LAB #8
Indexes, Constraints and Partitions

Indexes

First, let's create a database and table:

```
CREATE DATABASE Base_3;
GO
USE Base_3;
GO
CREATE TABLE Table_3
(
col_2 INT,
col_3 INT,
col_4 INT
);
```

In database Base_1 create unique, clustered index 'ind_col_3' for column 'col_3' of the table Table_3 (this table must not have a primary key):

```
USE Base_1;
CREATE  UNIQUE CLUSTERED INDEX ind_col_3
            ON Table_3(col_3);
```

In database Base_1 create unique, nonclustered index 'ind_col_2' for column 'col_2' of the table Table_1:

```
USE Base_1;
CREATE  UNIQUE NONCLUSTERED INDEX ind_col_2
           ON Table_3(col_2);
```

In database Base_1 create nonclastered index 'ind_col_4' for column 'col_4' of the table Table_1:

```
USE Base_1;
CREATE  NONCLUSTERED INDEX ind_col_4
            ON Table_3 (col_4);
```

In database Base_1 rebuild the index 'ind_col_4' of the table Table_1:

```
USE Base_1;
ALTER INDEX ind_col_4 ON Table_3
         REBUILD;
```

In database Base_1 disable index 'ind_col_4' of the table Table_1:

```
USE Base_1;
ALTER INDEX ind_col_4 ON Table_3
         DISABLE;
```

In database Base_1 enable index 'ind_col_4' of the table Table_1 by rebuilding it:

```
USE Base_1;
ALTER INDEX ind_col_4 ON Table_3
        REBUILD;
```

In database Base_1 rebuild all indexes of the table Table_1 (by using ALTER statement):

```
ALTER INDEX ALL ON Table_3
REBUILD;
```

In database Base_1 rename index 'Ind_col_4' of the table Table_1, new name is 'Indcol_4':

```
USE Base_1
EXEC sp_rename 'Table_3.Ind_col_4', 'Indcol_4', 'INDEX';
```

In database Base_1 delete index 'Ind_col_4' of the table Table_1:

```
DROP INDEX Table_3.Indcol_4;
```

Display information about all indexes of the table Table_1 in database Base_1:

```
EXEC sp_helpindex 'Table_1';
```

Constraints

First, let's create a database and tables:

```
GO
CREATE TABLE Table_1 (col_1 INT, col_3 INT, col_6 INT, col_7 INT);
GO
CREATE TABLE Table_2 (col_1 INT PRIMARY KEY, table1_ID INT);
```

In database Base_1 add table_1ID new column, which have PRIMARY KEY and IDENTITY constraints to the table Table_1. Increment value is -3, seed value is 300. The name of the constraint – table1_ID_PK:

```
USE Base_3;
ALTER TABLE Table_1
ADD table1_ID INT IDENTITY(300, -3)
CONSTRAINT table1_ID_PK PRIMARY KEY;
```

In the table Table_2 of the database Base_1 add and then delete constraint FOREIGN KEY:

```
Use Base_3;
--      Add constraint FOREIGN KEY
ALTER TABLE Table_2
ADD CONSTRAINT FK_ID FOREIGN KEY (table1_ID) REFERENCES Table_1(table1_ID);
--      Delete constraint FOREIGN KEY
ALTER TABLE Table_2 DROP CONSTRAINT FK_ID;
```

In database Base_1 delete table1_ID_PK constraint from table Table_1:

```
ALTER TABLE Table_1
DROP CONSTRAINT table1_ID_PK;
```

In database Base_1 add constraint CHECK, named as 'col_check', to the column 'col_3' of the table Table_1. In accordance with this constraint the value of the column 'col_3' must exceed 1:

```
ALTER TABLE Table_1 WITH NOCHECK
ADD CONSTRAINT col_check CHECK ( col_3 > 1 );
```

In database Base_1 delete constraint CHECK named as 'col_check' from the table Table_1:

```
ALTER TABLE Table_1
DROP CONSTRAINT col_check;
```

In database Base_1 add constraint CHECK, named as 'col_7_check', to the column 'col_7' of the table Table_1. In accordance with this constraint each symbol of the column 'col_7' must be symbol-digit:

```
USE Base_3;
ALTER TABLE Table_1
ADD CONSTRAINT col_7_check
CHECK ((col_7 LIKE '[0-9][0-9][0-9][0-9][0-9]'));
```

In database Base_1 add constraint DEFAULT, named as 'col_3_default', to the column 'col_3' of the table Table_1. In accordance with this constraint the default value of the column 'col_3' is 50:

```
USE Base_3;
ALTER TABLE Table_1
ADD CONSTRAINT col_3_default
DEFAULT 50 FOR col_3;
```

In database Base_1 delete constraint DEFAULT, named as 'col_3_default', from the table Table_1:

```
USE Base_3;
ALTER TABLE Table_1
DROP CONSTRAINT col_3_default;
```

In database Base_1 add new column col_5 to the table Table_1, that has constraint UNIQUE named as 'col_5_unique':

```
USE Base_3;
ALTER TABLE Table_1 ADD col_5 VARCHAR(20) NULL
     CONSTRAINT col_5_unique UNIQUE;
```

In database Base_1 from the table Table_1 delete constraint UNIQUE, which name is 'col_5_unique':

```
ALTER TABLE Table_1
DROP CONSTRAINT col_5_unique;
```

In database Base_1 add constraint UNIQUE to the column col_6 of the table Table_1 by creating an index. The name of the index is 'ind_col6':

```
USE Base_3;
CREATE UNIQUE INDEX ind_col_6
ON Table_1(col_6);
```

PARTITIONED TABLES

A partitioned table is a special table that is divided into segments, called partitions, that make it easier to manage and query your data. By dividing a large table into smaller partitions, you can improve query performance, and you can control costs by reducing the number of bytes read by a query.

Before creating a partitioned table by using CREATE TABLE, you must first create a partition function to specify how the table becomes partitioned. A partition function is created by using CREATE PARTITION FUNCTION. Second, you must create a partition scheme to specify the filegroups that will hold the partitions indicated by the partition function. A partition scheme is created by using CREATE PARTITION SCHEME.

CREATE PARTITION FUNCTION creates a function in the current database that maps the rows of a table or index into partitions based on the values of a specified column. Using CREATE PARTITION FUNCTION is the first step in creating a partitioned table or index. A table or index can have a maximum of 15,000 partitions.

Syntax.

CREATE PARTITION FUNCTION *partition_function_name* **(** *input_parameter_type* **)**
AS RANGE [LEFT | RIGHT]
FOR VALUES ([*boundary_value* **[,...n]]);**

partition_function_name - Is the name of the partition function. Partition function names must be unique within the database and comply with the rules for identifiers.

input_parameter_type - Is the data type of the column used for partitioning. All data types are valid for use as partitioning columns, except text, ntext, image, xml, timestamp, varchar(max), nvarchar(max), varbinary(max) data types.

The actual column, known as a partitioning column, is specified in the CREATE TABLE or CREATE INDEX statement.

boundary_value - Specifies the boundary values for each partition of a partitioned table or index that uses ***partition_function_name***. If ***boundary_value*** is empty, the partition function maps the whole table or index using ***partition_function_name*** into a single partition. Only one partitioning column, specified in a CREATE TABLE or CREATE INDEX statement, can be used.

boundary_value is a constant expression that can reference variables. This includes user-defined type variables, or functions and user-defined functions. It cannot reference Transact-SQL expressions. ***boundary_value*** must either match or be implicitly convertible to the data type supplied in ***input_parameter_type***.

...n - Specifies the number of values supplied by ***boundary_value***, not to exceed 14,999. The number of partitions created is equal to n + 1. If the values are not in order, the Database Engine sorts them, creates the function, and returns a warning that the values are not provided in order. The Database Engine returns an error if n includes any duplicate values.

LEFT | RIGHT - Specifies to which side of each boundary value interval, left or right, the ***boundary_value [,...n]*** belongs, when interval values are sorted by the Database Engine in ascending order from left to right. If not specified, ***LEFT*** is the default.

CREATE PARTITION SCHEME creates a scheme in the current database that maps the partitions of a partitioned table or index to filegroups. The number and domain of the partitions of a partitioned table or index are determined in a partition function. Symtax:

CREATE PARTITION SCHEME partition_scheme_name
AS PARTITION partition_function_name
TO ({ file_group_name | [PRIMARY] } [,...n]);

partition_scheme_name. Is the name of the partition scheme. Partition scheme names must be unique within the database and comply with the rules for identifiers.

partition_function_name. Is the name of the partition function using the partition scheme. Partitions created by the partition function are mapped to the filegroups specified in the partition scheme. partition_function_name must already exist in the database. A single partition cannot contain both FILESTREAM and non-FILESTREAM filegroups.

file_group_name | [PRIMARY] [,...n]. Specifies the names of the filegroups to hold the partitions specified by ***partition_function_name***. **file_group_name** must already exist in the database.

First, let's create a database and table:

```
CREATE DATABASE Base_2
ON
(
NAME = Base_2,
FILENAME = 'D:\SQL\Base_2.mdf'
),
FILEGROUP Group1
(
NAME = Base_21,
FILENAME = 'D:\SQL\Base_21.ndf'
),
FILEGROUP Group2
(
NAME = Base_22,
FILENAME = 'D:\SQL\Base_22.ndf'
),
FILEGROUP Group3
(
NAME = Base_23,
FILENAME = 'D:\SQL\Base_23.ndf'
),
FILEGROUP Group4
(
NAME = Base_24,
FILENAME = 'D:\SQL\Base_4.ndf'
);
GO
USE Base_2;
```

Create partition table with left range (filegroups must exist):

```
USE Base_Partition_1;
--      Creation of partition function
CREATE PARTITION FUNCTION Partition_Function (INT)
AS RANGE LEFT FOR VALUES (1, 100, 1000) ;
/*
LEFT -   col_1 <=1; col_1 > 1 AND col_1 <= 100; col_1 > 100 AND col_1 <= 1000; col_1 > 1000
RIGHT - col_1 <1; col_1 >= 1 AND col_1 < 100; col_1 >= 100 AND col_1 < 1000; col_1 >= 1000
*/
GO
--      Creation of partition scheme
CREATE PARTITION SCHEME Partition_Scheme
AS PARTITION Partition_Function
TO (Group1, Group2, Group3, Group4) ;
GO
--      Creation of partition table
CREATE TABLE Partition_Table
(
col_1 INT,
col_2 CHAR(10)
)
ON Partition_Scheme (col_1) ;
GO
```

Creating a RANGE RIGHT partition function on a DATETIME column:

```
CREATE PARTITION FUNCTION Partition_Function_2 (DATETIME)
AS RANGE RIGHT FOR VALUES ('20220201', '20220301', '20220401',
                           '20220501', '20220601', '20220701', '20220801',
                           '20220901', '20221001', '20221101', '20221201');
```

Partitions	1	2	…	11	12
Values	col_4 < 1.02.2022	col_4 >= 1.02.2022 AND col_4 < 1.03.2022		col_4 >= 1.11.2022 AND col_4 < 1.12.2022	col_4 >= 1.12.2022

Creating a RANGE LEFT partition function on a DATETIME column:

```
CREATE PARTITION FUNCTION PARTITION_FUNCTION_2 (DATETIME)
AS RANGE LEFT FOR VALUES  ('20220201', '20220301', '20220401',
                           '20220501', '20220601', '20220701', '20220801',
                           '20220901', '20221001', '20221101', '20221201');
```

Partitions	1	2	…	11	12
Values	col_4 <= 1.02.2022	col_4 > 1.02.2022 AND col_4 <= 1.03.2022		col_4 > 1.11.2022 AND col_4 <= 1.12.2022	col_4 > 1.12.2022

Creating a RANGE RIGHT partition function on a CHAR column:

```
CREATE PARTITION FUNCTION PARTITION_FUNCTION_3 (char(20))
AS RANGE RIGHT FOR VALUES ('BT', 'GTB', 'TG');
```

Partitions	1	2	3	4
Values	col_4 < BT…	col_4 >= BT.. . AND col_4 < GTB…	col_4 >= GTB… AND col_4 < TG…	col_4 >= TG…

Creating a RANGE LEFT partition function on a CHAR column:

```
CREATE PARTITION FUNCTION PARTITION_FUNCTION_3 (char(20))
AS RANGE LEFT FOR VALUES ('BT', 'GTB', 'TG');
```

Partitions	1	2	3	4
Values	col_4 <= BT…	col_4 > BT… AND col_4 <= GTB…	col_4 >=GTB… AND col_4 <= TG…	col_4 > TG…

DATA COMPRESSION

System tables cannot be enabled for compression. When you are creating a table, data compression can be set to ROW, PAGE or NONE. The default value is NONE.

In Base_1 compress the table Table_7 by using row compression:

```
USE Base_1;
CREATE TABLE Table_7
(
col_1 INT,
col_2 NVARCHAR(200)
)
WITH (DATA_COMPRESSION = ROW);
```

In Base_1 compress the table Table_71 by using page compression:

```
USE Base_1;
CREATE TABLE Table_71
(
col_1 INT,
col_2 NVARCHAR(200)
)
WITH (DATA_COMPRESSION = PAGE);
```

In Base_1 do not compress the table Table_72:

```
USE Base_1;
CREATE TABLE Table_72
(
col_1 INT,
```

```
col_2 NVARCHAR(200)
)
WITH (DATA_COMPRESSION = NONE);
```

To evaluate how changing the compression state will affect a table, an index, or a partition, use the **sp_estimate_data_compression_savings** stored procedure. Syntax:

```
sp_estimate_data_compression_savings
'schema_name',
'object_name',
index_id,
partition_number,
'data_compression';
```

Returns the current size of the requested object and estimates the object size for the requested compression state. Compression can be evaluated for whole tables or parts of tables. This includes heaps, clustered indexes, nonclustered indexes, columnstore indexes, indexed views, and table and index partitions. The objects can be compressed by using row, page, columnstore or columnstore archive compression. If the table, index, or partition is already compressed, you can use this procedure to estimate the size of the table, index, or partition if it is recompressed.

Arguments:

'*schema_name*' - Is the name of the database schema that contains the table or indexed view. If *schema_name* is NULL, the default schema of the current user is used.

'*object_name*' - Is the name of the table or indexed view.

index_id - Is the ID of the index. *index_id* is **int**, and can be one of the following values: the ID number of an index, NULL, or 0 if *object_id* is a heap. To return information for all indexes for a base table or view, specify NULL. If you specify NULL, you must also specify NULL for *partition_number*.

partition_number - Is the partition number in the object. *partition_number* is **int**, and can be one of the following values: the partition number of an index or heap, NULL or 1 for a nonpartitioned index or heap.

'*data_compression*' - Is the type of compression to be evaluated. *data_compression* can be one of the following values: NONE, ROW, PAGE, COLUMNSTORE.

Get information about compression of the Table_Compression_Row table:

```
EXEC sp_estimate_data_compression_savings 'dbo', 'Table_Compression_Row',
NULL, NULL, 'ROW' ;
EXEC sp_estimate_data_compression_savings 'dbo', 'Table_Compression_PAGE',
NULL, NULL, 'PAGE' ;
```

Exercises.

1. Create unique, clustered index 'ind_ col_3' for column 'col_31' of the table Table_3 (this table must not have a primary key).
2. Create unique, nonclastered index 'ind_col_21' for column 'col_21' of the table Table_3.
3. Create nonclastered index 'ind_col_11' for column 'col_11' of the table Table_3.
4. Create nonclastered index 'ind_col_41' for column 'col_41' of the table Table_3.
5. Rebuild the index 'ind_col_41' of the table Table_3.
6. Disable index 'ind_col_41' of the table Table_3.
7. Enable index 'ind_col_41' of the table Table_3 by rebuilding it.
8. Rebuild all indexes of the table Table_3 (by using ALTER statement).
9. Rebuild all indexes of the table Table_3 by using specifyed parameters.
10. Rename index 'Ind_col_41' of the table Table_3, new name is 'Indcol_4'.
11. Delete index 'Ind_col_41' of the table Table_3.
12. Rebuild index 'ind_col_11' of the table Table_3.
13. Rebuild all indexes of the table Table_3 (by using DBCC statement).
14. Display information about indexes of the table Table_3.
15. In database Base_1 in the table Table_3 add and then delete constraint FOREIGN KEY.
16. In database Base_1 add constraint CHECK, named as 'col_check_3', to the column 'col_31' of the table Table_3. In accordance with this constraint the value of the column 'col_31' must exceed 1.
17. In database Base_1 delete constraint CHECK named as 'col_check_3' from the table Table_3.
18. In database Base_1 add constraint DEFAULT, named as 'col_31_default', to the column 'col_31' of the table Table_3. In accordance with this constraint the default value of the column 'col_31' is 50.
19. In database Base_1 delete constraint DEFAULT, named as 'col_31_default', from the table Table_3.
20. In database Base_1 add 'Sales_ID' new column, which have PRIMARY KEY and IDENTITY constraints to the table Sales. The name of the constraint - Sales_PK.
21. In database Base_1 delete Sales_PK constraint from table Sales.
22. In database Base_1 add new column col_51 to the table Table_3, which have constraint UNIQUE named as 'col_51_unique'.
23. In database Base_1 from the table Table_3 delete constraint UNIQUE, which name is 'col_51_unique'.
24. In database Base_1 add constraint UNIQUE for the column col_16 of the table Table_3 by creating an index. The name of the index is 'ind_col_16'.
25. In database Base_1 add constraint CHECK, named as 'col_17_check', to the column 'col_17' of the table Table_3. In accordance with this constraint each symbol of the column 'col_17' must be symbol-digit.
26. Create partition table with left range (filegroups must exists).
27. In Base_1 compress the table Table_5 by using row compression.
28. In Base_1 compress the table Table_51 by using page compression.
29. In Base_1 do not compress the table Table_52.