nazwisko = "Mikulski",
        Kierunek = "Informatyka"},
    new Student { StudentId = 2, Nazwisko = "Włodarczyk",
        Kierunek = "Informatyka"},
    new Student { StudentId = 3, Nazwisko = "Kowalski",
        Kierunek ="Budowa Maszyn"},
    new Student { StudentId = 4, Nazwisko = "Nowak",
        Kierunek ="Elektrotechnika"}
};

var egzaminy = new List<Egzamin>()
{
    new Egzamin { StudentId = 1,
        Przedmiot = "Bazy Danych", Ocena = 4.0},
    new Egzamin { StudentId = 1,
        Przedmiot = "Grafika Komputerowa", Ocena = 5.0},
    new Egzamin { StudentId = 2,
        Przedmiot = "Bazy Danych", Ocena = 4.5},
    new Egzamin { StudentId = 2,
```

```
            Przedmiot = "Podstawy Ochrony Danych", Ocena =3.5},
        new Egzamin {StudentId = 3,
            Przedmiot = "Materiałoznawstwo", Ocena = 3.0 }
    };
```

Selekcja

LINQ

Ze wszystkich studentów, którzy studiują informatykę, wybieram tylko ich Id oraz Nazwiska:

```
var selekcja = from s in studenci
               where s.Kierunek.Equals("Informatyka")
               select new { s.StudentId, s.Nazwisko };
```

Wynik

```
1, Mikulski
2, Włodarczyk
```

Złączenie

LINQ

Łączę studentów z ich egzaminami po Id Studenta:

```
var złączenie = from s in studenci
                join e in egzaminy
                on s.StudentId equals e.StudentId
                select new { s.Nazwisko, e.Przedmiot, e.Ocena };
```

Wyniki

```
Mikulski, Bazy Danych, 4
Mikulski, Grafika Komputerowa, 5
Włodarczyk, Bazy Danych, 4,5
Włodarczyk, Podstawy Ochrony Danych, 3,5
Kowalski, Materiałoznawstwo, 3
```

Grupowanie

LINQ

Grupowanie studentów według ich kierunku oraz sortowanie wyników po nazwie kierunku:

```
var grupowanie = from s in studenci
                 group s by s.Kierunek into kierunki
                 orderby kierunki.Key
                 select kierunki;
```

Wyniki

```
Kierunek: Budowa Maszyn
        3, Kowalski
Kierunek: Elektrotechnika
        4, Nowak
```

```
Kierunek: Informatyka
        1, Mikulski
        2, Włodarczyk
```

Funkcje agregujące

LINQ

Obliczenie średniej ocen dla każdego przedmiotu zdawanego przez wszystkich studentów:

```
var agregacja = from e in egzaminy
                group e by e.Przedmiot into przedmioty
                orderby przedmioty.Key
                select new {
                    przedmiot = przedmioty.Key,
                    średnia = przedmioty.Average(x => x.Ocena)
                };
```

Wyniki

```
Bazy Danych 4,25
Grafika Komputerowa 5,00
Materiałoznawstwo 3,00
Podstawy Ochrony Danych 3,50
```

7. Składnia danych według modelu Key/Value

Model Key/Value

Każdy obiekt - dana według modelu Key-Value składa się z 4. elementów:

- PartitionKey - String jednoznacznie identyfikujący partycję,
- RowKey - String jednoznacznie identyfikujący krotkę (rekord w partycji)
- Timestamp - znacznik czasowym automatycznie aktualizowany przez system (czas ostatniej aktualizacji)
- Value - wartość danej o dowolnej strukturze - krotki z dowolnymi zagnieżdżeniem i z możliwością powtarzania atrybutów.\

Obiekt Key/Value:

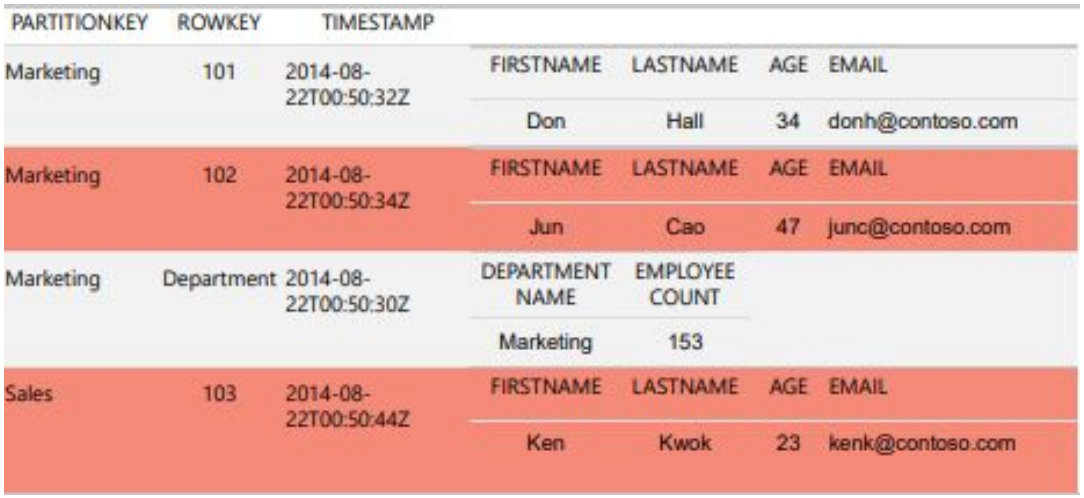
```
(PartitionKey=<string>,
 RowKey=<string>,
 Timestamp=<dateTime>,
 <value>)

value ::=  atrybut : stała  |
           atrybut : value |
```

```
atrybut: value, value
```

Przykład tablicy w Azure Table

Tablica zawierająca cztery jednostki: 3 pracowników i jeden wydział.



PARTITIONKEY	ROWKEY	TIMESTAMP				
Marketing	101	2014-08-22T00:50:32Z	FIRSTNAME	LASTNAME	AGE	EMAIL
			Don	Hall	34	donh@contoso.com
Marketing	102	2014-08-22T00:50:34Z	FIRSTNAME	LASTNAME	AGE	EMAIL
			Jun	Cao	47	junc@contoso.com
Marketing	Department	2014-08-22T00:50:30Z	DEPARTMENT NAME	EMPLOYEE COUNT		
			Marketing	153		
Sales	103	2014-08-22T00:50:44Z	FIRSTNAME	LASTNAME	AGE	EMAIL
			Ken	Kwok	23	kenk@contoso.com

Reprezentacja danych w modelu NoSQL Key/Value

Przyjmij, że *Student* opisany jest przez: *Nazwisko* i *Kierunek*, a zdawane przez niego *Egzaminy* przez nazwę przedmiotu (*NazPrzed*) i ocenę (*Ocena*). Dla przykładowych studentów (dwóch), z których każdy zdał egzaminy z dwóch przedmiotów podaj sposób reprezentacji tych danych za pomocą **modelu NoSQL Key/Value** (omów składnię)

PartitionKey	RowKey	
Informatyka	**POD_Kowalski**	**Ocena** 5.0
Informatyka	**BD_Kowalski**	**Ocena** 4.5
Elektrotechnika	**MZ_Nowak**	**Ocena** 3.5
Elektrotechnika	**TI_Nowak**	**Ocena** 3.0
Student	**Kowalski**	**Kierunek** Informatyka
Student	**Nowak**	**Kierunek** Elektrotechnika

PartitionKey ⇒ przechowuje informacje o egzaminach na danych kierunkach studiów lub w przypadku klucza **Student** informacje o studentach.
RowKey ⇒ dla partycji z nazwą kierunku klucz jest w postaci: *NazPrzed* + *Nazwisko*, a w przypadku partycji **Student** jest równy *Nazwisku*.

8. XML - eXtensible Markup Language

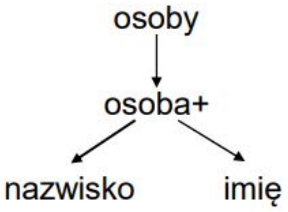
Przykład danej XML-owej:

```
<osoby>
      <osoba>
            <nazwisko>Nowak</nazwisko>
            <imię>Ewa</imię>
      </osoba>
      <osoba>
            <nazwisko>Kubiak</nazwisko>
            <imię>Jan</imię>
      </osoba>
</osoby>
```

DTD (*Document Type Definition*)
- element szczytowy deklarowany jest jako `DOCTYPE`,
- produkcje są deklarowane jako `ELEMENT`,
- wyrażenie puste jest deklarowane jako `#PCDATA`,
- dodatkowo: atrybuty

Przykład:

Gramatyka:
osoby → osoba+
osoba → nazwisko imię
nazwisko → ε
imię → ε



DTD:
```
<!DOCTYPE osoby[
  <!ELEMENT osoby (osoba+)>
  <!ELEMENT osoba (nazwisko imię)>
  <!ELEMENT nazwisko (#PCDATA)>
  <!ELEMENT imię (#PCDATA)>
]>
```

9. Schematy XML

Przykładowy schemat XML (XML Schema)

```
<xs:element name="sampleName">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="field1">
    <xs:simpleType>
     <xs:restriction base="xs:positiveInteger">
      <xs:maxExclusive value="100"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
   <xs:element name="field2" type="xs:string"/>
   <xs:element name="field3" type="xs:decimal"/>
  </xs:sequence>
  <xs:attribute name="atrName" type="niewiem" use="required"/>
 </xs:complexType>
</xs:element>
```

10. Model DOM

Model DOM - Document Object Model

Wg modelu DOM dokument XML przedstawiany jest jako drzewo o 7 typach wierzchołków. Kolejność wierzchołków jest istotna. Na egzamin obowiązują 4 typy wierzchołków:

- root lub document - korzeń drzewa, nie ma znacznika, ma jedno dziecko typu element;
- element - posiada znacznik, może mieć atrybuty i dzieci typu element lub tekst;
- atrybut - ma nazwę i wartość tekstową, ma ojca typu element;
- tekst - nie ma znacznika, ma wartość tekstową. Ma ojca typu element.

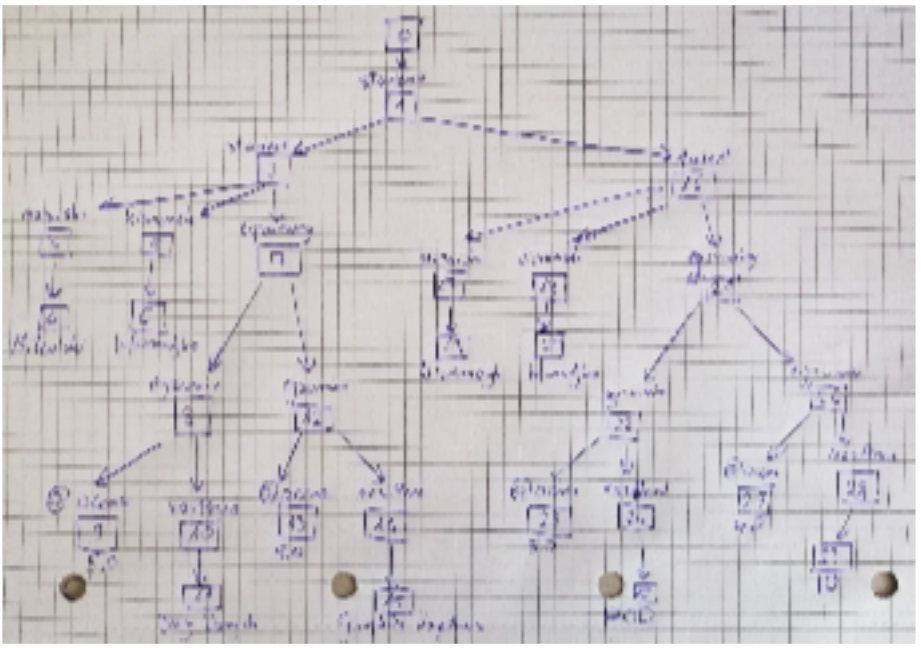
Reprezentacja danych w postaci drzewa DOM

Przyjmij, że *Student* opisany jest przez: *Nazwisko* i *Kierunek*, a zdawane przez niego *Egzaminy* przez nazwę przedmiotu (*NazPrzed*) i ocenę (*Ocena*). Dla przykładowych studentów (dwóch), z których każdy zdał egzaminy z dwóch przedmiotów podaj sposób reprezentacji tych danych za pomocą **drzewa DOM** (ocena ma być atrybutem)(omów składnię)

Dokument XML

```
<studenci>
   <student>
      <nazwisko>Mikulski</nazwisko>
      <kierunek>Informatyka</kierunek>
      <egzaminy>
         <egzamin ocena="5.0">
            <nazPrzed>Bazy Danych</nazPrzed>
         </egzamin>
         <egzamin ocena="5.0">
            <nazPrzed>Grafika Komputerowa</nazPrzed>
         </egzamin>
      </egzaminy>
   </student>
   <student>
      <nazwisko>Włodarczyk</nazwisko>
      <kierunek>Informatyka</kierunek>
      <egzaminy>
         <egzamin ocena="5.0">
            <nazPrzed>Podstawy Ochrony Danych</nazPrzed>
         </egzamin>
         <egzamin ocena="4.5">
            <nazPrzed>Inżynieria Oprogramowania</nazPrzed>
         </egzamin>
      </egzaminy>
   </student>
</studenci>
```

Model DOM

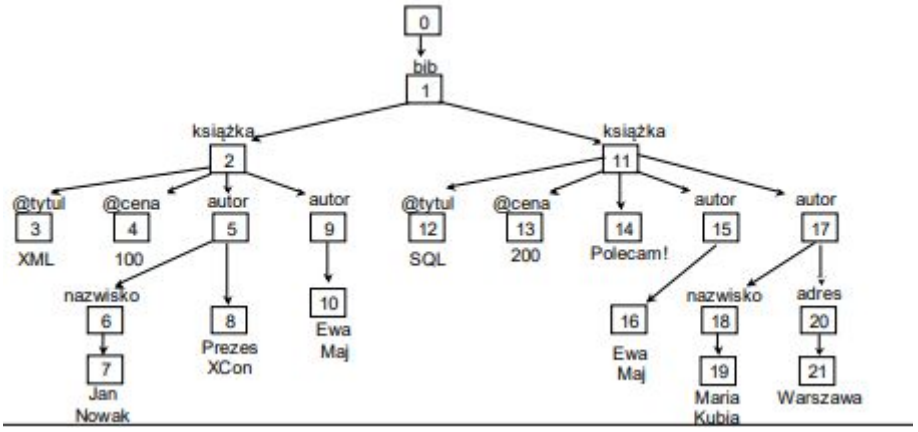


Przykładowy model DOM wraz z dokumentem XML

XML:

```
<bib>
   <książka tytuł="XML" cena="100">
      <autor>
         <nazwisko>Jan Nowak</nazwisko>
         Prezes XCon
      </autor>
      <autor>Ewa Maj</autor>
   </książka>
   <książka tytuł="SQL" cena="200">
      Polecam
      <autor>Ewa Maj</autor>
      <autor>
         <nazwisko>Maria Kubiak</nazwisko>
         <adres>Warszawa</adres>
      </autor>
   </książka>
</bib>
```

Model DOM:

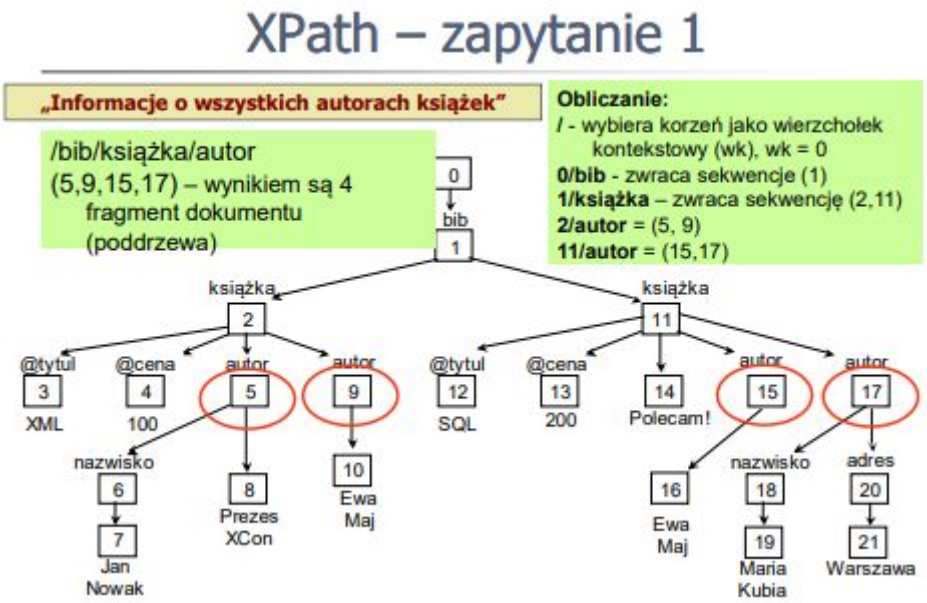


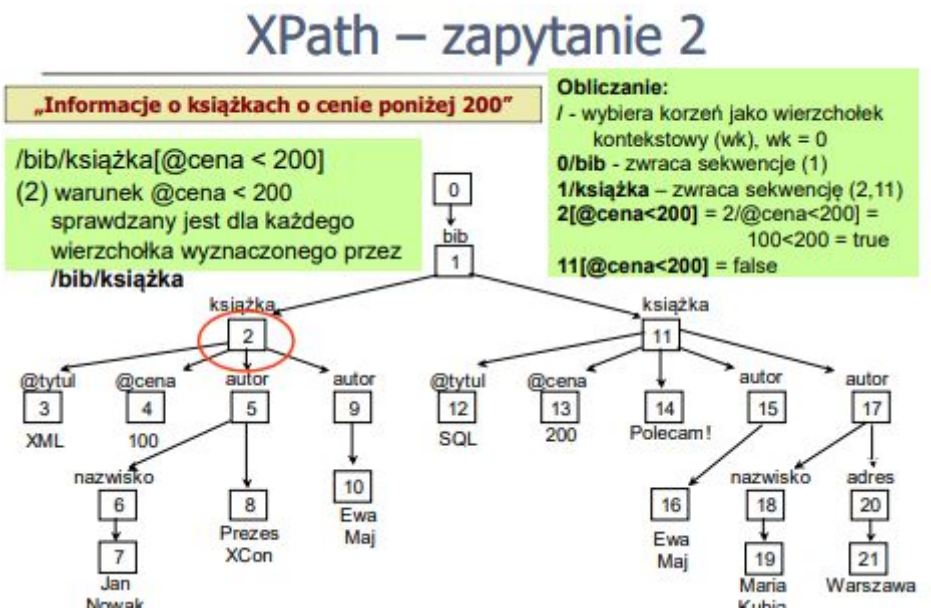
11. Zapytania XPath

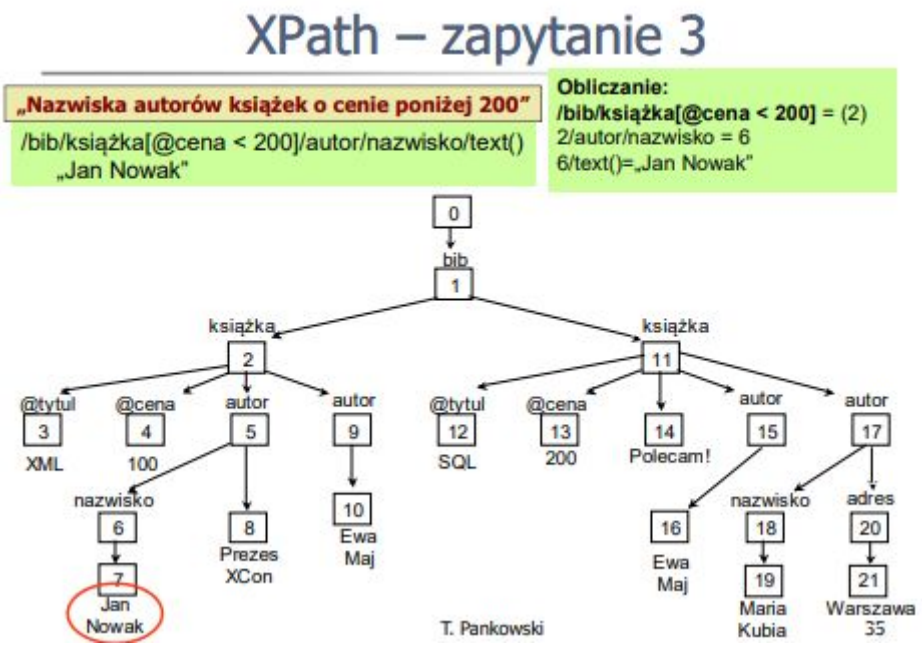
Język XPath:

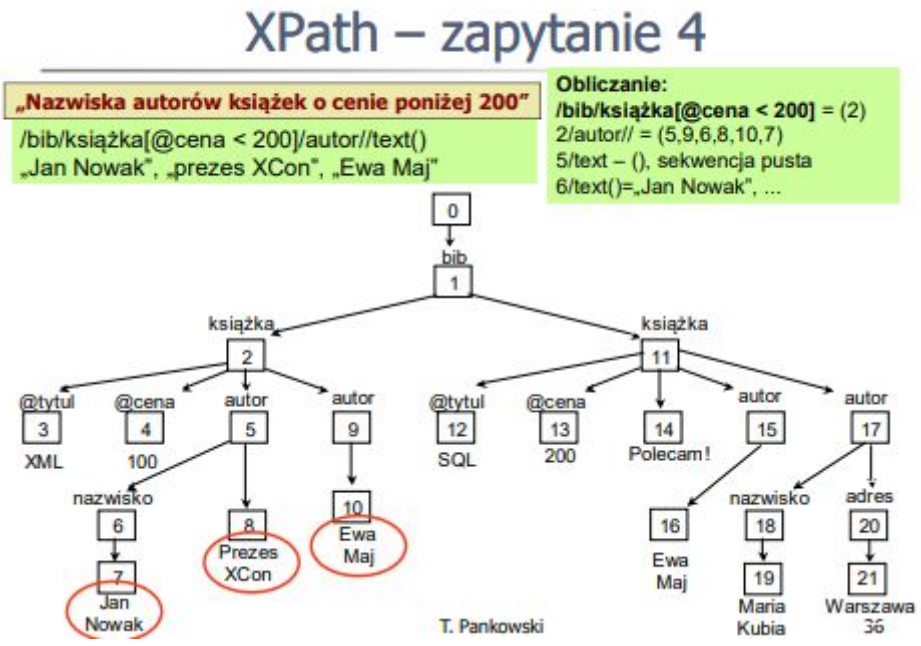
- pozwala wybierać fragment dokument XML-owego,
- wyrażenie XPath definiuje ścieżkę w drzewie danych podając etykiety (znaczniki elementów lub nazwy atrybutów) wierzchołków, kierunki przechodzenia drzewa i warunki jakie mają spełnić wybierane wierzchołki,
- każdy wybrany wierzchołek określa poddrzewo, którego jest korzeniem

Przykłady zapytań:









12. XQuery

XQuery jest językiem zapytań, który:

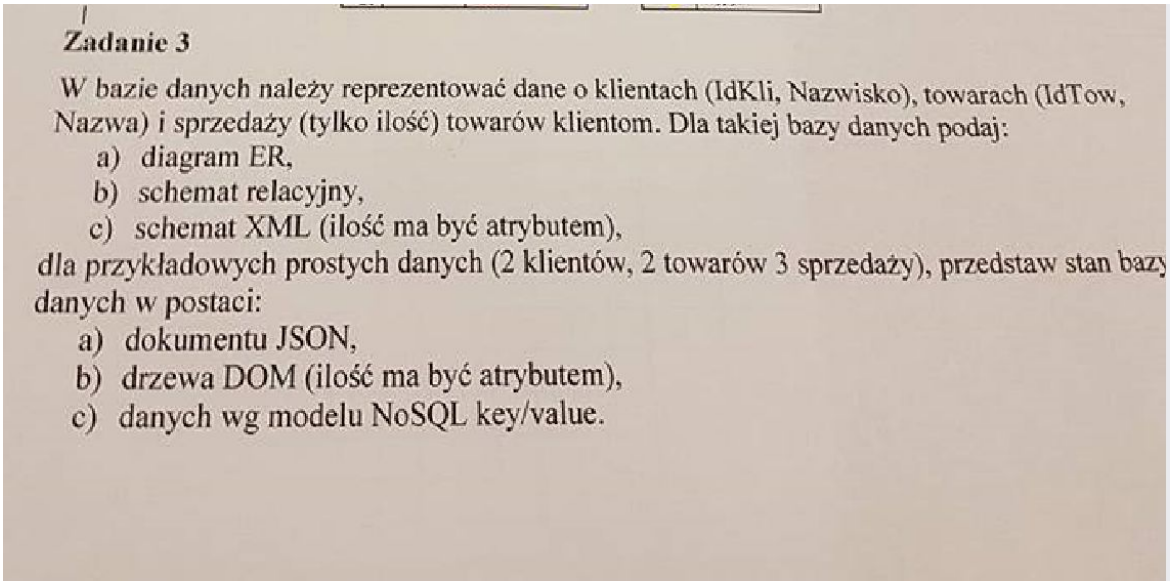
- wybiera elementy/atrybuty z dokumentu wejściowego
- łączy dane z wielu dokumentów wejściowych
- umożliwia modyfikację danych
- wylicza nowe dane
- umożliwia budowanie dokumentu wynikowego
- dodaje nowe elementy/atrybuty do wyniku
- sortuje wynikowy dokument

Typy zapytań XQuery dla przykładowego zapytania - Podaj autorów książek w cenie mniejszej niż 60 zł:

- wyrażenie ścieżkowe:
```
doc("bib.xml")/bib/ksiazka[@cena<60]/autor
```
- wyrażenie FLWOR:
```
for $k in doc("bib.xml")/bib/ksiazka
where $k/@cena<60
return $k/autor
```
- konstruowanie dokumentu:
```
SELECT OpisXML.query('
<wynik>{
      for $k in /bib/ksiazka
      where $k/@cena<60
      return $k/autor}
</wynik>’)
FROM Bib
```

Przykładowe zapytanie XQuery, Dla każdego autora PWN podaj wykaz jego książek:

```
SELECT OpisXML.query('
<wynik>{
 for
  &a in distinct-values(/bib/ksiazk[wydawnictwo/text()="PWN"]/autor/text())
   return
    <wykaz>
     { $a }
     { for $t in /bib/ksiazka[autor/text()=$a]/tytul/text()
       return <tytul>{$t}</tytul> }
    </wykaz>}
</wynik>')
FROM Bib
```



Zadanie 3

W bazie danych należy reprezentować dane o klientach (IdKli, Nazwisko), towarach (IdTow, Nazwa) i sprzedaży (tylko ilość) towarów klientom. Dla takiej bazy danych podaj:

a) diagram ER,
b) schemat relacyjny,
c) schemat XML (ilość ma być atrybutem),

dla przykładowych prostych danych (2 klientów, 2 towarów 3 sprzedaży), przedstaw stan bazy danych w postaci:

a) dokumentu JSON,
b) drzewa DOM (ilość ma być atrybutem),
c) danych wg modelu NoSQL key/value.


```
{
    "sprzedaż": [{
        "ilość": 200,
        "klient": [{
            "IdKli": 1,
            "Nazwa": "Marszałek"
        }],
        "towar": [{
            "IdTow": 6,
            "Nazwa": "Kupa"
        }]
    }, {
        "ilość": 50,
        "klient": [{
            "IdKli": 7,
            "Nazwa": "Zbąszyniak"
        }],
        "towar": [{
            "IdTow": 2,
            "Nazwa": "Deser"
        }]
    }, {
        "ilość": 75,
        "klient": [{
            "IdKli": 69,
            "Nazwa": "Plastuś"
        }],
        "towar": [{
            "IdTow": 8,
            "Nazwa": "Rzynga"
        }]
    }]
}
```


POMYSŁ ROZWIĄZANIA ZAPROPONOWANY PRZEZ KAMILA “**MORSUSA**” ZIELIŃSKIEGO, JAKBY CO TO WIECIE CO Z NIM ZROBIĆ