Playing with a small DB and storage

Setup visualization libraries

In [2]:
```
def displaySectionCaption(caption, color='coral'):
  html_string = f'<hr><strong><p style="color:{color};font-size:16px;">{caption}</p.</strong>'
  display(HTML(html_string))
```

We study a simplified IO model for HDDs and SSDs in CMPE-138. The model will work well in practice, for our query optimzation and data layout problems.

In [4]:
```
import math
from math import ceil, log

# We'll use MBs -- for basic i to MBs
(MB, GB, TB, KB, Bytes) = (1.0, 1024.0, 1024.0*1024.0,
                           1.0/1024.0, 1.0/(1024.0*1024))

# 64 MB-Blocks (default)
PageSizeMB = 64.0*MB
size_of_types = {'int64': 8, 'int32': 4, 'double': 8, 'char': 1} # in bytes

class IOdevice:
  def __init__(self, accessTime, scanSpeed, C_w):
    self.C_r = 1.0  # Cost of reads
    self.C_w = C_w  # Cost of writes relative to reads
    self.accessTime = accessTime
    self.scanSpeed = scanSpeed

  # Read costs: Simple IOcost model using Access time + Scan speeds
  def read_pages_cost(self, numPages):
    # Assume you need to read full pages. (i.e., no partial pages)
    numPages = math.ceil(numPages)
    tsecs = numPages*self.accessTime  # time to access
    tsecs += numPages*PageSizeMB/self.scanSpeed # time to scan
    return (tsecs)

  def write_pages_cost(self, numPages):
    return self.C_w*self.read_pages_cost(numPages)

# Example IO devices in 2024
# Access and Scan speeds in [seconds, MBps], Cw cost of write vs reads.
ram1 = IOdevice(100*pow(10, -9), 100.0*1024, 1.0)
ssd1 = IOdevice(10*pow(10, -6), 5.0*1024, 1.0) # 10 microsecs, 5GBps
hdd1 = IOdevice(10*pow(10, -3), 100.0, 1.0) # 10 millissecs, 100 MBps
# machine to machine over network (modeling a network as an IO device)
m2m1 = IOdevice(10*pow(10, -6), 5.0*1024, 1.0) # 1 micro, 5 GBps

IOdevices1 = {'HDD': hdd1, 'SSD': ssd1, 'RAM': ram1}
```

In [6]:
```
"""
Basic physical table
"""
class Table:
  def __init__(self, sizeInMBs, rowSize):
    self.sizeInMBs = sizeInMBs
    self.rowSize = rowSize
    self.numRows = ceil(self.sizeInMBs/self.rowSize)

    # self.numTuples = numTuples
    self.isSorted = False
    self.isHPed = False

  # P(R) -- number of Pages for table
  def P(self):
    P = ceil(self.sizeInMBs/PageSizeMB)
    return P
  def RowSize(self):
    return self.rowSize
  def T(self):
    return self.numRows
  def SizeInMBs(self):
    return self.sizeInMBs

  # Keeping track of is table sorted, HPed, or neither (default)
  def Sort(self):
    self.isSorted = True
    self.isHPed = False
  def HP(self):
    self.isSorted = False
    self.isHPed = True
  def Reset(self):
    self.isSorted = False
    self.isHPed = False
```

Exercises:

In [12]:
```
# Spotify Songs Table [songid: int64, title: text, name: text, genre: text]
#    -- Size of row = 8 bytes (int64) + avg size of title+name+genre.
#    -- Assume avg row size = 1024 Bytes
songs_rowSize = 1024.0*Bytes
songs_numRows = 500000000.0 # 500 million songs

"""Problem 1:
Calculate the size (MBs) of SongsTable, and num pages."""
songsTableSize = songs_rowSize * songs_numRows / (1024 * 1024)  # Convert to MBs
songsNumPages = math.ceil(songsTableSize / PageSizeMB)
print(f"Songs Table Size: {songsTableSize:.2f} MB, Number of Pages: {songsNumPages}")


"""Problem 2: Read costs
Compute the cost in seconds to read 100 pages from the SongsTable"""
numPagesToRead = 100
hdd_read_cost = hdd1.read_pages_cost(numPagesToRead)
ssd_read_cost = ssd1.read_pages_cost(numPagesToRead)
ram_read_cost = ram1.read_pages_cost(numPagesToRead)
print(f"Read cost (HDD): {hdd_read_cost:.6f} sec")
print(f"Read cost (SSD): {ssd_read_cost:.6f} sec")
print(f"Read cost (RAM): {ram_read_cost:.6f} sec")



"""Problem 3: Effect of caching
Read 200 pages. 1st check RAM.
- Cache hit of 90% in RAM.
- For RAM cache misses (the other 10%), 75% are in SSD and 25% are in HDD."""
numPagesToRead = 200
cache_hit_ratio = 0.90
ssd_miss_ratio = 0.75
hdd_miss_ratio = 0.25

ram_hits = cache_hit_ratio * numPagesToRead
ram_misses = numPagesToRead - ram_hits
ssd_hits = ssd_miss_ratio * ram_misses
hdd_hits = hdd_miss_ratio * ram_misses

total_read_time = (
    ram1.read_pages_cost(ram_hits) +
    ssd1.read_pages_cost(ssd_hits) +
    hdd1.read_pages_cost(hdd_hits)
)

print(f"Total read time with caching: {total_read_time:.6f} sec")
```

```
Songs Table Size: 0.47 MB, Number of Pages: 1
Read cost (HDD): 65.000000 sec
Read cost (SSD): 1.251000 sec
Read cost (RAM): 0.062510 sec
Total read time with caching: 3.550168 sec
```

In [ ]: