

Indeksy, optymalizator

Lab 4

Imię i nazwisko: Dariusz Piwowarski, Wojciech Przybytek

Celem ćwiczenia jest zapoznanie się z planami wykonania zapytań (execution plans), oraz z budową i możliwością wykorzystaniem indeksów.

Swoje odpowiedzi wpisuj w miejsca oznaczone jako:

Wyniki:

Ważne/wymagane są komentarze.

Zamieść kod rozwiązania oraz zrzuty ekranu pokazujące wyniki, (dołącz kod rozwiązania w formie tekstowej/źródłowej)

Zwróć uwagę na formatowanie kodu

Oprogramowanie - co jest potrzebne?

Do wykonania ćwiczenia potrzebne jest następujące oprogramowanie

- MS SQL Server,
- SSMS - SQL Server Management Studio
- przykładowa baza danych AdventureWorks2017.

Oprogramowanie dostępne jest na przygotowanej maszynie wirtualnej

Przygotowanie

Uruchom Microsoft SQL Managment Studio.

Stwórz swoją bazę danych o nazwie XYZ.

```
create database xyz
go

use xyz
go
```

Wykonaj poniższy skrypt, aby przygotować dane:

```
select * into [salesorderheader]
from [adventureworks2017].sales.[salesorderheader]
go

select * into [salesorderdetail]
from [adventureworks2017].sales.[salesorderdetail]
go
```

Dokumentacja/Literatura

Celem tej części ćwiczenia jest zapoznanie się z planami wykonania zapytań (execution plans) oraz narzędziem do automatycznego generowania indeksów.

Przydatne materiały/dokumentacja. Proszę zapoznać się z dokumentacją:

- <https://docs.microsoft.com/en-us/sql/tools/dta/tutorial-database-engine-tuning-advisor>
- <https://docs.microsoft.com/en-us/sql/relational-databases/performance/start-and-use-the-database-engine-tuning-advisor>
- <https://www.simple-talk.com/sql/performance/index-selection-and-the-query-optimizer>

Ikony używane w graficznej prezentacji planu zapytania opisane są tutaj:

- <https://docs.microsoft.com/en-us/sql/relational-databases/showplan-logical-and-physical-operators-reference>

Zadanie 1 - Obserwacja

Wpisz do MSSQL Managment Studio (na razie nie wykonuj tych zapytań):

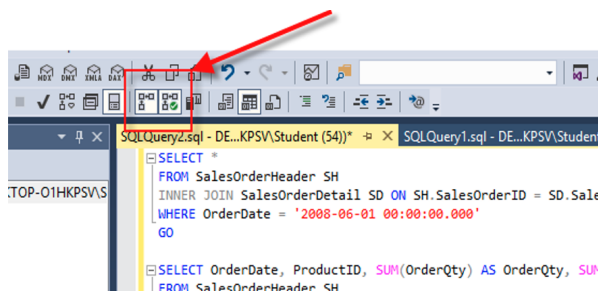
```
-- zapytanie 1
select *
from salesorderheader sh
inner join salesorderdetail sd on sh.salesorderid = sd.salesorderid
where orderdate = '2008-06-01 00:00:00.000'
go

-- zapytanie 2
select orderdate, productid, sum(orderqty) as orderqty,
       sum(unitpricediscount) as unitpricediscount, sum(linetotal)
from salesorderheader sh
inner join salesorderdetail sd on sh.salesorderid = sd.salesorderid
group by orderdate, productid
having sum(orderqty) >= 100
go

-- zapytanie 3
select salesordernumber, purchaseordernumber, due date, shipdate
from salesorderheader sh
inner join salesorderdetail sd on sh.salesorderid = sd.salesorderid
where orderdate in ('2008-06-01', '2008-06-02', '2008-06-03', '2008-06-04', '2008-06-05')
go

-- zapytanie 4
select sh.salesorderid, salesordernumber, purchaseordernumber, due date, shipdate
from salesorderheader sh
inner join salesorderdetail sd on sh.salesorderid = sd.salesorderid
where carriertrackingnumber in ('ef67-4713-bd', '6c08-4c4c-b8')
order by sh.salesorderid
go
```

Włącz dwie opcje: **Include Actual Execution Plan** oraz **Include Live Query Statistics**:

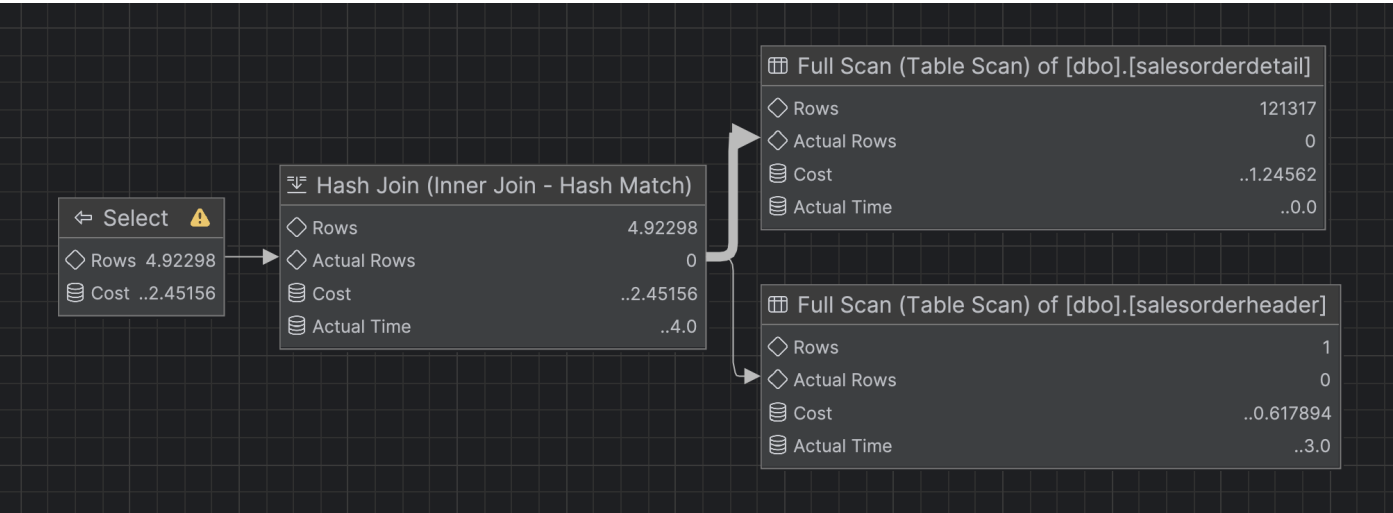


Teraz wykonaj poszczególne zapytania (najlepiej każde analizuj oddzielnie). Co można o nich powiedzieć? Co sprawdzają? Jak można je zoptymalizować?

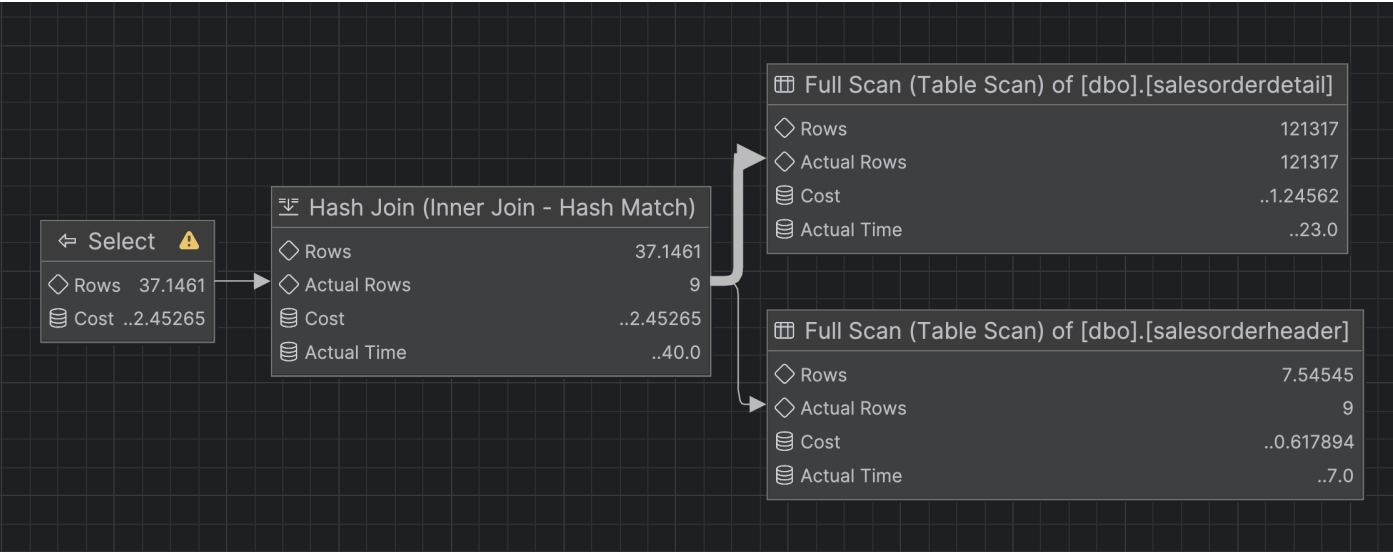
(Hint: aby wykonać tylko fragment kodu SQL znajdującego się w edytorze, zaznacz go i naciśnij F5)

Wyniki:

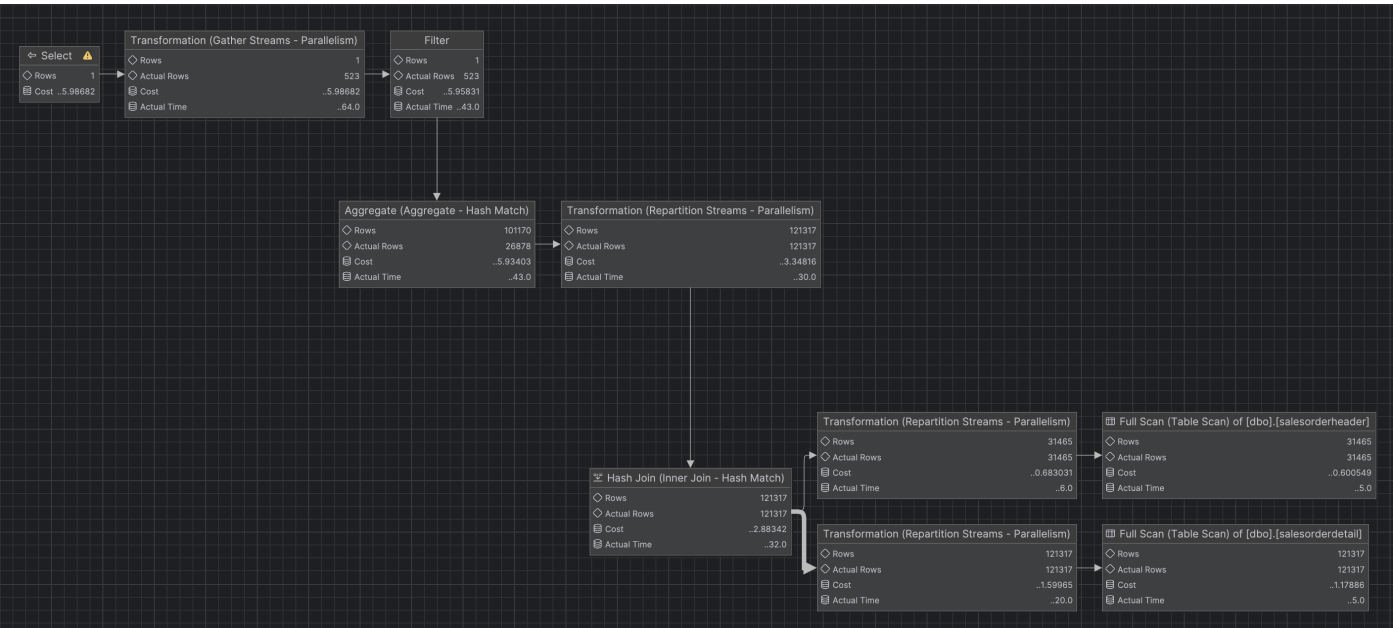
1. Dla podanej daty nie istnieją żadne rekordy i MSSQL jest w stanie zoptymalizować to zapytanie, nie wykonując joina. Jeżeli jednak wybierzemy datę, dla której istnieją rekordy, to MSSQL wykona 2 full scany na obu tabelach. Można to zoptymalizować dodając index do tabeli na kolumnie orderdate żeby szybciej wybrać tylko wiersze z datą która nas interesuje oraz indeks na salesorderid w tabeli salesorderdetail żeby przyspieszyć joina.



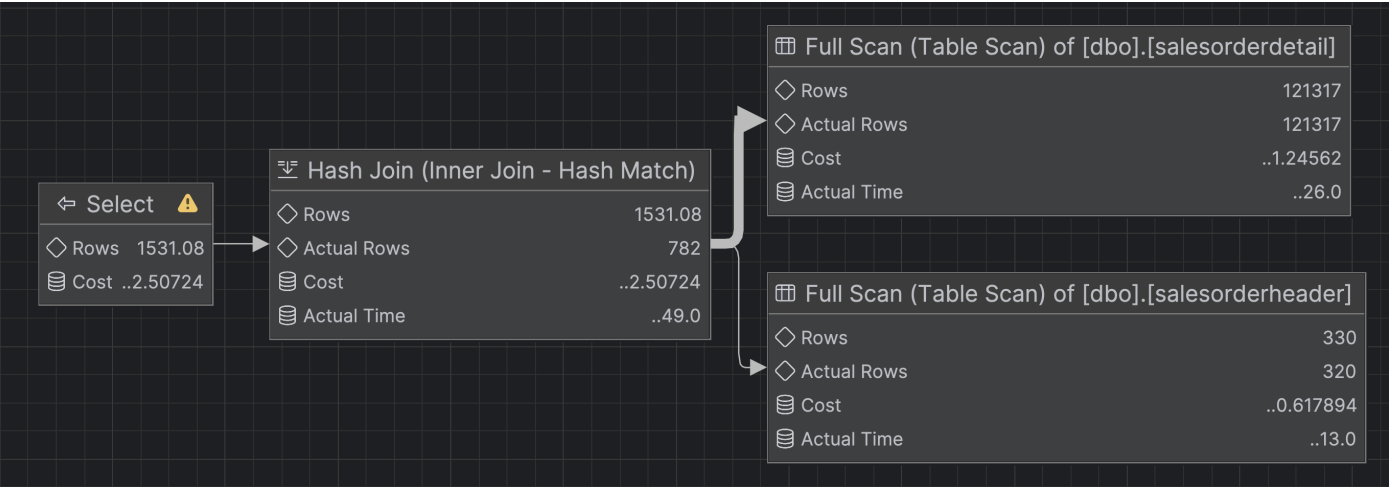
Analiza zapytania z datą, dla której istnieją rekordy:



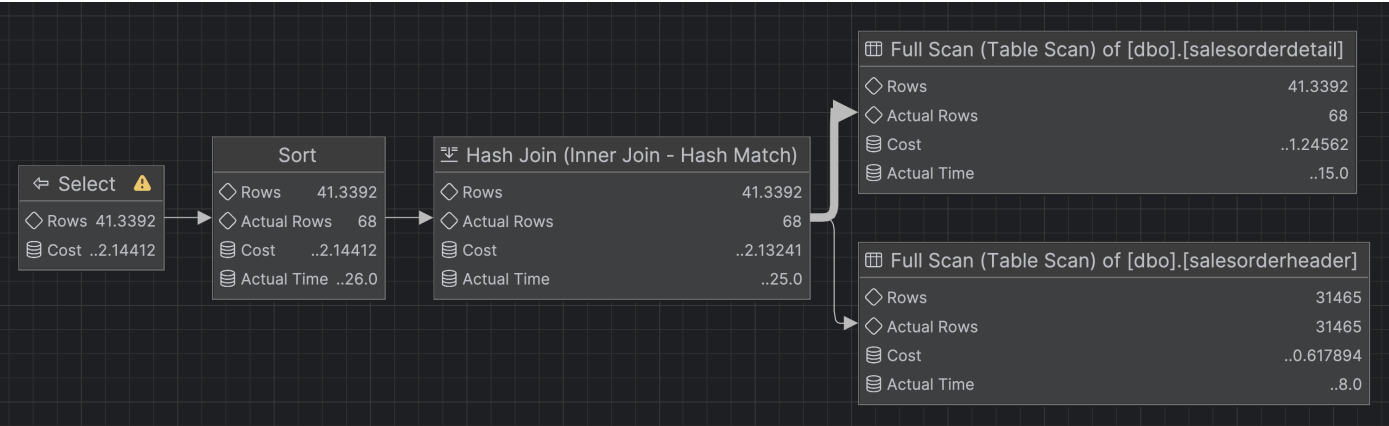
2. Najpierw równolegle wykonywane są Table Scan na tabelach, następnie robiony jest hash match tworzy hash mapę z salesorderid i dopasowuje rekordy z tabel do siebie czyli robi joina. Potem znów przy pomocy hash match-a tworzona jest hash mapa gdzie kluczami są kolumny po których robimy groupby czyli orderdate, productid i potem dopasowywane są rekordy z zjoinowanej tabeli po kluczu i sumowane są tam wartości orderqty, unitpricediscount, i linetotal. Sensowne jest dodanie indeksu na salesorderid żeby przyspieszyć joina, natomiast wydaje nam się że z kolei groupby nie będzie działał szybciej nawet jeśli dodamy indeksy na orderdate, productid, bo i tak musimy przejść przez całą zjoinowaną tabelę żeby te dane zgrupować.



3. Tak samo jak w przypadku 1 (po zmianie daty na istniejące w tabeli) wykonywane są 2 full scany na tabelach salesorderheader i salesorderdetail a następnie robiony jest hash match, który tworzy hash mapę pomiędzy gdzie kluczami są salesorderid z tabeli salesorderheader które spełniają where-a, a następnie jest robione przejście po tabeli salesorderdetail i sprawdzane jest czy jego salesorderid jest w hash mapie i jeśli tak to robiony jest join. Indeksy można dodać tak samo jak 1.

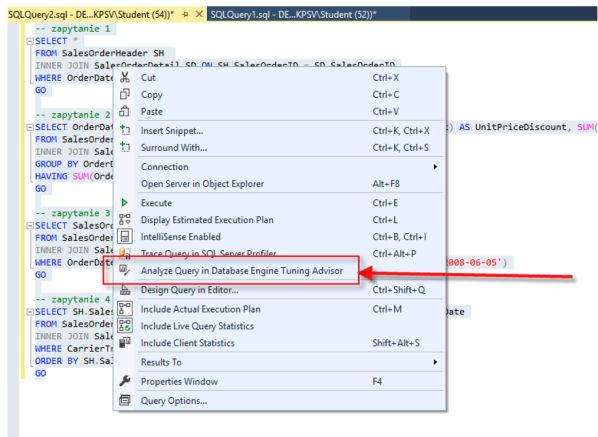


4. Podobnie jak w 1 i 3 mamy 2 full scany gdzie jeden wyciąga wszystkie wiersze z salesorderheader, a drugi tylko te z salesorderdetail które spełniają where-a. Potem tabele są joinowane hash matchem tylko tym razem dopasowywane będą wiersze z salesorderheader, bo jest ich więcej (z salesorderdetail wyciągamy tylko 68 wiersze które spełniają where-a) Na końcu robione jest sortowanie po salesorderid. Jeśli chodzi o optymalizację tego zapytania to w pierwszej kolejności sensowne jest dodanie indeksu na carriertrackingnumber, który przyspieszy where-a, potem aby przyspieszyć joina potrzebny jest indeks na salesorderid w tabeli salesorderheader (mógłby być to nawet klastrowany indeks bo salesorderid wygląda jak dobry kandydat na klucz główny w tej tabeli). Sortowanie można zrobić na końcu na tych kilkudziesięciu wierszach zjoinowanej tabeli i wtedy akurat indeks raczej nic nie przyspieszy.



Zadanie 2 - Optymalizacja

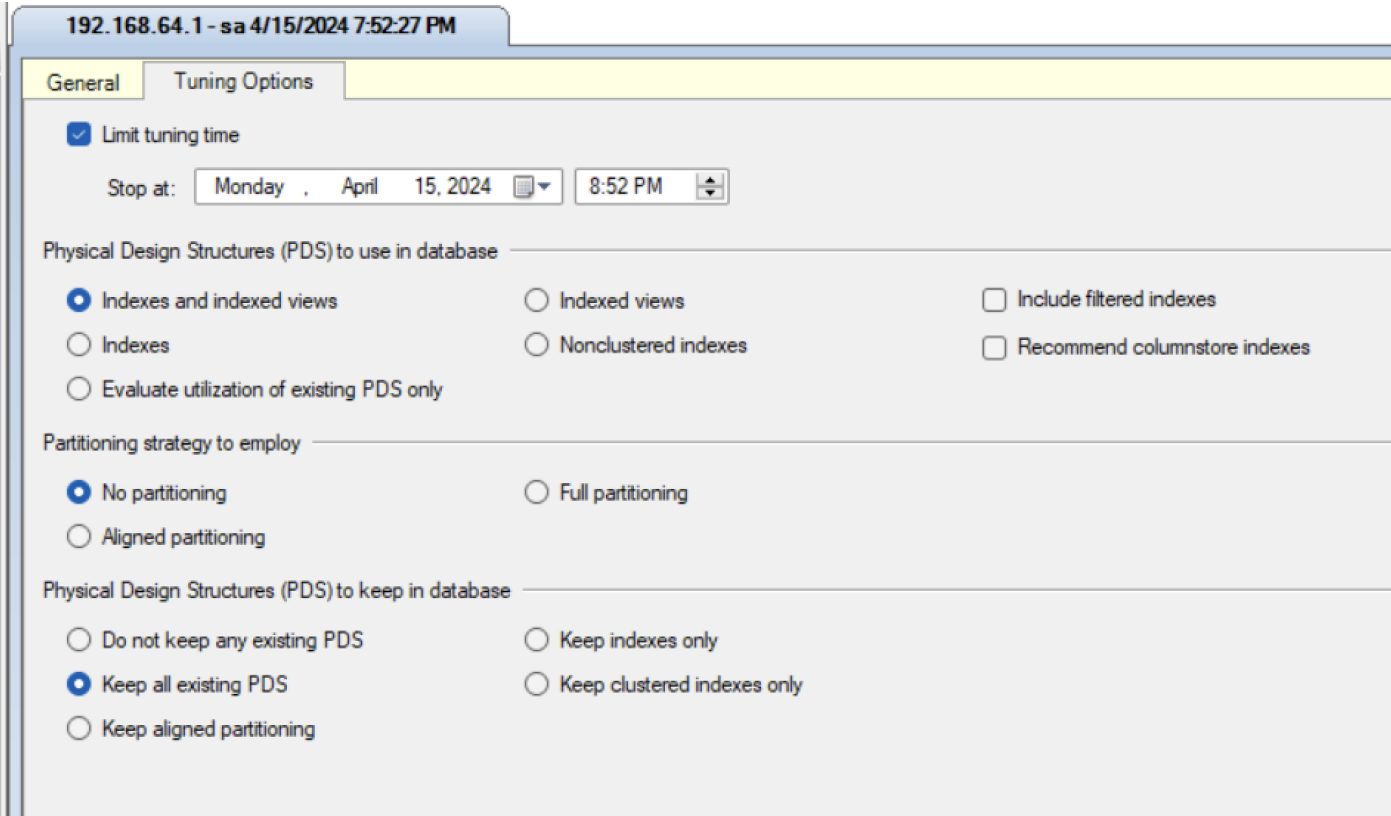
Zaznacz wszystkie zapytania, i uruchom je w **Database Engine Tuning Advisor**:



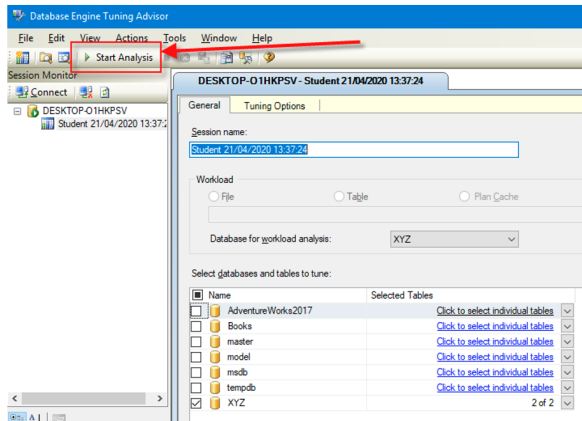
Sprawdź zakładkę **Tuning Options**, co tam można skonfigurować?

Wyniki:

Można w pierwszej kolejności wybrać PDS (Physical Design Structures) które mogą zostać użyte do tuningu. Do wyboru są między innymi indexes, indexed views, indexes and indexed views, nonclustered indexes itd. czyli w skrócie wybieramy tutaj formę w jakiej fizycznie będą zapisane nasze usprawnienia. Możemy też wybrać czy chcemy wykonać partycjonowanie. Na koniec możemy wybrać co się ma stać z istniejącymi PDS, czyli na przykład Keep all existing PDS lub Keep indexes only itd.



Użyj **Start Analysis**:



Zaobserwuj wyniki w **Recommendations**.

Przejdź do zakładki **Reports**. Sprawdź poszczególne raporty. Główną uwagę zwróć na koszty i ich poprawę:

Tuning reports				
Select report: Statement cost report				
Statement Id	Statement String	Percent Improvement	Statement Type	
3	SELECT SalesOrderNumber, Purch...	99.74	Select	
1	SELECT * FROM SalesOrderHeade...	99.73	Select	
4	SELECT SH.SalesOrderID, SalesO...	88.41	Select	
2	SELECT OrderDate, ProductID, S...	19.20	Select	

Zapisz poszczególne rekomendacje:

Uruchom zapisany skrypt w Management Studio.

Opisz, dlaczego dane indeksy zostały zaproponowane do zapytań:

192.168.64.1 - sa 4/15/2024 7:52:27 PM				
General Tuning Options Progress Recommendations Reports				
Estimated improvement: 61%				
Partition Recommendations				
Index Recommendations				
Database Name	Object Name	Recommendation	Target of Recommendation	Details
xyz	[dbo].[salesorderdetail]	create	_dta_index_salesorderdetail_6_1221579390__K3_K1	
xyz	[dbo].[salesorderdetail]	create	_dta_index_salesorderdetail_6_1221579390__K1_K5_4_8_9	
xyz	[dbo].[salesorderdetail]	create	_dta_index_salesorderdetail_6_1221579390__K1_2_3_4_5_6_7_8_9_10_11	
xyz	[dbo].[salesorderdetail]	create	_dta_stat_1221579390_1_3	
xyz	[dbo].[salesorderheader_6_1205579333__K3_K1]	create	_dta_index_salesorderheader_6_1205579333__K3_K1	
xyz	[dbo].[salesorderheader]	create	_dta_index_salesorderheader_6_1205579333__K1_4_5_8_9	
xyz	[dbo].[salesorderheader]	create	_dta_index_salesorderheader_6_1205579333__K3_K1_2_4_5_6_7_8_9_10_11_12_13_14_15_16_17_18_19_20_21_22_23_24_25_26	
xyz	[dbo].[salesorderheader]	create	_dta_stat_1205579333_1_3	

Tuning Reports				
Select report: Statement cost report				
Statement Id	Statement String	Percent Improvement	Statement Type	
3	-- zapytanie 3 select salesordemumb...	99.74	Select	
1	-- zapytanie 1 select * from salesor...	99.73	Select	
4	-- zapytanie 4 select sh.salesorderid,...	93.69	Select	
2	-- zapytanie 2 select orderdate, prod...	19.22	Select	

Wyniki:

Wybraliśmy opcję indexes, bo domyślna opcja indexes and indexed views tworzy widoki, które na przykład w przypadku 2 zapytania z grupy dosłownie stworzyło widok już z joinem i grupby, więc wtedy wystarczyłoby tylko wybrać z niego wherem po warunku dane wiersze, ciekawsze wydają się same indeksy które mogą być bardziej uniwersalne do

różnych zapytań.

Zostało zaproponowane 6 indeksów nonclustered:

Dla tabeli salesorderheader:

- 1. kolumny kluczowe: [OrderDate] ASC, [SalesOrderID] ASC ; kolumny dołączone: pozostałe kolumny tabeli
- 2. kolumny kluczowe: [SalesOrderID] ASC ; kolumny dołączone: [DueDate], [ShipDate], [SalesOrderNumber], [PurchaseOrderNumber]
- 3. kolumny kluczowe: [OrderDate] ASC ; kolumny dołączone: [SalesOrderID]

Dla tabeli salesorderdetails:

- 4. kolumny kluczowe: [SalesOrderID] ASC ; kolumny dołączone: pozostałe kolumny tabeli
- 5. kolumny kluczowe: [SalesOrderID] ASC, [ProductID] ASC ; kolumny dołączone: [OrderQty], [UnitPriceDiscount], [LineTotal]
- 6. kolumny kluczowe: [CarrierTrackingNumber] ASC, [SalesOrderID] ASC

Indeksy 1, 4 optymalizują zapytania 1 i 3.
Indeksy 3, 5 optymalizują zapytanie 2.
Indeksy 2, 6 optymalizują zapytanie 4.
Dokładna analiza tego w jaki sposób je optymalizują znajduje się przy porównaniu Execution Planów poniżej.

```
CREATE STATISTICS [_dta_stat_1221579390_1_3] ON (dbo].[salesorderdetail]([SalesOrderID], [CarrierTrackingNumber])
WITH AUTO_DROP = OFF
go

CREATE NONCLUSTERED INDEX [_dta_index_salesorderheader_6_1205579333_K3_K1_2_4_5_6_7_8_9_10_11_12_13_14_15_16_17_18_19_20_21_22_23_24_25_26] ON (dbo].[salesorderheader]
(
    [OrderDate] ASC,
    [SalesOrderID] ASC
)
INCLUDE([RevisionNumber],[DueDate],[ShipDate],[Status],[OnlineOrderFlag],[SalesOrderNumber],[PurchaseOrderNumber],[AccountNumber],[CustomerID],[SalesPersonID],[TerritoryID],[Bill
go

CREATE NONCLUSTERED INDEX [_dta_index_salesorderheader_6_1205579333_K1_4_5_8_9] ON (dbo].[salesorderheader]
(
    [SalesOrderID] ASC
)
INCLUDE([DueDate],[ShipDate],[SalesOrderNumber],[PurchaseOrderNumber]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go

CREATE NONCLUSTERED INDEX [_dta_index_salesorderheader_6_1205579333_K3_1] ON (dbo].[salesorderheader]
(
    [OrderDate] ASC
)
INCLUDE([SalesOrderID]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go

CREATE STATISTICS [_dta_stat_1205579333_1_3] ON (dbo].[salesorderheader]([SalesOrderID], [OrderDate])
WITH AUTO_DROP = OFF
go
```

```
use [xyz]
go

CREATE NONCLUSTERED INDEX [_dta_index_salesorderdetail_6_1221579390_K1_2_3_4_5_6_7_8_9_10_11] ON (dbo].[salesorderdetail]
(
    [SalesOrderID] ASC
)
INCLUDE([SalesOrderDetailID],[CarrierTrackingNumber],[OrderQty],[ProductID],[SpecialOfferID],[UnitPrice],[UnitPriceDiscount],[LineTotal],[rowguid],[ModifiedDate]) WITH (SORT_IN_
go

CREATE NONCLUSTERED INDEX [_dta_index_salesorderdetail_6_1221579390_K1_K5_4_8_9] ON (dbo].[salesorderdetail]
(
    [SalesOrderID] ASC,
    [ProductID] ASC
)
INCLUDE([OrderQty],[UnitPriceDiscount],[LineTotal]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go

SET ANSI_PADDING ON
go

CREATE NONCLUSTERED INDEX [_dta_index_salesorderdetail_6_1221579390_K3_K1] ON (dbo].[salesorderdetail]
(
    [CarrierTrackingNumber] ASC,
    [SalesOrderID] ASC
)
WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go

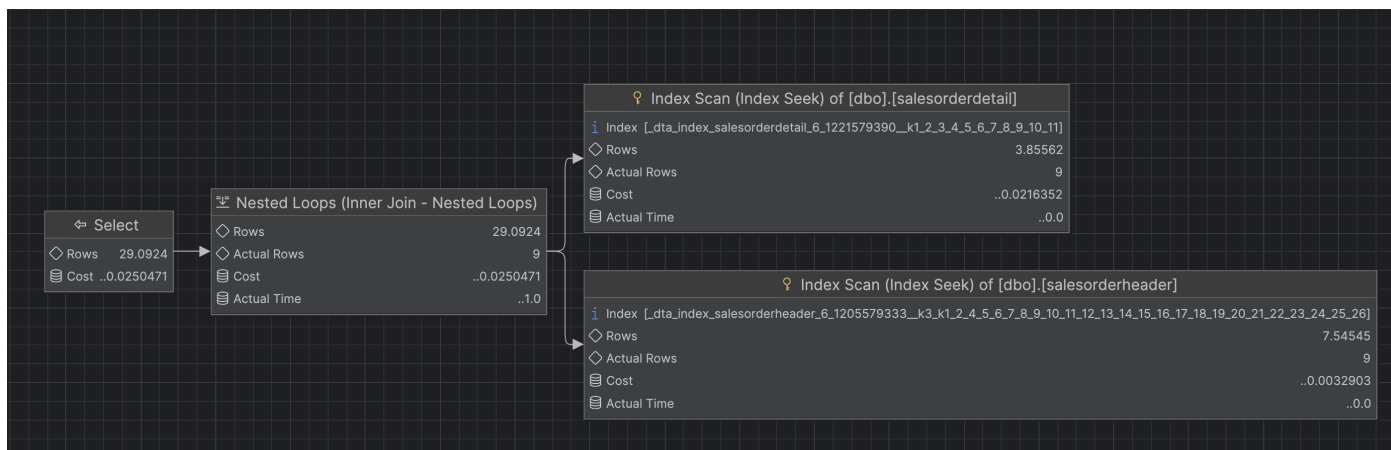
CREATE STATISTICS [_dta_stat_1221579390_1_3] ON (dbo].[salesorderdetail]([SalesOrderID], [CarrierTrackingNumber])
WITH AUTO_DROP = OFF
go
```

Sprawdź jak zmieniły się Execution Plany. Opisz zmiany:

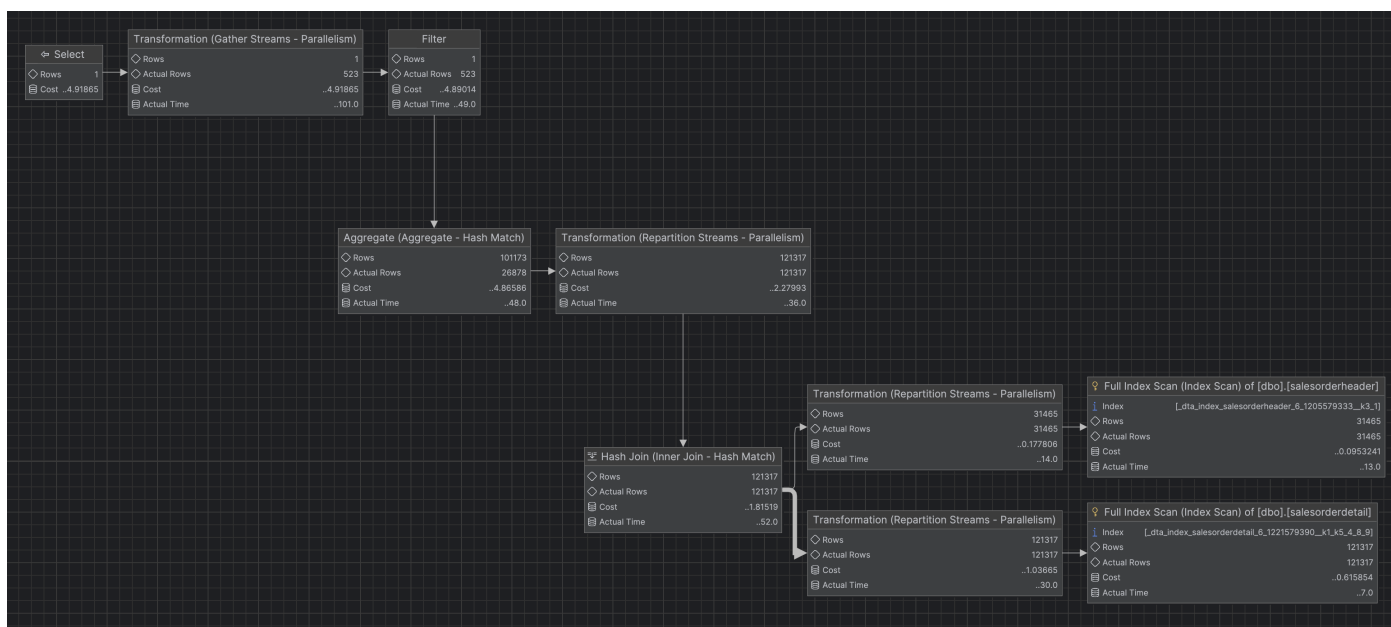
Wyniki:

Indeksy numerujemy jak w powyższej liście indeksów

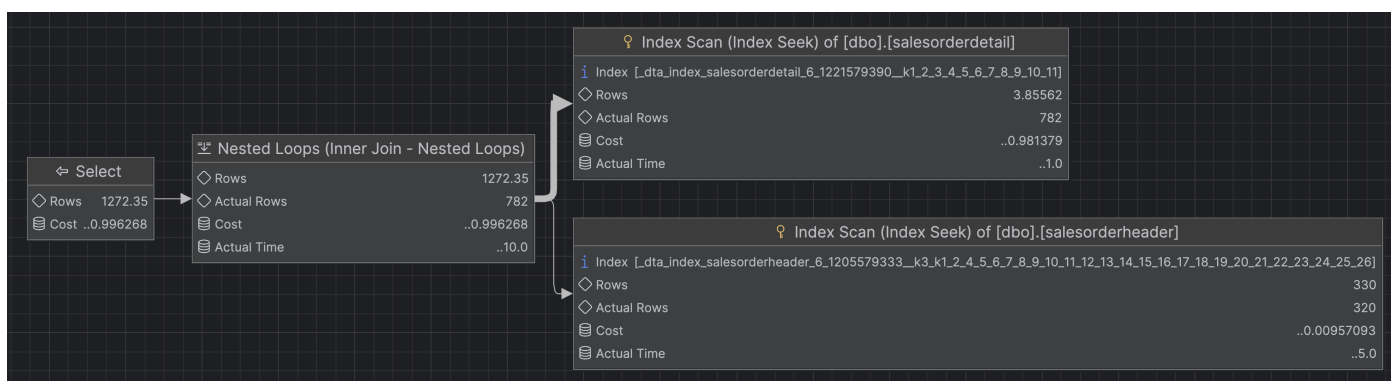
1. Pierwsze zapytanie dzięki indeksom zamiast robić 2 full scany wykorzystuje indeks numer 1 do wyciągnięcia z salesorderheader rekordów spełniających warunek z datą, a następnie indeks numer 4 do wyciągnięcia z salesorderdetails rekordów z pasującym SalesOrderID.
Wykorzystanie tych indeksów powoduje też że zmieniony został sposób wykonywania inner joina z hash match na nested loops, który z wykorzystaniem indeksów jest szybszy (prawdopodobnie z powodu niewielkiej liczby wierszy do której trzeba dopasować wiersz z drugiej tabeli).



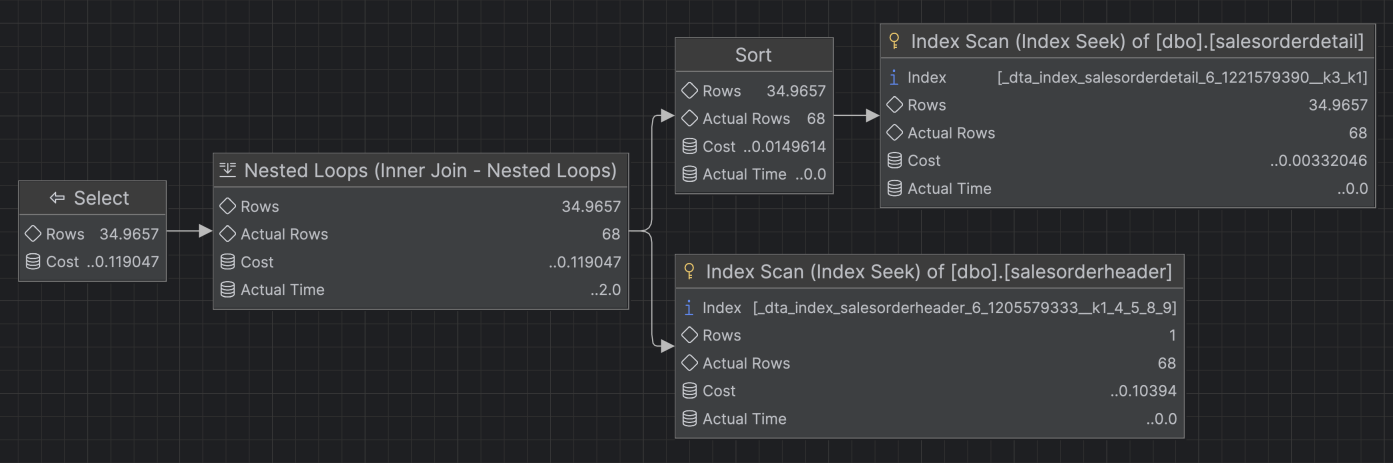
2. Zamiast 2 full scanów wykonywane są 2 full index scany, pierwszy dla tabeli salesorderheader wykorzystuje indeks numer 3 a drugi dla tabeli salesorderdetails używa indeksu numer 5. Indeks 3 wyciąga SalesOrderID posortowane po OrderDate, natomiast indeks 5 sortując po SalesOrderID i ProductID wyciąga dane potrzebne do policzenia sum orderqty, unitpricediscount i linetotal, dzięki temu po joinie tak pobrane dane są już posortowane po OrderDate i ProductID co pozwala na szybsze policzenie sum do groupby.



3. Plan wykonania taki sam jak w 1. oraz wykorzystywane są te same indeksy.



4. Wykonywany jest index seek który wyciąga z tabeli salesorderdetails dane o odpowiednim carriertrackingnumber przy pomocy indeksu numer 6, dane są od razu sortowane po SalesOrderID jeszcze przed joinem. Dla wyciągniętych SalesOrderID robiony jest drugi index seek na tabeli salesorderheader używający indeksu numer 2, który wyciąga pozostałe potrzebne dane. Na koniec robiony jest join i dane po joinie są już posortowane.



Zadanie 3 - Kontrola "zdrowia" indeksu

Dokumentacja/Literatura

Celem kolejnego zadania jest zapoznanie się z możliwością administracji i kontroli indeksów.

Na temat wewnętrznej struktury indeksów można przeczytać tutaj:

- <https://technet.microsoft.com/en-us/library/2007.03.sqlindex.aspx>
- <https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-index-physical-stats-transact-sql>
- <https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-index-physical-stats-transact-sql>
- <https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-indexes-transact-sql>

Sprawdź jakie informacje można wyczytać ze statystyk indeksu:

```
select *
from sys.dm_db_index_physical_stats (db_id('adventureworks2017')
,object_id('humanresources.employee')
,null -- null to view all indexes; otherwise, input index number
,null -- null to view all partitions of an index
,'detailed') -- we want all information
```

Jakie są według Ciebie najważniejsze pola?

Wyniki:

database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_percent	fragment_count	avg_fragment_size_in_pages	page_count	avg_page_space_used_in_percent
5	1893581784	1	1	1 CLUSTERED INDEX	IN_ROW_DATA	2	0	0	1	7	7	7
5	1893581784	1	1	1 CLUSTERED INDEX	IN_ROW_DATA	2	1	0	1	1	1	1
5	1893581784	2	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	1	0	0	1	1	1	1
5	1893581784	3	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	1	0	0	1	1	1	1
5	1893581784	4	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	2	0	33.3333333333333	2	1.5	3	1
5	1893581784	4	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	2	1	0	1	1	1	1
5	1893581784	5	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	2	0	50	2	1	2	1
5	1893581784	5	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	2	1	0	1	1	1	1
5	1893581784	6	1	1 NONCLUSTERED INDEX	IN_ROW_DATA	1	0	0	1	1	1	1

Naszym zdaniem najważniejsze pola to:

- database_id, object_id, index_id (pola pozwalające na identyfikację rekordu)
- index_type_desc (typ indeksu)
- avg_fragmentation_in_percent (średni procent fragmentacji, może sugerować kiedy należy przebudować indeks)
- avg_page_space_used_in_percent (ile procent każdej strony indeksu jest zajęte, dla odczytu lepszy duży procent, dla zapisywania lepszy mały procent)
- record_count (ilość rekordów w indeksie, określa jak duży jest indeks)
- ghost_record_count (ilość zbędnych rekordów, które zostaną usunięte przez system)

Sprawdź, które indeksy w bazie danych wymagają reorganizacji:

```
use adventureworks2017

select object_name([object_id]) as 'table name',
index_id as 'index id'
from sys.dm_db_index_physical_stats (db_id('adventureworks2017')
,null -- null to view all tables
,null -- null to view all indexes; otherwise, input index number
,null -- null to view all partitions of an index
,'detailed') --we want all information
where ((avg_fragmentation_in_percent > 10
and avg_fragmentation_in_percent < 15) -- logical fragmentation
or (avg_page_space_used_in_percent < 75
and avg_page_space_used_in_percent > 60)) --page density
and page_count > 8 -- we do not want indexes less than 1 extent in size
and index_id not in (0) --only clustered and nonclustered indexes
```

Wyniki: zrzut ekranu/komentarz:

Session

Result 10

5 rows

	[table name]	[index id]
1	JobCandidate	1
2	ProductModel	1
3	BillOfMaterials	2
4	WorkOrder	3
5	WorkOrderRouting	2

Zapytanie zwróciło indeksy jako nazwe tabeli i numer indeksu, aby odczytać nazwy indeksów wykonaliśmy dodatkowe zapytanie:

```
select sys.objects.name, sys.indexes.index_id, sys.indexes.name
from sys.indexes
inner join sys.objects on sys.indexes.object_id = sys.objects.object_id
where sys.objects.name in ('JobCandidate', 'ProductModel', 'BillOfMaterials', 'WorkOrder', 'WorkOrderRouting')
```

	objects.name	index_id	indexes.name
1	BillOfMaterials	1	AK_BillOfMaterials_ProductAssemblyID_ComponentID_StartDate
2	BillOfMaterials	2	PK_BillOfMaterials_BillOfMaterialsID
3	BillOfMaterials	3	IX_BillOfMaterials_UnitMeasureCode
4	JobCandidate	1	PK_JobCandidate_JobCandidateID
5	JobCandidate	2	IX_JobCandidate_BusinessEntityID
6	ProductModel	1	PK_ProductModel_ProductModelID
7	ProductModel	2	AK_ProductModel_Name
8	ProductModel	3	AK_ProductModel_rowguid
9	ProductModel	256000	PXML_ProductModel_CatalogDescription
10	ProductModel	256001	PXML_ProductModel_Instructions
11	WorkOrder	1	PK_WorkOrder_WorkOrderID
12	WorkOrder	2	IX_WorkOrder_ScrapReasonID
13	WorkOrder	3	IX_WorkOrder_ProductID
14	WorkOrderRouting	1	PK_WorkOrderRouting_WorkOrderID_ProductID_OperationSequence
15	WorkOrderRouting	2	IX_WorkOrderRouting_ProductID

Ostatecznie odczytaliśmy nazwy indeksów wymagających reorganizacji:

- PK_JobCandidate_JobCandidateID
- PK_ProductModel_ProductModelID
- PK_BillOfMaterials_BillOfMaterialsID
- IX_WorkOrder_ProductID
- IX_WorkOrderRouting_ProductID

Sprawdź, które indeksy w bazie danych wymagają przebudowy:

```
use adventureworks2017

select object_name([object_id]) as 'table name',
index_id as 'index id'
from sys.dm_db_index_physical_stats (db_id('adventureworks2017')
,null -- null to view all tables
,null -- null to view all indexes; otherwise, input index number
,null -- null to view all partitions of an index
,'detailed') --we want all information
where ((avg_fragmentation_in_percent > 15) -- logical fragmentation
or (avg_page_space_used_in_percent < 60)) --page density
```

```
and page_count > 8 -- we do not want indexes less than 1 extent in size
and index_id not in (0) --only clustered and nonclustered indexes
```

Wyniki: zrzut ekranu/komentarz:

Session		Result 41	
[table name]		[index id]	
1	Person	256002	
2	Person	256003	
3	Person	256004	

W analogiczny sposób odczytaliśmy nazwy indeksów wymagających przebudowy:

- XMLPATH_Person_Demographics
- XMLPROPERTY_Person_Demographics
- XMLVALUE_Person_Demographics

Czym się różni przebudowa indeksu od reorganizacji?

(Podpowiedź: <http://blog.plik.pl/2014/12/defragmentacja-indeksow-ms-sql.html>)

Wyniki:

Reorganizacja defragmentuje poszczególne strony indeksu, nie usuwa jednak fragmentacji pomiędzy stronami i nie zmienia ich struktury. Jest to proces nie zajmujący dużo dodatkowej pamięci. Przebudowa tworzy nowy indeks, a następnie usuwa stary. W jej trakcie eliminowana jest cała fragmentacja nie tylko ze stron, ale również między stronami. Jest to jednak proces bardziej kosztowny i wymagający więcej dodatkowej pamięci niż reorganizacja.

Sprawdź co przechowuje tabela sys.dm_db_index_usage_stats:

Wyniki:

database_id	object_id	index_id	user_seeks	user_scans	user_lookups	user_updates	last_user_seek	last_user_scan	last_user_lookup	last_user_update	system_seeks	system_scans	sys
4	402816497	2	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.276	0	0	
4	402816497	1	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.276	0	0	
4	18815129	1	1	0	0	0	1 2024-04-15 19:57:28.256	<null>	<null>	2024-04-15 17:54:29.316	0	0	
4	64719283	3	0	3	0	0	0 <null>	2024-04-15 17:57:09.886	<null>	<null>	0	0	
5	1810105489	1	0	1	0	0	0 <null>	2024-04-15 17:39:36.672	<null>	<null>	0	0	
4	66815308	1	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.337	0	0	
4	66815308	2	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.337	0	0	
4	482816782	1	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.357	0	0	
4	482816782	2	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.357	0	0	
4	1627152842	1	1	0	0	0	0 2024-04-15 17:45:43.137	<null>	<null>	<null>	0	0	
4	528728936	1	12	0	0	0	0 2024-04-15 17:57:09.886	<null>	<null>	<null>	0	0	
4	114815471	2	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.347	0	0	
4	114815471	1	0	0	0	0	1 <null>	<null>	<null>	2024-04-15 17:54:29.347	0	0	
4	1398296841	1	18	36	0	4	2024-04-15 17:54:40.937	2024-04-15 21:47:23.726	<null>	2024-04-15 17:54:29.523	0	1	
6	1285579333	0	0	38	0	0	0 <null>	2024-04-15 21:03:29.713	<null>	<null>	0	11	
6	1221579390	0	0	10	0	0	0 <null>	2024-04-15 21:03:29.713	<null>	<null>	0	9	
4	1691153070	1	0	1	0	0	0 <null>	2024-04-15 17:45:43.137	<null>	<null>	0	0	
4	162815642	3	0	0	0	1	1 <null>	<null>	2024-04-15 17:54:29.246	2024-04-15 17:54:29.246	0	0	
4	162815642	1	0	0	1	1	1 <null>	<null>	2024-04-15 17:54:29.246	2024-04-15 17:54:29.246	0	0	
4	162815642	2	1	0	0	1	1 2024-04-15 17:54:29.246	<null>	<null>	2024-04-15 17:54:29.246	0	0	
5	1922105888	1	0	1	0	0	0 <null>	2024-04-15 17:39:36.516	<null>	<null>	0	0	
4	1878297751	1	2	0	0	0	0 2024-04-15 17:54:40.937	<null>	<null>	<null>	0	0	
4	1910297865	2	1	0	0	1	1 2024-04-15 17:54:25.507	<null>	<null>	2024-04-15 17:54:25.483	0	0	
4	1910297865	1	6	1	0	1	1 2024-04-15 17:54:29.246	2024-04-15 17:54:29.176	<null>	2024-04-15 17:54:25.483	0	1	
4	1526296497	1	11	0	0	11	2024-04-15 17:54:40.733	<null>	<null>	2024-04-15 17:54:29.366	0	0	
4	1958298836	1	2	1	0	0	0 2024-04-15 17:54:29.283	2024-04-15 17:54:29.177	<null>	<null>	0	0	
4	306816155	2	0	0	0	1	1 <null>	<null>	<null>	2024-04-15 17:54:29.327	0	0	
4	306816155	1	0	0	0	1	1 <null>	<null>	<null>	2024-04-15 17:54:29.327	0	0	
4	2086298207	1	0	2	0	0	0 <null>	2024-04-15 17:54:29.283	<null>	<null>	0	0	
4	1622296839	1	4	34	0	1	1 2024-04-15 17:54:40.976	2024-04-15 21:47:23.726	<null>	2024-04-15 17:54:29.413	0	0	
4	354816326	1	0	0	1	1	1 <null>	<null>	2024-04-15 17:54:29.227	2024-04-15 17:54:29.217	0	0	
4	354816326	2	1	0	0	1	1 2024-04-15 17:54:29.227	<null>	<null>	2024-04-15 17:54:29.217	0	0	
4	3054308428	1	0	1	0	1	1 <null>	2024-04-15 19:54:30.306	<null>	2024-04-15 19:54:30.306	0	0	

Przechowuje informacje na temat liczby użyć i daty ostatniego użycia operacji seek, scan, lookup i update na poszczególnych indeksach przez użytkownika i system.

Napraw wykryte błędy z indeksami ze wcześniejszych zapytań. Możesz użyć do tego przykładowego skryptu:

```
use adventureworks2017

--table to hold results
declare @tablevar table(lngid int identity(1,1), objectid int,
index_id int)

insert into @tablevar (objectid, index_id)
select [object_id],index_id
from sys.dm_db_index_physical_stats (db_id('adventureworks2017')
,null -- null to view all tables
,null -- null to view all indexes; otherwise, input index number
,null -- null to view all partitions of an index
,'detailed') --we want all information
where ((avg_fragmentation_in_percent > 15) -- logical fragmentation
or (avg_page_space_used_in_percent < 60)) --page density
and page_count > 8 -- we do not want indexes less than 1 extent in size
and index_id not in (0) --only clustered and nonclustered indexes

select 'alter index ' + ind.[name] + ' on ' + sc.[name] + '.'
+ object_name(objectid) + ' rebuild'
from @tablevar tv
inner join sys.indexes ind
on tv.objectid = ind.[object_id]
and tv.index_id = ind.index_id
inner join sys.objects ob
on tv.objectid = ob.[object_id]
inner join sys.schemas sc
on sc.schema_id = ob.schema_id
```

Napisz przygotowane komendy SQL do naprawy indeksów:

Wyniki:

```
alter index XMLPATH_Person_Demographics on Person.Person rebuild
alter index XMLPROPERTY_Person_Demographics on Person.Person rebuild
alter index XMLVALUE_Person_Demographics on Person.Person rebuild
alter index PK_JobCandidate_JobCandidateID on HumanResources.JobCandidate reorganize
alter index PK_ProductModel_ProductModelID on Production.ProductModel reorganize
alter index PK_BillOfMaterials_BillOfMaterialsID on Production.BillOfMaterials reorganize
alter index IX_WorkOrder_ProductID on Production.WorkOrder reorganize
alter index IX_WorkOrderRouting_ProductID on Production.WorkOrderRouting reorganize
```

Zadanie 4 - Budowa strony indeksu

Dokumentacja

Celem kolejnego zadania jest zapoznanie się z fizyczną budową strony indeksu

- <https://www.mssqltips.com/sqlservertip/1578/using-dbcc-page-to-examine-sql-server-table-and-index-data/>
- <https://www.mssqltips.com/sqlservertip/2082/understanding-and-examining-the-uniqueifier-in-sql-server/>
- <http://www.sqlskills.com/blogs/paul/inside-the-storage-engine-using-dbcc-page-and-dbcc-ind-to-find-out-if-page-splits-ever-roll-back/>

Wypisz wszystkie strony które są zaalokowane dla indeksu w tabeli. Użyj do tego komendy np.:

```
dbcc ind ('adventureworks2017', 'person.address', 1)
-- '1' oznacza nr indeksu
```

Zapisz sobie kilka różnych typów stron, dla różnych indeksów:

Wyniki:

PageID	PageID	PageID	PageID	ObjectID	IndexID	PartitionNumber	PartitionID	Ism_chain_Type	PageType	IndexLevel	NextPageID	NextPageID	PrevPageID	PrevPageID
1	1	10674	<null>	1029578706	1	1	72057594047889408	In-row data	10	<null>	0	0	0	0
2	1	11712	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11713	1
3	1	11713	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11714	1
4	1	11714	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11715	1
5	1	11715	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11716	1
6	1	11716	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11717	1
7	1	11717	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11718	1
8	1	11718	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11719	1
9	1	11719	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11720	1
10	1	11720	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11721	1
11	1	11721	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11722	1
12	1	11722	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11723	1
13	1	11723	1	10674	1029578706	1	1	72057594047889408	In-row data	1	0	1	11724	1

- 11712 person.address, 1, PageType 1
- 17856 production.workorder, 3, PageType 2
- 1168 person.person, 1, PageType 3
- 1135 person.emailaddress, 1, PageType 10

Włącz flagę 3604 zanim zaczniesz przeglądać strony:

```
dbcc traceon (3604);
```

Sprawdź poszczególne strony komendą DBCC PAGE. np.:

```
dbcc page('adventureworks2017', 1, 13720, 3);
```

Zapisz obserwacje ze stron. Co ciekawego udało się zaobserwować?

Wyniki:

```
> dbcc page('adventureworks2017', 1, 11712, 3)
PAGE: (1:11712)
BUFFER:
BUF @0x00000000650F3300
bpage = 0x00000000E1965E00          bPmmpage = 0x0000000000000000          bsort_r_nextbP = 0x0000000000000000
bsort_r_prevbP = 0x0000000000000000 bhash = 0x0000000000000000          bpageno = (1:11712)
bpart = 4                          bstat = 0x9                          breferences = 0
berrcode = 0                       bUse1 = 22500                        bstat2 = 0x0
blog = 0x15a                       bsampleCount = 1                    bIoCount = 0
resPoolId = 0                      bcputicks = 130                     bReadMicroSec = 753
bDirtyPendingCount = 0             bDirtyContext = 0x0000000000000000 bDbPageBroker = 0x0000000000000000
bdbid = 6                          bpru = 0x00000000E04CA0040
PAGE HEADER:
Page @0x00000000E1965E00
m_pageId = (1:11712)               m_headerVersion = 1                 m_type = 1
m_typeFlagBits = 0x0              m_level = 0                         m_flagBits = 0x200
m_objId (AllocUnitId.idObj) = 268 m_indexId (AllocUnitId.idInd) = 256 Metadata: AllocUnitId = 7205759405491584
Metadata: PartitionId = 72057594047889408 Metadata: IndexId = 1
Metadata: ObjectId = 1029578706    m_prevPage = (1:12010)              m_nextPage = (1:11713)
pminlen = 36                      m_slotCnt = 58                     m_freeCnt = 32
m_freeData = 8044                 m_reservedCnt = 0                   m_lsn = (37:1228:307)
m_xactReserved = 0                m_xdesId = (0:1390)                m_ghostRecCnt = 0
m_tornBits = 2108258024           DB Frag ID = 1
Allocation Status
GAM (1:2) = ALLOCATED              SGAM (1:3) = NOT ALLOCATED          PFS (1:8088) = 0x40 ALLOCATED    0_PCT_FULL
DIFF (1:6) = NOT CHANGED          ML (1:7) = NOT MIN_LOGGED
Slot 0 Offset 0x60 Length 128     Record Attributes = NULL_BITMAP VARIABLE_COLUMNS
Record Type = PRIMARY_RECORD
Record Size = 128
```

```
Memory Dump @0x000000066E2A6060
0000000000000000: 30002400 346b0000 0e000000 aa9a06fd 12ad4c4a 0.$.4k.....ª .ý.LJ
0000000000000014: a54823b3 5dfcb241 00000000 d0a00000 09000400 ¥H#³]þ²A....Ð .. ...
0000000000000028: 05005800 58006200 6a008000 35003100 31003700 ..X.X.b.j...5.1.1.7.
000000000000003c: 20004f00 61006b00 20004300 72006500 65006b00 .0.a.k. .C.r.e.e.k.
0000000000000050: 20004300 74002e00 4c006500 65006400 73004c00 .C.t...L.e.e.d.s.L.
0000000000000064: 45003100 3800e610 0000010c 72c2b224 2fe04a40 E.1.8.æ.....rÂ²$/àJ@
0000000000000078: 16902e7e b1b7f8bf ...~±·ø¿
Slot 0 Column 1 Offset 0x4 Length 4 Length (physical) 4
AddressID = 27444
Slot 0 Column 2 Offset 0x34 Length 36 Length (physical) 36
AddressLine1 = 5117 Oak Creek Ct.
...
```

```
> dbcc page('adventureworks2017', 1, 1135, 3)
PAGE: (1:1135)
BUFFER:
BUF @0x0000000650F16000
bpage = 0x0000000E19EF0000      bPmmpage = 0x0000000000000000      bsort_r_nextbP = 0x0000000000000000
bsort_r_prevbP = 0x0000000000000000 bhash = 0x0000000000000000      bpageno = (1:1135)
bpart = 4                        bstat = 0x9                      breferences = 1
berrcode = 0                     bUse1 = 22626                     bstat2 = 0x0
blog = 0x15a                     bsampleCount = 0                 bIoCount = 0
resPoolId = 0                    bcputicks = 0                    bReadMicroSec = 1074
bDirtyPendingCount = 0           bDirtyContext = 0x0000000000000000 bDbPageBroker = 0x0000000000000000
bdbid = 6                         bpru = 0x0000000E04CA0040
PAGE HEADER:
Page @0x0000000E19EF0000
m_pageId = (1:1135)              m_headerVersion = 1              m_type = 10
m_typeFlagBits = 0x0             m_level = 0                       m_flagBits = 0x200
m_objId (AllocUnitId.idObj) = 289 m_indexId (AllocUnitId.idInd) = 256 Metadata: AllocUnitId = 72057594056867840
Metadata: PartitionId = 72057594049134592 Metadata: IndexId = 1
Metadata: ObjectId = 1845581613 m_prevPage = (0:0) m_nextPage = (0:0)
pminlen = 90                     m_slotCnt = 2                     m_freeCnt = 6
m_freeData = 8182                m_reservedCnt = 0                 m_lsn = (37:2043:182)
m_xactReserved = 0               m_xdesId = (0:0)                 m_ghostRecCnt = 0
m_tornBits = -734821128          DB Frag ID = 1
Allocation Status
GAM (1:2) = ALLOCATED             SGAM (1:3) = NOT ALLOCATED        PFS (1:1) = 0x70 IAM_PG MIXED_EXT ALLOCATED 0_PCT_FULL
DIFF (1:6) = NOT CHANGED          ML (1:7) = NOT MIN_LOGGED
IAM: Header @0x000000066AE16064 Slot 0, Offset 96
sequenceNumber = 0                status = 0x0                      objectId = 0
indexId = 0                       page_count = 0                    start_pg = (1:0)
IAM: Single Page Allocations @0x000000066AE1608E
Slot 0 = (0:0)                    Slot 1 = (0:0)
Slot 3 = (0:0)                    Slot 4 = (0:0)
Slot 6 = (0:0)                    Slot 7 = (0:0)
Slot 2 = (0:0)
Slot 5 = (0:0)
IAM: Extent Alloc Status Slot 1 @0x000000066AE160C2
(1:0) - (1:808) = NOT ALLOCATED
(1:816) - (1:840) = ALLOCATED
(1:848) - = NOT ALLOCATED
(1:856) - = ALLOCATED
(1:864) - (1:904) = NOT ALLOCATED
(1:912) - (1:1120) = ALLOCATED
(1:1128) - = NOT ALLOCATED
(1:1136) - = ALLOCATED
(1:1144) - (1:33784) = NOT ALLOCATED
```

Strona zawiera następujące sekcje: numer strony, buffer, header, allocation status i dane. Dane podzielone są na sloty i kolumny, przy czym każda kolumna ma podaną długość. Istnieją również puste sloty, które nie mają kolumn. Nieco inna jest ostatnia ze sprawdzanych stron o typie 10, czyli IAM (Index Allocation Map). Opisuje ona jakie strony są zaalokowane w jednostce alokacji przypisanej do danej strony IAM.

Punktacja:

zadanie	pkt
1	3
2	3
3	3
4	1
razem	10