Disk Drive: Parameters

\[ \begin{align*}
D_{\text{cyl}} & \quad \text{total number of cylinders} \\
D_{\text{track}} & \quad \text{total number of tracks} \\
D_{\text{sec}} & \quad \text{total number of sectors} \\
D_{\text{tpc}} & \quad \text{number of tracks per cylinder (\(=\) number of surfaces)} \\
D_{\text{cmd}} & \quad \text{command interpretation time} \\
D_{\text{rot}} & \quad \text{time for a full rotation} \\
D_{\text{rdsettle}} & \quad \text{time for settle for read} \\
D_{\text{wrsettle}} & \quad \text{time for settle for write} \\
D_{\text{hdswitch}} & \quad \text{time for head switch}
\end{align*} \]
Disk Drive: Parameters (2)

\[ D_{\text{zone}} \]  
\hspace{1cm} \text{total number of zones}

\[ D_{\text{zcyl}}(i) \]  
\hspace{1cm} \text{number of cylinders in zone } i

\[ D_{\text{zspt}}(i) \]  
\hspace{1cm} \text{number of sectors per track in zone } i

\[ D_{\text{zspc}}(i) \]  
\hspace{1cm} \text{number of sectors per cylinder in zone } i \left( = D_{\text{tpc}} D_{\text{zspt}}(i) \right)

\[ D_{\text{zscan}}(i) \]  
\hspace{1cm} \text{time to scan a sector in zone } i \left( = D_{\text{rot}} / D_{\text{zspt}}(i) \right)
Disk Drive: Parameters (3)

\[ D_{\text{seekavg}} \] average seek costs
\[ D_{\text{clim}} \] parameter for seek cost function
\[ D_{\text{ca}} \] parameter for seek cost function
\[ D_{\text{cb}} \] parameter for seek cost function
\[ D_{\text{cc}} \] parameter for seek cost function
\[ D_{\text{cd}} \] parameter for seek cost function

\[ D_{f\text{seek}}(d) \] cost of a seek of \( d \) cylinders
\[
D_{f\text{seek}}(d) = \begin{cases} 
D_{ca} + D_{cb}\sqrt{d} & \text{if } d \leq D_{\text{clim}} \\
D_{cc} + D_{cd}d & \text{if } d > D_{\text{clim}} 
\end{cases}
\]

\[ D_{\text{frot}}(s, i) \] rotation cost for \( s \) sectors of zone \( i \) (\( = sD_{\text{zscan}}(i) \))
Extraction of Disk Drive Parameters

- documentation: often not sufficient
- mapping: interrogation via SCSI-Mapping command (disk drives lie)
- use benchmarking tools, e.g.:
  - Diskbench
  - Skippy (Microbenchmark)
  - Zoned
Seek Curve Measured with Diskbench
Skippy Benchmark Example
Interpretation of Skippy Results

- x-axis: distance (sectors)
- y-axis: time
- difference topmost/bottommost line: rotational latency
- difference two lowest ‘lines’: head switch time
- difference lowest ‘line’ topmost spots: cylinder switch time
- start lowest ‘line’: minimal time to media
- plus other parameters
Upper bound on Seek Time

**Theorem (Qyang)**

*If the disk arm has to travel over a region of $C$ cylinders, it is positioned on the first of the $C$ cylinders, and has to stop at $s - 1$ of them, then $sD_{fseek}(C/s)$ is an upper bound for the seek time.*
Database Buffer

The database buffer

1. is a finite piece of memory,

2. typically supports a limited number of different page sizes (mostly one or two),

3. is often fragmented into several buffer pools,

4. each having a replacement strategy (typically enhanced by hints).

Given the page identifier, the buffer frame is found by a hashtable lookup. Accesses to the hash table and the buffer frame need to be synchronized. Before accessing a page in the buffer, it must be fixed. These points account for the fact that the costs of accessing a page in the buffer are therefore greater than zero.
### Buffer Accesses

Consider page accesses in a buffer with 2 pages:

<table>
<thead>
<tr>
<th>page no</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read page 0, place it in buffer</td>
</tr>
<tr>
<td>1</td>
<td>read page 1, place it in buffer</td>
</tr>
<tr>
<td>0</td>
<td>fix page 0 in buffer</td>
</tr>
<tr>
<td>2</td>
<td>swap out a page (e.g. 1), read 2, place it in buffer</td>
</tr>
<tr>
<td>0</td>
<td>fix page 0 in buffer</td>
</tr>
<tr>
<td>3</td>
<td>swap out a page, read 3, place it in buffer</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- replacement strategy is important
- unfixes omitted
Replacement Strategies

Some popular replacement strategies:

- random
- fifo
- lru
- Q2

lru is very popular
Replacement Strategies - random

- when a new page slot is needed, remove a random other page from the buffer
- easy to implements, needs no additional memory
- but does not take the access patterns into account
- primarily used as base line
- suitable for analytic results
Replacement Strategies - fifo

- first in - first out
- remove the page that was placed in the buffer first
- easy to implement, needs no/few additional memory
- but does not adapt very well to access patterns
- increasing buffer size may hurt it

Fifo Anomaly:
- access pattern: 3 2 1 0 3 2 4 3 2 1 0 4
- buffer sizes: 3 vs. 4
Replacement Strategies - lru

- least recently used
- remove the page that has not been accessed for longest time
- requires a priority queue/linked list
- adapt to access patterns, popular pages stay in memory
- but slow to remove pages

very popular replacement strategy
Replacement Strategies - 2Q

- two queues
- a fifo queue and a lru queue
- place pages first in fifo, if they are accessed again place them in lru
- gets rid of pages that are accessed only once fast
- superior to lru, example of a "real" replacement strategy
Replacement Strategies - Effect on the Cost Model

- replacement affects the costs
- cost model needs predictions, though
- very hard to do in general

Typical approaches:
- ignore buffer effects
- assume random replacement
- make use of known access characteristics
Physical Database Organization

The database organizes the physical storage in multiple layers:

1. partition: sequence of pages (consecutive on disk)
2. extent: subsequence of a partition
3. segment (file): logical sequence of pages (implemented e.g. as set of extents)
4. record: sequence of bytes stored on a page

Note:

- partition/extent/page/record are physical structures
- a segment is a logical structure
Physical Storage of Relations

Mapping of a relation’s tuples onto records stored on pages in segments:
Access to Database Items

- database item: something stored in DB
- database item can be set (bag, sequence) of items
- access to a database item then produces stream of smaller database items
- the operation that does so is called *scan*
Scan Example

Using a relation scan \texttt{rscan}, the query

\begin{verbatim}
select * 
from Student
\end{verbatim}

can be answered by \texttt{rscan(Student)}

\texttt{(segments? extents?): Assumption:}

\begin{itemize}
  \item segment scans and each relation stored in one segment
  \item segment and relation name identical
\end{itemize}

Then \texttt{fscan(Student)} and \texttt{Student} denote scans of all tuples in a relation
Model of a Segment

- for our cost model, we need a model of segments.
- we assume an extent-based segment implementation.
- every segment then is a sequence of extents.
- every extent can be described by a pair \((F_j, L_j)\) containing its first and last cylinder.
  (For simplicity, we assume that extents span whole cylinders.)
- an extent may cross a zone boundary.
- hence: split extents to align them with zone boundaries.
- segment can be described by a sequence of triples \((F_i, L_i, z_i)\) ordered on \(F_i\) where \(z_i\) is the zone number in which the extent lies.
Model of a Segment

\[ S_{\text{ext}} \] number of extents in the segment
\[ S_{\text{cfirst}}(i) \] first cylinder in extent \( i \) \( (F_i) \)
\[ S_{\text{clast}}(i) \] last cylinder in extent \( i \) \( (L_i) \)
\[ S_{\text{zone}}(i) \] zone of extent \( i \) \( (z_i) \)
\[ S_{\text{cpe}}(i) \] number of cylinders in extent \( i \) \( (= S_{\text{clast}}(i) - S_{\text{cfirst}}(i) + 1) \)
\[ S_{\text{sec}} \] total number of sectors in the segment
\[ (= \sum_{i=1}^{S_{\text{ext}}} S_{\text{cpe}}(i)D_{\text{zspc}}(S_{\text{zone}}(i))) \]
Slotted Page

- page is organized into areas (slots)
- slots point to data chunks
- slots may point to other pages
Tuple Identifier (TID)

TID is conjunction of

- page identifier (e.g. partition/segment no, page no)
- slot number

TID sometimes called Row Identifier (RID)
Record Layout

Different layouts possible:

- Fixed-length size variable-length size variable-length size variable-length
- Fixed-length offset offset offset variable-length variable-length
- Codes data
  - Fixed-length variable-length strings
  - Length and offset encoding
  - Encoding for dictionary-based compression
Record Layout (2)

Record layout is a compromise:

- space consumption vs. CPU
- data model specific properties: e.g. generalization
- versioning / easy schema migration
- record layout typically not trivial
- accessing an attribute value has non-zero cost
Physical Algebra

- building blocks for query execution
- implements the algorithms for query execution
- very generic, reusable components
- describes the general execution approach
- annotated with predicates etc. for query specific parts
Iterator Concept

The general interface of each operator is:

- open
- next
- close

All physical algebraic operators are implemented as iterators.

- produce a stream of data items (tuples)

Implementations vary slightly for performance tuning (concept the same):

- first/next instead of next
- blocks of tuples instead of single tuples
Iterator Example

\[ \sigma \]
\[ \times \]
\[ \sigma \quad \Gamma \]
\[ \text{scan} \quad \text{scan} \]

Note: all details (subscripts, implementations etc.) are omitted here
Pipelining

Pipelining is fundamental for the physical algebra:

- physical operators are iterators over the data
- they produce a stream of single tuples
- tuple stream if passed through other operators
- pipelining operators just pass the data through, they only filter or augment
- data is not copied or materialized
- very efficient processing

*pipeline breakers* disrupt this pipeline and materialize data:

- very expensive, can cause superfluous work
- sometimes cannot be avoided, though
Simple Scan

- a \texttt{rscan} operation is rarely supported.
- instead: scans on segments (files).
- since a (data) segment is sometimes called \textit{file}, the correct plan for the above query is often denoted by \texttt{fscan(Student)}.

Several assumptions must hold:
- the \texttt{Student} relation is not fragmented, it is stored in a single segment,
- the name of this segment is the same as the relation name, and
- no tuples from other relations are stored in this segment.

Until otherwise stated, we assume that these assumptions hold. Instead of \texttt{fscan(Student)}, we could then simply use \texttt{Student} to denote leaf nodes in a query execution plan.
Attributes/Variables and their Binding

select * from Student

can be expressed as $Student[s]$ instead of $Student$. Result type: set of tuples with a single attribute $s$. $s$ is assumed to bind a pointer

- to the physical record in the buffer holding the current tuple or
- a pointer to the slot pointing to the record holding the current tuple
Building Block

- scan
- a leaf of a query execution plan

Leaf can be complex.
But: Plan generator does not try to reorder within building blocks
Nonetheless:
  - building block organized around a single database item
If more than a single database item is involved: access path