

# Practice Class 8

## Objectives

Relational Database Indexes - based on SQL Server.

Note: You must follow the response template provided.

## Assignment 8.1

This problem is based on the *Production.WorkOrder* table from the *AdventureWorks2012* database. You must download the *AdventureWorks2012.bak*<sup>4</sup> file and restore the database following the tutorial: *Restore to SQL Server*<sup>5</sup>. The *Production.WorkOrder* table has a *Clustered Unique index* associated with the *WorkOrderID* PK<sup>6</sup>.

Using the *SQL Server Profiler and Query Execution Plan*<sup>7</sup> tools, **record and discuss** the values obtained (index/query/rows/cost/pag. reads/...) for each of the experiences below. It is recommended that you present the results obtained in the form of a table containing the following elements:

#	Query	Rows	Cost	Pag. Reads	Time (ms)	Index used	Index on.
1	select * from Production.WorkOrder	72591	.484	531	1171	...	Clustered Index Scan
2	...	...	...	...	...	...	...

Note: Before executing each of the queries you must execute the following instructions:

```
DBCC FREEPROCCACHE;  
DBCC DROPCLEANBUFFERS;
```

### Experiences:

- #1. Index: WorkOrderID (PK)  
Query: select \* from Production.WorkOrder
- #2. Index: WorkOrderID (PK)  
Query: select \* from Production.WorkOrder where WorkOrderID=1234
- #3. Index: WorkOrderID (PK)  
Query1: SELECT \* FROM Production.WorkOrder  
WHERE WorkOrderID between 10000 and 10010  
  
Query2: SELECT \* FROM Production.WorkOrder  
WHERE WorkOrderID between 1 and 72591

---

<sup>4</sup> <https://github.com/Microsoft/sql-server-samples/releases/download/adventureworks/AdventureWorks2012.bak>

<sup>5</sup> <https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms#restore-to-sql-server>

<sup>6</sup> [https://github.com/CarlosCosta-UA/BD-UA/blob/main/aula8/adventure\\_works\\_2012\\_clustered\\_idx.JPG](https://github.com/CarlosCosta-UA/BD-UA/blob/main/aula8/adventure_works_2012_clustered_idx.JPG)

<sup>7</sup> Available in SQL Server Management Studio

- #4. Index: WorkOrderID (PK)  
Query: SELECT \* FROM Production.WorkOrder  
WHERE StartDate = '2012-05-14'
- #5. Index: ProductID  
Query: SELECT \* FROM Production.WorkOrder WHERE ProductID = 757
- #6. Index: ProductID Covered (StartDate)  
Query1: SELECT WorkOrderID, StartDate FROM Production.WorkOrder  
WHERE ProductID = 757  
Query2: SELECT WorkOrderID, StartDate FROM Production.WorkOrder  
WHERE ProductID = 945  
Query3: SELECT WorkOrderID FROM Production.WorkOrder  
WHERE ProductID = 945 AND StartDate = '2011-12-04'
- #7. Index: ProductID and StartDate  
Query: SELECT WorkOrderID, StartDate FROM Production.WorkOrder  
WHERE ProductID = 945 AND StartDate = '2011-12-04'
- #8. Index: Composite (ProductID, StartDate)  
Query: SELECT WorkOrderID, StartDate FROM Production.WorkOrder  
WHERE ProductID = 945 AND StartDate = '2011-12-04'

## Assignment 8.2

Base the following table:

```
CREATE TABLE mytemp (
    rid BIGINT /*IDENTITY (1,1)*/ NOT NULL,
    at1 INT NULL,
    at2 INT NULL,
    at3 INT NULL,
    lixo varchar(100) NULL
);
```

- Set *rid* as the primary key of the *Clustered Index type*.
- Record the entry times of 50,000 new records (tuples) in the table using the code below:

```
-- Record the Start Time
DECLARE @start_time DATETIME, @end_time DATETIME;
SET @start_time = GETDATE();
PRINT @start_time

-- Generate random records
DECLARE @val as int = 1;
DECLARE @nelem as int = 50000;

SET nocount ON

WHILE @val <= @nelem
BEGIN
    DBCC DROPCLEANBUFFERS;                -- need to be sysadmin

    INSERT mytemp (rid, at1, at2, at3, lixo)
```

```

SELECT cast((RAND()*@nelem*40000) as int), cast((RAND()*@nelem) as int),
       cast((RAND()*@nelem) as int), cast((RAND()*@nelem) as int),
       'garbage... garbage... garbage... garbage... garbage... garbage...
garbage... garbage... garbage';
SET @val = @val + 1;
END

PRINT 'Inserted ' + str(@nelem) + ' total records'

-- Duration of Insertion Process
SET @end_time = GETDATE();
PRINT 'Milliseconds used: ' + CONVERT(VARCHAR(20), DATEDIFF(MILLISECOND,
@start_time, @end_time));

```

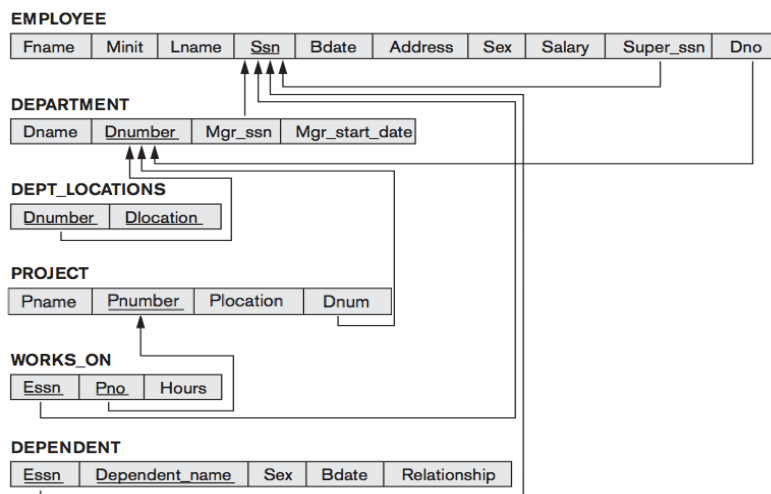
What is the percentage of fragmentation of indexes and occupation of index pages?

- Change the *fillfactor* (for example: 65, 80, and 90) from *the clustered index* and see the effect on insertion times.
- Change the table *mytemp* that make the *rid* attribute as type *identity*. Re-measure insertion times<sup>8</sup>.
- Create an index for each attribute of the *mytemp* table. Compare the insertion times obtained, without and with all indexes. What can you conclude?

Note: The results obtained in this exercise may vary depending on the type of computer/virtual machine (e.g. HDD/SSD hardware) and the machine load at the time the experiment is taking place;

## Assignment 8.3

Based on the database schema presented in the figure below (developed in the theoretical classes):



- Define the indexes that you find convenient for each of the relationships. Please note that we need to do the following database queries:

<sup>8</sup> You must change the code provided in point (b) for this new situation.

- i. The employee with certain number ssn;
- ii. The employee(s) with a certain first and last name;
- iii. Employees working for a particular department;
- iv. Employees who work for certain project;
- v. Dependents of a particular employee;
- vi. The projects associated with a given department;