Exadata Üzerinde Veri Sıkıştırma Yöntemleri

İçindekiler

1. Advanced Compression ve Hybrid Columnar Compression

Veri sıkıştırma, günümüz teknolojisinin olmazsa olmazıdır. Oracle veritabanında da yıllardır bu özelliği kullanıyoruz. Ancak 11g'ye kadar sıkıştırma yaparken, düşünmemiz gereken çeşitli kıstaslarımız vardı. Toplu kayıt girişleri (bulk insert) yapıldığında, bu özelliği append hint'i vererek efektif biçimde kullanabiliyorduk. Tek tek, satır satır gelen insert'ler içinse maalesef veriyi sıkıştırma şansımız yoktu.

Oracle 11g ile birlikte Advanced Compression ile tanıştık. Advanced compression ile birlikte, teker teker gelen insert'ler sorun olmaktan çıktı. Artık onları da sıkıştırabiliyorduk. Ancak sıkıştırma teknolojisindeki bütün ilerleme bununla sınırlı değildi. Exadata ile birlikte Hybrid Columnar Compression özelliğiyle tanıştık. Bu özellik sayesinde, veriyi artık sütun bazlı sıkıştırabiliyoruz.

2. Hybrid Columnar Compression'i özel kılan nedir?

Oracle'da veri sıkıştırma block içinde yapılır. Block içindeki satırlar incelenir ve benzer '*pattern*'lara sahip değerler, pointer'larla ifade edilir. Çok basit bir örnek üzerinden gidelim ve AD/SOYAD sütunlarından oluşan bir tablomuzun olduğunu düşünelim. *AHMET ÇELEBİ*, *MEHMET ÇELEBİ* yazmak yerine *A[*]* ve *ME[*]* şeklinde ifade etsek ve "*" işaretçisinin "*HMET ÇELEBİ*" olduğunu farz etsek, burada bir yer kazanacağımız muhakkak. İşaretlerle ifadenin block içinde defalarca kere tekrar edileceğini düşünürsek, yer kazancı daha da artacaktır. Fakat daha efektif bir yöntem de kullanılabilir.

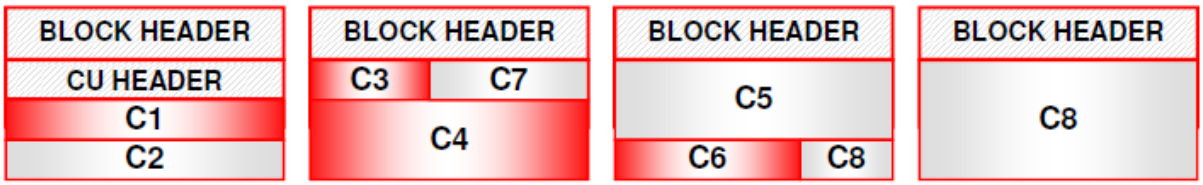
Sütun bazlı tekrarın olması, satır bazlı tekrara göre çok daha yüksek bir olasılığa sahiptir. Bu yüzden satır bazında benzer pattern'ları kullanmak yerine, sütun bazında bir sıkıştırma yapılması, sıkıştırma oranını daha yüksek rasyolara çıkartır. HCC'yi özel kılan da, sıkıştırma tekniğinden dolayı kazandığımız yüksek kazanım oranıdır.

3. Dikkat edilmesi gerekenler

Devam etmeden önce ufak bazı bilgilendirmeler yapmam gerekiyor. Advanced Compression kullanmak için bu lisansa sahip olmak gerekmektedir. HCC içinse ek bir lisans gerekmez ancak kullanılacak sistemin HCC'yi desteklemesi istenir. Sütuın bazlı sıkıştırmayı kullanabilmek için Exadata, ZFS Storage Appliance (ZFSSA) veya

Pillar Axiom gibi bir donanıma sahip olmalısınız. Standart bir disk sistemleri üzerinde maalesef bu özelliği kullanamıyoruz. Bu konunun bir yönü daha var: Asıl ortamınız Exadata olurken; standby ortamınız non-exadata bir makine olabilir. Exadata ortamında HCC tablolarınız varsa, bu tablolar exadata olmayan standby ortama yansır. Fakat sütun bazlı sıkıştırılmış tabloları okumak için standby ortamda veriyi önce "uncompress" etmeniz gerekecektir. Yalnız tekrar edeyim; standby ortamınız Exadata, ZFSSA veya Pillar Axiom kullanıyorsa, uncompress etmeden ve hiçbir ek iş yapmadan tablolarınızı okuyabilirsiniz.

Bir diğer önemli konuysa, bu özelliğin sık güncellenen tablolarda kullanılmaması gerekliliği... Sıkıştırılması istenen tablonun sütunları, Compression Unit adı verilen aşağıdakine benzer mantıksal yapılarda tutulur.



Bir silme veya güncelleme işlemi yapıldığında, sıkıştırma ünitesini tamamen kilitlersiniz. Yani bir satır silmenin bedeli, sıkıştırma performansına göre 10bin satırı tutmak anlamına gelebilir. OLTP ortamda, yoğun kullanımlı bir tabloda böyle bir yaklaşım pek uygun olmaz. Veri ambarı projelerindeyse, bu durum zaten sorun olmaz.

4. Sıkıştırma Yöntemleri

Toplam 6 çeşit sıkıştırma yöntemimiz bulunuyor:

1. **Basic:** Çok uzun zamandır kullandığımız standart sıkıştırma yöntemi.
2. **OLTP:** Oracle 11 ile birlikte çıkan ve OLTP ortamlarında kullanılabilen sıkıştırma yöntemi.
3. **QUERY LOW (Exadata):** Sıkıştırmanın getirdiği yüke dikkat edilmesi gerekli ortamlarda kullanılan sıkıştırma yöntemi.
4. **QUERY HIGH (Exadata):** Disk kazanımının önemli olduğu durumlarda tercih ettiğimiz sıkıştırma yöntemi.

5. **ARCHIVE LOW (Exadata):** Arşiv verisinin nispeten az iş yükü yaratarak, sıkıştırılması istendiği durumlarda kullandığımız sıkıştırma yöntemi.
6. **ARCHIVE HIGH (Exadata):** Arşiv verisinin olabildiğince yüksek oranda sıkıştırılması amacıyla kullanılan sıkıştırma yöntemi. (Disk kazanımı maksimum olmakla birlikte, sıkıştırma işinin en yoğun yük yarattığı yöntemdir.)

5.Test Tablosunun Hazırlanması

Aşağıdaki gibi bir test tablosu yaratıp, içini veriyle dolduralım. Testlerimiz açısından tablonun bir milyon satırın üstünde olması gerektiğinden insert ifadelerini birkaç kere çalıştırıyoruz.

```
CREATE TABLE SYSTEM.DENEME_0_NOCOMPRESSION
NOCOMPRESS NOLOGGING AS SELECT * FROM DBA_OBJECTS;
INSERT INTO SYSTEM.DENEME_0_NOCOMPRESSION
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- ...
INSERT INTO SYSTEM.DENEME_0_NOCOMPRESSION
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
commit;
```

Tablo boyutu ve satır sayısı, ortamdan ortama farklılık gösterecektir. Benim yaptığım çalışmadaysa, 2.976.048 satıra sahip, 272MB boyutunda bir tablo elde ettim.

6.Sıkıştırma Öncesi Tahmin

Tabloyu gerçekten sıkıştırmadan, sıkıştırmanın bize neler sağlayacağını görebiliriz. Bunun için DBMS_COMPRESSION paketini kullanıyoruz. Tabloyla ilgili bilgileri verdikten sonra, hangi sıkıştırma yöntemini kullanmak istediğimizi belirtiyoruz. Sonrasında bu paket aracılığıyla hesaplamalar yapılıyor ve ne kadar kazanım elde edeceğimizi buluyoruz. Esasında paket tarafından yapılan işlem örnekleme yaparak, temp tablo oluşturmak gibi duruyor. Temp tablo, işlem sonunda da drop ediliyor. Bütün süreç kendiliğinden ve dinamik bir şekilde yürüyor.

Aşağıdaki örnekte, SYSTEM tablespace'inde bulunan DENEME_0_COMPRESSION tablosunun Query High biçiminde sıkıştırılmasının neler kazandıracağına bakıyoruz.

```
set serveroutput on
declare
   v_scratchtbsname      varchar2(32) := 'SYSTEM';
   v_ownname             varchar2(32) := 'SYSTEM';
   v_tabname             varchar2(32) := 'DENEME_0_NOCOMPRESSION';
   v_partname            varchar2(32) := null;
   v_comptype            number(10,0) := DBMS_COMPRESSION.COMP_FOR_QUERY_HIGH;
   v_blkcnt_cmp          pls_integer;
   v_blkcnt_uncmp        pls_integer;
   v_row_cmp             pls_integer;
   v_row_uncmp           pls_integer;
   v_cmp_ratio           number;
   v_comptype_str        varchar2(4000);

begin

   DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
      Scratchtbsname     => v_scratchtbsname,
      ownname            => v_ownname,
      tabname            => v_tabname,
      partname           => v_partname,
      comptype           => v_comptype,
      blkcnt_cmp         => v_blkcnt_cmp,
      blkcnt_uncmp       => v_blkcnt_uncmp,
      row_cmp            => v_row_cmp,
      row_uncmp          => v_row_uncmp,
      cmp_ratio          => v_cmp_ratio,
      comptype_str       => v_comptype_str
   );

   dbms_output.put_line('Sıkıştırılan block adedi :' || v_blkcnt_cmp);
   dbms_output.put_line('Sıkıştırılmayan block adedi : ' || v_blkcnt_uncmp);
   dbms_output.put_line('Sıkıştırılan block başına satır adedi : ' || v_row_cmp);
   dbms_output.put_line('Sıkıştırılmamış block başına satır adedi : ' || v_row_uncmp);
   dbms_output.put_line('Sıkıştırma oranı :' || v_cmp_ratio);
   dbms_output.put_line('Kazanım (%) :' || round((v_blkcnt_uncmp-v_blkcnt_cmp)/v_blkcnt_uncmp*100,2));
   dbms_output.put_line('Yöntem : ' || v_comptype_str);

end;
/
```

```
Compression Advisor self-check validation successful. select count(*) on
both Uncompressed and EHCC Compressed format = 1000001 rows
Sıkıştırılan block adedi :672
Sıkıştırılmayan block adedi : 12769
Sıkıştırılan block başına satır adedi : 1488
Sıkıştırılmamış block başına satır adedi : 78
Sıkıştırma oranı :19
Kazanım (%) :94.74
Yöntem : "Compress For Query High"
PL/SQL procedure successfully completed.
```

Tahmine göre %94.74'lük bir kazanımımız var. Yani sıkıştırma sonrasında tablo boyutunun 14MB'a düşmesini bekliyoruz. (272MB x (100-94.74) / 100 = 14.3 MB) Diğer sıkıştırma yöntemlerine göre tablonun ne kadara düşeceğini de çıkarttım:

1. Compress For OLTP : 94MB
2. Compress For Query Low : 39MB
3. Compress For Query High : 14MB
4. Compress For Archive Low : 14MB
5. Compress For Archive High : 14MB

Peki, BASIC compression için tahminde bulunma şansımız var mı? Biraz daha zahmetli olmak kaydıyla, DMBS_COMPRESSION paketine benzer bir yöntem izleyebiliriz. Önce toplam satır sayısını çıkartırız.

```
SELECT COUNT(*) FROM SYSTEM.DENEME_0_NOCOMPRESSION;
  COUNT(*)
----------
   2976048
```

Sıkıştırma tahmininde kullanacağımız satır sayımız, 1.000.000 satır olacak şekilde kabul etmeliyiz. Bir milyon satır, toplam satır sayısının 1/3'üne denk geliyor. Örnekleme yaparak, tablonun üçte birinden (%33) bir tablo yaratıyoruz.

```
CREATE TABLE SYSTEM.DENEME_1_BASIC_INTERIM
NOLOGGING COMPRESS as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION SAMPLE( 33 ); -- << 1000000/2976048
```

INTERIM olarak yaratılan bu tablonun boyutu 34MB oluyor. Üçte birinin 34MB olduğunu düşünürsek, bütün tablonun 102MB olacağını tahmin edebiliriz. (Bu arada öbelirtmek isterim ki, tablo basic compression ile tamamen sıkıştırıldığında 96MB oluyor. %33 üzerinden giderek başarılı bir tahmin elde ettik.)

7.Sıkıştırma Testi

Sıkıştırma tahminlerimizi yaptıktan sonra, sıkıştırma testlerine başlayabiliriz. Çalışmamızda, var olan tabloyu bozmadan, farklı sıkıştırma teknikleriyle kopyalarını üreteceğiz.

Her SQL ifadesinin altında yazan "***Elapsed***" değeri, o işlemin gerçekte ne kadar sürdüğünü gösteriyor.

```
SET TIMING ON

CREATE TABLE SYSTEM.DENEME_1_BASIC_COMPRESSION
NOLOGGING COMPRESS as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- Elapsed: 00:00:06.79

CREATE TABLE SYSTEM.DENEME_2_OLTP_COMPRESSION
NOLOGGING COMPRESS FOR OLTP as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- Elapsed: 00:00:06.67

CREATE TABLE SYSTEM.DENEME_3_QUERYLOW_COMPRESSION
NOLOGGING COMPRESS FOR QUERY LOW
as SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- Elapsed: 00:00:03.95

CREATE TABLE SYSTEM.DENEME_4_QUERYHIGH_COMPRESSION
NOLOGGING COMPRESS FOR QUERY HIGH as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- Elapsed: 00:00:10.28

CREATE TABLE SYSTEM.DENEME_5_ARCHLOW_COMPRESSION
NOLOGGING COMPRESS FOR ARCHIVE LOW as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- Elapsed: 00:00:10.81

CREATE TABLE SYSTEM.DENEME_6_ARCHHIGH_COMPRESSION
NOLOGGING COMPRESS FOR ARCHIVE HIGH as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION;
-- Elapsed: 00:00:55.30
```

Kullanılan yönteme göre boyut ve sürelerin değerlerini, aşağıdaki tabloda görebilirsiniz. Ayrıca daha rahat mukayese için tahmini sıkıştırma değerlerini de tabloya ekledim. Farkedeceğiniz gibi tahmin ve gerçek sıkıştırma sonuçları arasında yüksek bir fark olmadığı görülüyor.

SEGMENT_NAME	ESTIMATED_SIZE_MB	REAL_SIZE_MB	ELAPSED_TIME
DENEME_0_NOCOMPRESSION	272	272	00:00:00.00
DENEME_1_BASIC_COMPRESSION	102	96	00:00:06.79
DENEME_2_OLTP_COMPRESSION	94	104	00:00:06.67
DENEME_3_QUERYLOW_COMPRESSION	39	43	00:00:03.95
DENEME_4_QUERYHIGH_COMPRESSION	14	16	00:00:10.28
DENEME_5_ARCHLOW_COMPRESSION	14	16	00:00:10.81
DENEME_6_ARCHHIGH_COMPRESSION	14	15	00:00:55.30

Sıkıştırma işleminin boyut kazandırmak dışında performansı arttırmaya yönelik artısı da vardır. Artık daha az block okunacağı için, I/O miktarı azalır ve bu da performansa yansır. Aşağıda aynı veriyi sorgulayan SQL ifadelerinin, Execution plan'ları bulunmaktadır. Sıkıştırma miktarı arttıkça, cost'un giderek düştüğü görülür.

```
SELECT COUNT(*) FROM SYSTEM.DENEME_0_NOCOMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    18 |  9292   (1)| 00:01:52 |
|   1 |  SORT AGGREGATE            |                         |     1 |    18 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME_0_NOCOMPRESSION  |     5 |    90 |  9292   (1)| 00:01:52 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT COUNT(*) FROM SYSTEM.DENEME_1_BASIC_COMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    79 |  3244   (1)| 00:00:39 |
|   1 |  SORT AGGREGATE            |                         |     1 |    79 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME_1_BASIC_COMPRESSION |   130 | 10270 |  3244   (1)| 00:00:39 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT COUNT(*) FROM SYSTEM.DENEME_2_OLTP_COMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    79 |  3608   (1)| 00:00:44 |
|   1 |  SORT AGGREGATE            |                         |     1 |    79 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME 2 OLTP COMPRESSION |   145 | 11455 |  3608   (1)| 00:00:44 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT COUNT(*) FROM SYSTEM.DENEME_3_QUERYLOW_COMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    79 |  1510   (2)| 00:00:19 |
|   1 |  SORT AGGREGATE            |                         |     1 |    79 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME_3_QUERYLOW_COMPRESSION |    60 |  4740 |  1510   (2)| 00:00:19 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT COUNT(*) FROM SYSTEM.DENEME_4_QUERYHIGH_COMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    79 |   565   (5)| 00:00:07 |
|   1 |  SORT AGGREGATE            |                         |     1 |    79 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME_4_QUERYHIGH_COMPRESSION |    63 |  4977 |   565   (5)| 00:00:07 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT COUNT(*) FROM SYSTEM.DENEME_5_ARCHLOW_COMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    79 |   576   (7)| 00:00:07 |
|   1 |  SORT AGGREGATE            |                         |     1 |    79 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME_5_ARCHLOW_COMPRESSION |    22 |  1738 |   576   (7)| 00:00:07 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT COUNT(*) FROM SYSTEM.DENEME_6_ARCHHIGH_COMPRESSION
WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%';
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                         |     1 |    79 |   553   (8)| 00:00:07 |
|   1 |  SORT AGGREGATE            |                         |     1 |    79 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| DENEME_6_ARCHHIGH_COMPRESSION |    30 |  2370 |   553   (8)| 00:00:07 |
-----------------------------------------------------------------------------------------------------
```

Şimdi de ifadeleri gerçekten çalıştıralım:

```
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_0_NOCOMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:00.17
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_1_BASIC_COMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:01.09
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_2_OLTP_COMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:01.06
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_3_QUERYLOW_COMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:01.50
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_4_QUERYHIGH_COMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:00.36
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_5_ARCHLOW_COMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:00.40
SQL>  SELECT COUNT(*) FROM SYSTEM.DENEME_6_ARCHHIGH_COMPRESSION
      WHERE OBJECT_ID < 17 AND OBJECT_NAME LIKE 'S%'
      -- Elapsed: 00:00:01.00
```

İfadelerin çalışma sürelerine bakarsak, sıkıştırma arttıkça çalışma sürelerinde paralel bir düşüş olmamıştır. Aynı testi **cell_offload_processing**'i kapatıp, tekrar edersek, sıkıştırma seviyesi arttıkça, ifadelerin çalışma süreleri düşecektir.

8.Daha efektif sıkıştırma

Sütun bazlı sıkıştırmayı daha verimli kullanmak için, sütunları sıralayabiliriz.

```
SET TIMING ON

CREATE TABLE SYSTEM.DENEME_1_BASIC_COMPRESS2
NOLOGGING COMPRESS as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION
order by 1,2,3,4,5,6,7,8,9,10,11;
-- Elapsed: 00:00:07.67

CREATE TABLE SYSTEM.DENEME_2_OLTP_COMPRESS2
NOLOGGING COMPRESS FOR OLTP as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION
order by 1,2,3,4,5,6,7,8,9,10,11;
-- Elapsed: 00:00:07.75

CREATE TABLE SYSTEM.DENEME_3_QUERYLOW_COMPRESS2
NOLOGGING COMPRESS FOR QUERY LOW as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION
order by 1,2,3,4,5,6,7,8,9,10,11;
-- Elapsed: 00:00:07.23
```

```
CREATE TABLE SYSTEM.DENEME_4_QUERYHIGH_COMPRESS2
NOLOGGING COMPRESS FOR QUERY HIGH as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION
order by 1,2,3,4,5,6,7,8,9,10,11;
-- Elapsed: 00:00:09.75

CREATE TABLE SYSTEM.DENEME_5_ARCHLOW_COMPRESS2
NOLOGGING COMPRESS FOR ARCHIVE LOW as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION
order by 1,2,3,4,5,6,7,8,9,10,11;
-- Elapsed: 00:00:09.10

CREATE TABLE SYSTEM.DENEME_6_ARCHHIGH_COMPRESS2
NOLOGGING COMPRESS FOR ARCHIVE HIGH as
SELECT * FROM SYSTEM.DENEME_0_NOCOMPRESSION
order by 1,2,3,4,5,6,7,8,9,10,11;
-- Elapsed: 00:00:26.65
```

Yarattığımız tabloların boyutlarını kontrol edelim:

```
select segment_name,bytes/1024/1024 as mb from user_segments
where segment_name like 'DENEME%COMPRESS2' OR segment_name in
('DENEME_0_NOCOMPRESSION')
order by 1;
```

SEGMENT_NAME	MB
DENEME_0_NOCOMPRESSION	272
DENEME_1_BASIC_COMPRESS2	41
DENEME_2_OLTP_COMPRESS2	45
DENEME_3_QUERYLOW_COMPRESS2	4
DENEME_4_QUERYHIGH_COMPRESS2	3
DENEME_5_ARCHLOW_COMPRESS2	3
DENEME_6_ARCHHIGH_COMPRESS2	2

Sorgu sonucuna göre, 272MB'lık tablo, sıkıştırma sonunda sadece **2MB**'a düşüyor. Görüldüğü gibi veriyi sıralama imkanı varsa, bunun getireceği kazanım hâyli yüksek oluyor.