Лабораторна робота 4
ДОСЛІДЖЕННЯ ІНДЕКСНИХ СТРУКТУР У SQL SERVER

Мета роботи: досліджувати розміри та поведінку індексних структур та невпорядкованих таблиць SQL Server.

Завдання на лабораторну роботу:

1. Дослідження кучі.
2. Дослідження кластеризованого індексу
3. Дослідження некластеризованого індексу.
4. Створення індексів у БД (за індивідуальним варіантом).

Хід роботи

1. ДОСЛІДЖЕННЯ КУЧІ

Структуру SQL Server простіше зрозуміти за допомогою прикладів. Наступний код створює таблицю, організовану як кучі.

```
create table test_index ( _
id int not null _
pole1 char ( 36 ) not null _
pole2 char ( 216 ) not null
)
```

Якщо ви не створюєте кластеризований індекс явно або неявно за допомогою первинного ключа або обмеження унікальності, таблиця буде організована як куча SQL Server не виділяє таблиці жодних сторінок під час її створення. Він виділяє першу сторінку, а також першу IAM-сторінку, коли виконується перша вставка рядка в таблицю. Загальну інформацію про таблиці та індекси можна знайти у представлені каталогу **sys. indexes**.

Наступний запит отримує основну інформацію про таблицю dbo.TestStructure яка була створена у попередньому коді.

```
select OBJECT_NAME ( object_id ) as table_name ,
name as index_name , type , type_desc
from sys . indexes
where OBJECT_ID = OBJECT_ID ( N'test_index ' )
```

Ось як виглядають результати цього запиту.

	table_name	index_name	type	type_desc
1	TestStructure	NULL	0	HEAP

Стовпець type зберігає значення 0 для куч, 1 для кластеризованих таблиць (індексів) і 2 для некластеризованих індексів. Можна дізнатися, скільки сторінок виділено під об'єкт, з функції динамічного управління **sys.dm_db_index_physical_stat** або за допомогою системної процедури **dbo.sp _spaceused** як показано в наступному коді. Оскільки цей код використовується багато разів у цій лабораторній, для простоти посилаємось на нього як на "перевірку виділення кучі".

```
select index_type_desc,  page_count,
record_count, avg_page_space_used_in_percent
from sys.dm_db_index_physical_stats
(db_id(N'test_index_db'), OBJECT_ID(N'test_index'), Null,
 Null, 'Detailed')

exec dbo.sp_spaceused @objname = N'test_index', @updateusage = true;
```

Вихідні дані цих двох команд виглядають так:

	index_type_desc	page_count	record_count	avg_page_space_used_in_percent
1	HEAP	0	0	0

	name	rows	reserved	data	index_size	unused
1	test_index	0	0 KB	0 KB	0 KB	0 KB

Ви бачите, що таблиця порожня, а порожня таблиця не займає місця. Зверніть увагу на останній стовпець першого запиту, avg_space_used_in_percent . Цей стовпець демонструє внутрішню фрагментацію. *Внутрішня фрагментація* означає, що сторінки не заповнені. Чим більше рядків ви зберігаєте на одній сторінці, тим менше сторінок SQL Server повинен читати для вилучення цих рядків, і тим менше пам'яті він використовує для кешованих сторінок при однаковій кількості рядків. У кучах ви не отримаєте великої внутрішньої фрагментації, оскільки SQL Server, як ви вже знаєте, зберігає нові рядки на існуючих сторінках, якщо там є достатньо місця. Тепер вставте перший рядок.

```
insert into test_index
values ( 1 , ' a' , 'b ' )
```

Якщо ви виконаєте код перевірки виділення кучі знову, результат буде наступним:

	index_type_desc	page_count	record_count	avg_page_space_used_in_percent
1	HEAP	1	1	3,24932048430937

	name	rows	reserved	data	index_size	unused
1	test_index	1	72 KB	8 KB	8 KB	56 KB

Таблиця з одним рядком займає одну сторінку. Середнє використання сторінки є низьким, оскільки на сторінці лише один рядок. Результат процедури dbo.sp _ spaceused показує, що таблиця має дві зарезервовані сторінки: одна для даних і одна для першої сторінки IAM. Ви бачите, що SQL Server виділяє лише сторінку, але не екстент для цієї таблиці. Тепер заповніть сторінку за допомогою наступного коду:

Після виділення кучі знову запустіть код. Ви отримаєте наступний результат:

```
declare @i as int = 31
while @ i < 240
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , ' a' , 'b ' )
```

```
        end ;
```

	index_type_desc	page_count	record_count	avg_page_space_used_in_percent
1	HEAP	1	30	98,1961947121324

	name	rows	reserved	data	index_size	unused
1	test_index	30	72 KB	8 KB	8 KB	56 KB

Як і раніше виділено лише одну сторінку; але ця сторінка немає внутрішньої фрагментації, оскільки сторінка неспроможна розмістити додаткові рядки. Спробуйте вставити додаткові рядки.

```
insert into test_index
values ( 31 , ' a' , 'b ' )
```

Код перевірки виділення купи повертає наступні вихідні дані:

	index_type_desc	page_count	record_count	avg_page_space_used_in_percent
1	HEAP	2	31	50,7227575982209

	name	rows	reserved	data	index_size	unused
1	test_index	31	72 KB	16 KB	8 KB	48 KB

Тепер ви бачите, що виділено одну додаткову сторінку зі змішаного екстенту. Зрозуміло, внутрішня фрагментація зросла, оскільки друга сторінка майже порожня. Заповніть 8 сторінок за допомогою наступного коду:

```
declare @i as int = 31
while @ i < 240
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , ' a' , 'b ' )
        end ;
```

Результати виконання коду перевірки виділення кучі виглядають наступним чином:

```
select index_type_desc,  page_count,
record_count, avg_page_space_used_in_percent
from sys.dm_db_index_physical_stats
(db_id(N'test_index_db'), OBJECT_ID(N'test_index'), Null,
 Null, 'Detailed')

exec dbo.sp_spaceused @objname = N'test_index', @updateusage = true;
```

8 сторінок заповнено. Що станеться, якщо ви вставите ще один рядок? Спробуйте це зробити за допомогою наступного коду:

```
insert into test_index
values ( 241 , ' a' , 'b ' )
```

Результати виконання коду перевірки виділення кучі виглядають наступним чином:

	index_type_desc	page_count	record_count	avg_page_space_used_in_percent
1	HEAP	2	32	52,3597726711144

	name	rows	reserved	data	index_size	unused
1	test_index	32	72 KB	16 KB	8 KB	48 KB

Тепер ви бачите, що хоча таблиця займає лише 9 сторінок, під неї зарезервовані 16 сторінок даних плюс перша сторінка IAM. Як показує результат процедури dbo.sp _ spaceused, SQL Server зарезервував 136 Кбайт під таблицю, що означає 17 сторінок; 56 Кбайт, як і раніше, не використовуються. Невикористані 56 Кбайт простору означають, що 7 сторінок з однорідного екстенту, як і раніше, порожні. Перші 8 сторінок розміщуються у змішаних екстентах. Оскільки таблиця вже більше 8 сторінок, SQL Server виділяє однорідні екстенти для додаткового простору, який необхідно.

2. ДОСЛІДЖЕННЯ КЛАСТЕРИЗОВАНОГО ІНДЕКСУ

Наступний код усікає створену та заповнену таблицю та реорганізує цю таблицю у збалансоване дерево за допомогою стовпця id як ключа кластеризації.

```
truncate table test_index

create clustered index idx_cl_id on test_index ( id )
```

	table_name	index_name	type	type_desc
1	test_index	idx_cl_id	1	CLUSTERED

Як бачите, значення type змінилося на 1 і куча більше не існує. Коли створюєте кластеризований індекс, то фактично реорганізуєте таблицю. Тепер заповніть 621 сторінку цієї таблиці, використовуючи унікальні значення для ключа кластеризації.

```
declare @i as int = 0
while @ i < 18630
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , ' a' , 'b ' )
        end ;
```

Зверніть увагу, якщо ви знаєте, що значення мають бути унікальними, ви повинні створити первинний ключ або обмеження унікальності цієї таблиці. Ви можете створити унікальний індекс; але оскільки унікальність обмежує значення, слід використовувати обмеження.

Основну інформацію про індекс можна отримати, запросивши функцію динамічного - управління **sys.dm_db_index_physical_stats**. Наступний фрагмент коду буде використовуватися в цій лабораторній багато разів, тому надалі називатимемо його кодом "перевірки виділення кластеризованого індексу".

```
select index_type_ desc ,  index _depth , index_level , page_count ,
record_count , avg_page_space_used_in_percent ,
        avg_fragmentation_in_percent
```

```
from sys . dm_db_index_physical_stats
( db_id ( N'test_index_db ' ), OBJECT_ID ( N'dbo.test_index ' ), Null ,
 Null , ' Detailed ' )

exec dbo . sp _spaceused @objname _ = N'test_index ' , @updateusage _ = true ;
```

Результат виконання коду перевірки виділення кластеризованого індексу буде наступним:

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	2	0	621	18630	98,1961947121324	0,161030595813205
2	CLUSTERED INDEX	2	1	1	621	99,7158388930072	0

	name	rows	reserved	data	index_size	unused
1	test_index	18630	5000 KB	4968 KB	16 KB	16 KB

Ви бачите, що індекс має лише два рівні: кінцевий рівень та кореневу сторінку. Коренева сторінка має 621 рядок, які вказують на 621 кінцеву сторінку. І тут внутрішня фрагментація відсутня. Тепер вставте ще один рядок.

```
insert into test_index
        values ( 18631 , ' a' , 'b ' )
```

Виконавши код перевірки виділення кластеризованого індексу, ви отримаєте наступні вихідні дані:

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	3	0	622	18631	98,0435507783543	0,321543408360129
2	CLUSTERED INDEX	3	1	2	622	49,9258710155671	0
3	CLUSTERED INDEX	3	2	1	2	0,296515937731653	0

	name	rows	reserved	data	index_size	unused
1	test_index	18631	5064 KB	4976 KB	32 KB	56 KB

Тепер індекс має три рівні. Оскільки нова сторінка була виділена на кінцевому рівні, початкова коренева сторінка не може вже посилатися на всі кінцеві сторінки. SQL Server додав проміжний рівень із двома сторінками, що вказують на 622 кінцеві сторінки, та нову кореневу сторінку, що вказує на дві сторінки проміжного рівня.

Щоб продемонструвати вплив uniquifier, наступний код усікає таблицю та заповнює 423 сторінки за допомогою неунікальних значень ключа кластеризації.

```
truncate table test_index
declare @i as int = 0
while @ i < 8906
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @ i % 100 , ' a' , 'b ' )
        end ;
```

Якщо ви виконаєте код перевірки виділення кластеризованого індексу, то отримайте - наступний результат:

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	2	0	423	8906	70,9655670867309	99,5271867612293
2	CLUSTERED INDEX	2	1	1	423	99,8393872003954	0

	name	rows	reserved	data	index_size	unused
1	test_index	8906	3400 KB	3384 KB	16 KB	0 KB

Зауважте, що коренева сторінка може посилатися лише на 423 сторінки кінцевого рівня. Для заповнення двох рівнів індексу потрібно було лише 8908 рядків, тоді як з унікальними значеннями ключа кластеризації в попередньому прикладі SQL Server міг

розмістити 18630 рядків на двох рівнях.

Щоб довести це, додайте ще один рядок.

```
insert into test_index
        values ( 8909 % 100 , ' a' , 'b ' )
```

Код перевірки виділення кластеризованого індексу повертає такі вихідні дані:

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	2	0	423	8906	70,9655670867309	99,5271867612293
2	CLUSTERED INDEX	2	1	1	423	99,8393872003954	0

	name	rows	reserved	data	index_size	unused
1	test_index	8906	3400 KB	3384 KB	16 KB	0 KB

Ви бачите, що якщо значення ключа не є унікальними, SQL Server повинен додати додатковий рівень до індексу набагато раніше.

До цього часу значення ключа кластеризації були послідовними.

Що станеться, якщо вони не будуть послідовними? Наступний код усікає таблицю dbo.Teststructure, видаляє існуючий кластеризований індекс, створює новий за допомогою стовпця filierl як ключ кластеризації, а потім вставляє 9000 рядків у таблицю з унікальними послідовними значеннями в ключі кластеризації.

```
truncate table test_index
drop index idx_cl_id on test_index
create clustered index idx_cl_pole1 on test_index ( pole1 )

declare @i as int = 0
while @ i < 9000
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , format ( @i , '0000' ), 'b' )
        end ;
```

Тепер перевірте фрагментацію. Наш код, який ми далі називатимемо кодом перевірки фрагментації, перевіряє внутрішню фрагментацію (стовпчик avg_page_space_used_in_percent) і зовнішню фрагментацію (стовпець avg fragmentation in відсоток).

Вихідні дані коду перевірки фрагментації в цьому випадку будуть такими:

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	3	0	300	9000	98,1961947121324	1
2	CLUSTERED INDEX	3	1	3	300	55,5720286632073	0
3	CLUSTERED INDEX	3	2	1	3	1,64319248826291	0

	name	rows	reserved	data	index_size	unused
1	test_index	9000	2440 KB	2400 KB	40 KB	0 KB

Ви бачите, що індекс має три рівні. На кінцевому рівні внутрішня фрагментація - відсутня; крім того, зовнішньої фрагментації також майже немає. Усі сторінки на кінцевому рівні заповнені, і фізична послідовність майже така сама, як і логічна. Тепер виконайте усічення таблиці та заповніть її випадковими значеннями в стовпці filierl.

Наступний код використовує функцію **NEWID( )** мови T-SQL, яка генерує ідентифікатори GUID і зберігає їх у стовпці fillerl .

```
truncate table test_index
declare @i as int = 0
while @ i < 9000
```

```
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , cast ( newid () as char ( 36 )), 'b' )
        end ;
```

```
    INSERT INTO dbo.TestStructure (id, filler1, filler2)
      VALUES (@i, CAST(NEWID() AS CHAR(36)), 'b');
END;
```

Ідентифікатори GUID, згенеровані функцією NEWID( ), майже випадкові. Якщо ви виконаєте код перевірки фрагментації знову, то отримайте наступний результат:

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	3	0	441	9000	66,792241166296	98,4126984126984
2	CLUSTERED INDEX	3	1	4	441	61,2706943414875	50
3	CLUSTERED INDEX	3	2	1	4	2,19915987150976	0

	name	rows	reserved	data	index_size	unused
1	test_index	9000	3592 KB	3528 KB	48 KB	16 KB

Ви бачите, що сторінки кінцевого рівня займають лише 68% простору, заповненого рядками. Це тому, що SQL Server виконав розбиття кількох сторінок. З іншого боку, зовнішня фрагментація становить близько 99%; майже жодна сторінка не розташована у правильному логічному порядку. Ви бачите, що використання ідентифікаторів GUID як ключі кластеризації можуть призвести до створення досить неефективних індексів. Зовнішня фрагментація в основному уповільнює перегляди, що часто не повинно відбуватися в середовищах OLTP, які дуже важливі для сховищ даних. Внутрішня фрагментація є проблемою в обох сценаріях, тому що таблиця набагато більша, ніж вона має бути з послідовним ключем.

Фрагментацію можна позбутися, якщо перебудувати або реорганізувати індекс. Реорганізація індексу - процес повільніший, але менш ресурсомісткий. У загальному випадку, слід виконувати реорганізацію індексу, якщо зовнішня фраг ментація менша за 30%, і перебудовувати його, якщо вона більша за 30%. Наступний код змінює індекс.

```
alter index idx_cl_pole1 on test_index rebuild
```

Якщо ж ви бажаєте реорганізувати таблицю, вам потрібно просто замінити ключове слово REBUILD ключовим словом REORGANIZE . Якщо ви виконаєте код перевірки фрагментації після перебудови індексу, побачите у вихідних даних, що фрагментація практично зникла.

	index_type_desc	index_depth	index_level	page_count	record_count	avg_page_space_used_in_percent	avg_fragmentation_in_percent
1	CLUSTERED INDEX	3	0	300	9000	98,1961947121324	0
2	CLUSTERED INDEX	3	1	2	300	83,3703978255498	0
3	CLUSTERED INDEX	3	2	1	2	1,08722510501606	0

	name	rows	reserved	data	index_size	unused
1	test_index	9000	2504 KB	2400 KB	32 KB	72 KB

3. ДОСЛІДЖЕННЯ НЕКЛАСТЕРНОГО ІНДЕКСУ

За аналогією із завданнями 1,2 використовуючи наступний код досліджувати некластерні індекси та зробити відповідні пояснення:

```
-- індекс на купі
 drop index idx_cl_pole1 on test_index

 create nonclustered index idx_ncl_pole1 on test_index ( pole1 )
 truncate table test_index

declare @i as int = 0
```

```
while @ i < 24472
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , format ( @i , '0000' ), 'b' )
        end ;

insert into test_index
                values ( 24473 , ' 000024473' , 'b ' )


drop index idx_cl_id on test_index

 create clustered index idx_cl_pid on test_index ( id )
 create nonclustered index idx_ncl_pole1 on test_index ( pole1 )
 truncate table test_index

declare @i as int = 0
while @ i < 28864
begin
        set @ i = @ i + 1 ;
        insert into test_index
                values ( @i , format ( @i , '0000' ), 'b' )
        end ;

        insert into test_index
                values ( 28865 , ' 000028865' , 'b ' )

        select * from test_index
        select index_type_ desc ,  index _depth , index_level , page_count ,
record_count , avg_page_space_used_in_percent ,
        avg_fragmentation_in_percent
from sys . dm_db_index_physical_stats
( db_id ( N'test1 ' ), OBJECT_ID ( N'dbo.test_index ' ), Null ,
 Null , ' Detailed ' )
 alter index idx_ncl_pole1 on test_index rebuild
 alter index idx_cl_pid on test_index rebuild
```

4. СТВОРЕННЯ ІНДЕКСІВ ДО ВЛАСНОЇ БД

1. Вивести перелік вже створених у БД ІНДЕКСІВ

```
SELECT sysobjects . name AS Таблиця , sysindexes . name AS Індекс , sysindexes . indid
AS Номер
FROM sysobjects INNER JOIN
 sysindexes ON sysobjects . id = sysindexes . id
WHERE ( sysobjects . xtype = 'U' ) AND ( sysindexes . indid > 0 )
ORDER BY sysobjects . name , sysindexes . indid
```

2. Створіть некластеризований індекс для столиця date_rozm таблиці zakaz , із заповненням простору кожної сторінки листя індексу на 70%.

```
create nonclustered index ix_date_rozm on zakaz ( date_rozm )
with  fillfactor = 70
```

```
select OBJECT_NAME ( object_id ) as table_name ,
name as index_name , type , type_desc
from sys.indexes
where OBJECT_ID = OBJECT_ID ( N'zakaz ')
```

3. Створити однозначний складовий індекс для столбців Nazva і City таблиці клієнт.

```
drop index ix_klient_nazva_city on klient
create nonclustered index ix_klient_nazva_city on klient ( city , nazva )
```

4. Чи буде якась різниця, якщо змінити порядок стовпців зразків у складеному індексі? (перевірити експеременталько)

```
select index_type_desc , page_count ,
record_count , avg_page_space_used_in_percent , avg_fragment_size_in_pages
from sys.dm_db_index_physical_stats
( db_id ( N'torg_firm '), OBJECT_ID ( N'klient '), Null ,
 Null , ' Detailed ')
```

5. Усунути індекс, який був створений для первинного ключа таблиці sotrudnik ?

```
alter table zakaz drop constraint FK __ zakaz __id_ sotrud __ 2D27B809
alter table sotrudnik drop constraint PK __ sotrudni __ 668829F13DE3C28F
```

6. Створити індекси які підвищуватимуть продуктивність запитів. Перевірити за планом вионання запитів

```
SELECT id_tovar , Nazva , price FROM zboží WHERE Nazva = ' Moloko '
SELECT id_tovar , Nazva , price FROM zboží WHERE Nazma = 'Молоко' AND Price = 10
SELECT nazva FROM zakaz_tovar _ _ WHERE tovar.id_tovar = tovar.id_tovar
SELECT nazva FROM zakaz_tovar _ _ WHERE tovar.id_tovar = tovar.id_tovar
AND Price >10
```

Звіт містить:

1. Лістингу коду завдання 1,2.
2. Лістинг коду завдання 3 із коментарями-порівнянням параметрів некластерного індексу із купою та кластеризованим.
3. Лістінг коду завдання 4 із обґрунтуванням вибору стовпців обраних для індексації.
4. Звіт у форматі pdf.