

SQL projektfeladat dokumentáció

Bookings tábla elkészítése¹

Kezdeti adatok importálása

Az európai városoknevek és az országok ISO kód listái külső forrásból² lettek importálva két segéd táblába.

Countries tábla:

```
CREATE TABLE Countries(  
    [asciiname] VARCHAR(255) NOT NULL PRIMARY KEY,  
    [country] VARCHAR(255) NOT NULL,  
    [population] int NOT NULL  
)  
BULK INSERT Countries  
FROM '<Elérési Útvonal>\countries.csv'  
WITH (FORMAT='CSV',  
    FIRSTROW=2,  
    FIELDTERMINATOR = '|',  
    ROWTERMINATOR = '0x0a')
```

Iso2Codes tábla:

```
CREATE TABLE Iso2Codes(  
    [Code] VARCHAR(2) NOT NULL,  
    [Name] VARCHAR(255) NOT NULL PRIMARY KEY  
)  
BULK INSERT Iso2Codes  
FROM '<Elérési Útvonal>\iso2.csv'  
WITH (FORMAT='CSV',  
    FIRSTROW=2,  
    FIELDTERMINATOR = '|',  
    ROWTERMINATOR = '0x0a')
```

Kiegészítő adathalmazok létrehozása

A kiegészítő adatok ideiglenes táblában vannak tárolva a felhasználásukig, az átláthatóság kedvéért.

Az ISO kódok hozzárendelése csak az európai országokhoz:

```
SELECT c.asciiname, ic.Code  
INTO #euCountriesIsos  
FROM Iso2Codes ic  
INNER JOIN Countries c on ic.Name = c.country
```

Városok kiválasztása:

```
CREATE TABLE #selectedCities (  
    ID int NOT NULL IDENTITY(1,1) PRIMARY KEY,  
    asciiname VARCHAR(255) NOT NULL UNIQUE,  
    Code CHAR(2) NOT NULL  
)  
  
INSERT INTO #selectedCities  
SELECT tmp.asciiname, tmp.Code  
FROM (  
    SELECT DISTINCT TOP 15 Code, asciiname  
    FROM #euCountriesIsos eci  
    WHERE (ABS(CAST((CHECKSUM(*) * RAND()) as int)) % 100) < 30) tmp  
UNION  
(SELECT 'Luton' asciiname, 'GB' Code)
```

¹A teljes scriptek a mellékelt SSMS solution-ben találhatóak

²ISO Kódok, Városok

A **TABLESAMPLE** utasítás pontatlansága miatt a **WHERE (ABS(CAST((CHECKSUM() * RAND()) as int)) % 100) < 30*** segítségével veszünk véletlenszerű mintát a **#euCountriesIsos** táblából.

Ezek után generálunk 2500, hatjegyű **CustomerID**-t:

```
WITH
  10(i) AS (SELECT 0 UNION ALL SELECT 0),
  11(i) AS (SELECT 0 FROM 10 a, 10 b),
  12(i) AS (SELECT 0 FROM 11 a, 11 b),
  13(i) AS (SELECT 0 FROM 12 a, 12 b),
  14(i) AS (SELECT 0 FROM 13 a, 13 b),
  numbers(i) AS (SELECT ROW_NUMBER() OVER(ORDER BY (SELECT 0)) FROM 14)
SELECT DISTINCT TOP(2500)
  (CEILING(RAND(CHECKSUM(NEWID()))*899999)+100000) CustomerID
INTO #cids
FROM numbers
ORDER BY CustomerID
```

CustomerID-khoz **CCountry** azonosítókat rendelünk:

```
SELECT cp.CustomerID CustomerID, sc.Code CCountry
INTO #cidPairs
FROM (
  SELECT cids.CustomerID,
    (CEILING(RAND(CHECKSUM(NEWID()))*(SELECT COUNT(*)
      FROM #selectedCities))) CountryID
  FROM #cids cids
) cp
INNER JOIN #selectedCities sc on cp.CountryID = sc.ID
ORDER BY CustomerID
```

Bookings feltöltése

Bookings tábla létrehozása:

```
CREATE TABLE Bookings(
[BookingID] int NOT NULL IDENTITY(1,1) PRIMARY KEY,
[CustomerID] int NOT NULL,
[CCountry] varchar(2) NOT NULL,
[DepartureStation] varchar(30) NOT NULL,
[Date] datetime NOT NULL DEFAULT (GETDATE()),
[Price] money NOT NULL,
[Seats] int NOT NULL
)
```

Egy kezdeti adathalmaz létrehozása:

```
WITH
  10(i) AS (SELECT 0 UNION ALL SELECT 0),
  11(i) AS (SELECT 0 FROM 10 a, 10 b),
  12(i) AS (SELECT 0 FROM 11 a, 11 b),
  13(i) AS (SELECT 0 FROM 12 a, 12 b),
  numbers(i) AS (SELECT ROW_NUMBER() OVER(ORDER BY (SELECT 0)) FROM 13)
SELECT TOP(1000000)
  cp.CustomerID,
  cp.CCountry,
  sc.asciiname DepartureStation,
  (DATEADD(ms,RAND(CHECKSUM(NEWID())) * 86400000,
    DATEADD(d,RAND(CHECKSUM(NEWID()))*365,'2019-01-01')))
  ) [Date],
  (ROUND(RAND(CHECKSUM(NEWID()))*200+10,2)) Price,
  (CEILING(RAND(CHECKSUM(NEWID()))*50)) Seats
INTO #bookingsAll
from numbers
```

```

CROSS APPLY (
    SELECT
        cp.CustomerID,
        cp.CCountry,
        (
            SELECT CEILING(RAND(CHECKSUM(NEWID())) * (SELECT COUNT(*)
            FROM #selectedCities))
        ) DepartureStationID
    FROM #cidPairs AS cp
) AS cp
INNER JOIN #selectedCities sc
    ON sc.ID = cp.DepartureStationID

```

Az adathalmazból véletlenszerűen kiválasztásra kerül tízezer bejegyzés a végleges *Bookings* táblába:

```

INSERT INTO Bookings
SELECT TOP(10000) *
FROM #bookingsAll
WHERE (ABS(CAST((CHECKSUM(*) * RAND()) as int)) % 100) < 30

```

Indexek létrehozása

BookingID mezőn alapértelmezetten létrejön egy clustered index a *PRIMARY KEY* feltétel miatt

Non Clustered Index a CCountry és CustomerID párokon:

```

CREATE NONCLUSTERED INDEX NC_Bookings_countryCid ON [dbo].[Bookings]
(
    [CCountry] ASC,
    [CustomerID] ASC
)

```

Ezen kívül a feladat lekérdezéseihez a Database Engine Tuning Advisor által ajánlott optimális non clustered indexek:

```

CREATE NONCLUSTERED INDEX NC_Bookings_countryCidDate
ON [dbo].[Bookings]
(
    [Date] ASC,
    [CCountry] ASC,
    [CustomerID] ASC
)
INCLUDE([Price])

CREATE NONCLUSTERED INDEX NC_Bookings_countryDateStation
ON [dbo].[Bookings]
(
    [DepartureStation] ASC,
    [Date] ASC,
    [CCountry] ASC
)
INCLUDE([Price])

```

Lekérdezések

1. Lekérdezés - 2019. májusi megrendelések csökkenő sorrendben

```
SELECT b.CustomerID, b.CCountry , SUM(b.Price) 'Total Amount'
FROM Bookings b
WHERE b.[Date] >= '2019-05-01' AND b.[Date] < '2019-06-01'
GROUP BY b.CustomerID,b.CCountry
ORDER BY 'Total Amount' DESC
```

2. Lekérdezés - Második negyedév Lutonból induló megrendeléseinek összegei

```
SELECT b.CCountry Country,
       ISNULL(
         SUM(case when DATEPART(m,b.[Date]) = 4 THEN b.Price END)
         ,0) '2019-04',
       ISNULL(
         SUM(case when DATEPART(m,b.[Date]) = 5 THEN b.Price END)
         ,0) '2019-05',
       ISNULL(
         SUM(case when DATEPART(m,b.[Date]) = 6 THEN b.Price END)
         ,0) '2019-06'
FROM Bookings b
WHERE b.DepartureStation = 'Luton'
      AND b.[Date] >= '2019-04-01'
      AND b.[Date] < '2019-07-01'
GROUP BY b.CCountry
ORDER BY SUM(b.Price) DESC
```

Megvalósítható *PIVOT* használatával is, ez azonban csak kevésbé rugalmas megjelenítést tesz lehetővé

```
SELECT * FROM
(SELECT b.CCountry Country, FORMAT(b.[Date],'yyyy-MM') dpart, b.Price
 FROM Bookings b
 WHERE b.DepartureStation = 'Luton'
       AND b.[Date] >= '2019-04-01'
       AND b.[Date] < '2019-07-01') x
PIVOT (
SUM(x.Price)
FOR dpart IN ([2019-04],[2019-05],[2019-06])
) PivotTable;
```

3. Lekérdezés - Ügyfelek vásárlási számának kimutatása régióként

```
SELECT cntQry.CCountry Country,
       COUNT(
         case when cntQry.buyCnt = 1 THEN 1 ELSE null END
       ) [1],
       COUNT(
         case when cntQry.buyCnt = 2 THEN 1 ELSE null END
       ) [2],
       COUNT(
         case when cntQry.buyCnt = 3 THEN 1 ELSE null END
       ) [3],
       COUNT(
         case when cntQry.buyCnt BETWEEN 4 AND 9 THEN 1 ELSE null END
       ) [4-10],
       COUNT(
         case when cntQry.buyCnt BETWEEN 10 AND 100 THEN 1 ELSE null END
       ) [10-100]
FROM (
```

```
SELECT COUNT(b.BookingID) buyCnt, b.CustomerID, b.CCountry
FROM Bookings b
GROUP BY b.CustomerID,b.CCountry
) cntQry
GROUP BY cntQry.CCountry
ORDER BY SUM(cntQry.buyCnt)
```

6. feladat - Helymegtakarítás az indexek eldobásával

Az indexek méretét több dolog is befolyásolja, így a tényleges tábla és az indexek ismerete nélkül nem lehet meghatározni a helymegtakarítást. Befolyásoló szempontok lehetnek például a tábla mérete, a pagek fill-factora vagy indexben használt oszlopok típusai és száma. Továbbá nehezen meghatározható tényező a tömörített indexek és LOB-ok tárhely igénye.

Az indexek által foglalt hely lekérdezhető az alábbi queryvel:

```
SELECT
OBJECT_NAME(i.OBJECT_ID) AS TableName,
i.name AS IndexName,
i.index_id AS IndexID,
8 * SUM(a.used_pages) AS 'Indexsize(KB)'
FROM sys.indexes AS i
JOIN sys.partitions AS p
    ON p.OBJECT_ID = i.OBJECT_ID
    AND p.index_id = i.index_id
JOIN sys.allocation_units AS a
    ON a.container_id = p.partition_id
WHERE OBJECT_NAME(i.OBJECT_ID) = <TÁBLA>
GROUP BY i.OBJECT_ID,i.index_id,i.name
ORDER BY OBJECT_NAME(i.OBJECT_ID),i.index_id
```

7. feladat - Helyfelszabadítás az adatbázis adat fájl törlésével

Ahhoz, hogy egy nem üres adatfájlt eltávolíthassunk, először a benne tárolt adatokat a többi fájba kell migrálnunk (*DBCC SHRINKFILE, EMPTYFILE* argumentuma). Majd a fájl törölhető.

Shrinkelésért és törlésért felelős script:

```
DBCC SHRINKFILE ('<FAJL>', EMPTYFILE);
```

```
ALTER DATABASE SQL_Projekt_Feladat REMOVE FILE <FAJL>
```


8. feladat - tempdb elosztása

Új adat fájlok létrehozása (ismételve a kívánt alkalommal):

```
ALTER DATABASE [tempdb]
ADD FILE (
    NAME = N'temp<NÉV>',
    FILENAME = N'T:\<Elérési Útvonal>\temp<NÉV>.ndf' ,
    SIZE = 10GB ,
    FILEGROWTH = 65536KB
)
```

Régi tempdb file tartalmának elosztása a többi fájl között:

```
DBCC SHRINKFILE (N'tempdev' , EMPTYFILE)
```

Régi tempdb file maximális méretének beállítása:

```
DBCC SHRINKFILE (N'tempdev' , 10240)
```

9. feladat - SQL Server snapshot isolation

A megadott beállítások esetén előfordulhat, hogy update tranzakciók hibát generálnak (optimistic concurrency miatt) Select utasítások esetén alkalmazott-e a fejlesztő update lockot, vagy használt-e *try-catch* vagy *if error* hibakezelést.

10. feladat - Adatbázis visszaállítás

A.

Tail log backup készítése:

```
BACKUP LOG SQL_Projekt_Feladat
TO DISK = N'<Elérési Út>\tail_log.bak'
WITH
    NORECOVERY,
    NO_TRUNCATE
```

Utolsó full backup visszaállítása

```
RESTORE DATABASE SQL_Projekt_Feladat
FROM DISK = N'<Elérési Út>\last_full.bak'
WITH NORECOVERY;
```

Utolsó differenciális backup visszaállítása

```
RESTORE DATABASE SQL_Projekt_Feladat
FROM DISK = N'<Elérési Út>\last_diff.bak'
WITH NORECOVERY;
```

A megfelelő transaction logok visszaállítása:

```
RESTORE LOG SQL_Projekt_Feladat
FROM DISK = N'<Elérési Út>\log<NÉV>.bak'
WITH NORECOVERY;
```

Tail log visszaállítása:

```
RESTORE LOG SQL_Projekt_Feladat
FROM DISK = N'<Elérési Út>\tail_log.bak'
WITH NORECOVERY;
```

Adatbázis kivétele recovery módból:

```
RESTORE DATABASE SQL_Projekt_Feladat
WITH RECOVERY;
```

B.

Visszaállítjuk a megfelelő mentést egy ideiglenes adatbázisba:

```
RESTORE DATABASE DatabaseCopy
FROM DISK=N'<Elérési Út>\database.bak'
WITH NORECOVERY,
MOVE 'DataBaseFile1'
TO '<Elérési Út>\databaseCopy.mdf',
MOVE 'DataBaseLog'
TO '<Elérési Út>\databaseCopy_Log.ldf';
```

Ha ez megtörtént, a szükséges táblákat a *MERGE* utasítás segítségével visszaállítjuk

```
MERGE Database.target_table target
USING DatabaseCopy.source_table source
ON source.primary_key = target.primary_key
WHEN MATCHED
    THEN UPDATE SET
        target.col1 = source.col1,
        target.col2 = source.col2
WHEN NOT MATCHED
    THEN INSERT (col1,col2) VALUES (val1,val2)
WHEN NOT MATCHED BY SOURCE
    THEN DELETE;
```

11. feladat - Hibaelhárítás

Performance monitor segítségével kideríthető, ha a kiszolgálón más szolgáltatások is futnak, hogy marad-e az SQL Szerver számára megfelelő szabad erőforrás

Activity monitorral nyomon követhetőek az adatfájlon folyó I/O műveletek, közelmúltban futtatott- és jelenleg is futó erőforrásigényes lekérdezések, vagy a táblán aktív lockok (*EXEC SP_LOCK* is képes az utóbbiakat listázni)

Dynamic management viewok segítségével az Activity monitorhoz hasonló, de bővebb adatok is elérhetőek

SQL Profiler, vagy Extended events segítségével (SQL Szerver verziófüggő) task timeok megvizsgálhatóak és kideríthetőek a hosszan futó utasítások

Töredezett indexek is lassíthatnak az utasítások futásán, ez javítható az indexek újra rendezésével vagy újraépítésével (reorganize, rebuild)

Tartalomjegyzék

Bookings tábla elkészítése	2
Kezdeti adatok importálása	2
Kiegészítő adathalmazok létrehozása	2
Bookings feltöltése	3
Indexek létrehozása	4
Lekérdezések	5
1. Lekérdezés - 2019. májusi megrendelések csökkenő sorrendben	5
2. Lekérdezés - Második negyedév Lutonból induló megrendeléseinek összegei	5
3. Lekérdezés - Ügyfelek vásárlási számának kimutatása régióként	5
6. feladat - Helymegtakarítás az indexek eldobásával	7
7. feladat - Helyfelszabadítás az adatbázis adat fájl törlésével	8
8. feladat - tempdb elosztása	9
9. feladat - SQL Server snapshot isolation	10
10. feladat - Adatbázis visszaállítás	11
A.	11
B.	11
11. feladat - Hibaelhárítás	12