Enterprise Programming 2

Lesson 04: Wrapped Responses, Errors and Pagination

Bogdan Marculescu

Goals

- Understand the concept of *Wrapped Responses*, and how it helps in logging/debugging of errors
- Understand and tune how Spring deals with uncaught exceptions and bean validation
- Understand how to enable *Pagination* with *Links* when dealing with requests retrieving large amounts of data

Wrapped Responses

Errors

- HTTP request can fail due to a 4xx or 5xx error
- But what was the reason?
- How to tell the user that a 400 was due to an invalid query parameter s/he provided?
- Not so great solution: provide an *error message* as a HTML body payload
- Why not so great? Need to marshal payloads in different ways based on status code…
  - ie JSON when OK, and HTML when errors

JSON Wrapped Response

```
{
  "code": 400,
  "status": "ERROR",
  "data": null,
  "message": "Invalid query parameter x"
}
```

```
{
  "code": 200,
  "status": "SUCCESS",
  "data": {foo:4, bar:"a"}
  "message": null
}
```

Wrapping

- Instead of returning a payload directly in the HTTP body of the response, wrap it in a JSON object
- The payload will be in a field called “**data**”
- If there is any error, then “**data**” will be null, with a field “**message**” explaining reason, ie the *error message*
- Can also have fields for the status of the response (eg success vs failure/error)

Benefits

- Error message, if any, is part of the response body, easy to access
- Unmarshaling of HTTP response payload from JSON regardless of success or failure/error
  - ie, success and error responses have the same JSON structure
- Very limited overhead

Standard

- How to specify which fields to use in a wrapped response?
- This is not part of HTTP, nor something discussed in REST
- There is no “standard”
- Could use your own format for your APIs
- Or use some existing specification like *JSend*
  - *https://labs.omniti.com/labs/jsend*

Errors/Validation In Spring

Validation

- In JEE, we have seen how *@annotation* constraints could be put on the inputs of EJB beans
  - same way as on JPA @Entity
- In Spring, we can do the same, which can be useful when handling query parameters
- But Spring does not do validation by default, needs to be activated with *@Validated*
  - *org.springframework.validation.annotation.Validated*

Exceptions

- When an exception is thrown and not caught, Spring creates a 500 HTTP response
  - exceptions in your code will not crash the whole server…
- When using *Wrapped Responses*, the format used by Spring might be different from ours, so we might want to override it
- Might be cases in which we want to manually throw exceptions, which should results in 400 responses with our Wrapped format

Overriding Spring Defaults

- To change how Spring deals with exceptions, we can create a bean tagged with *@ControllerAdvice,* extending *ResponseEntityExceptionHandler*

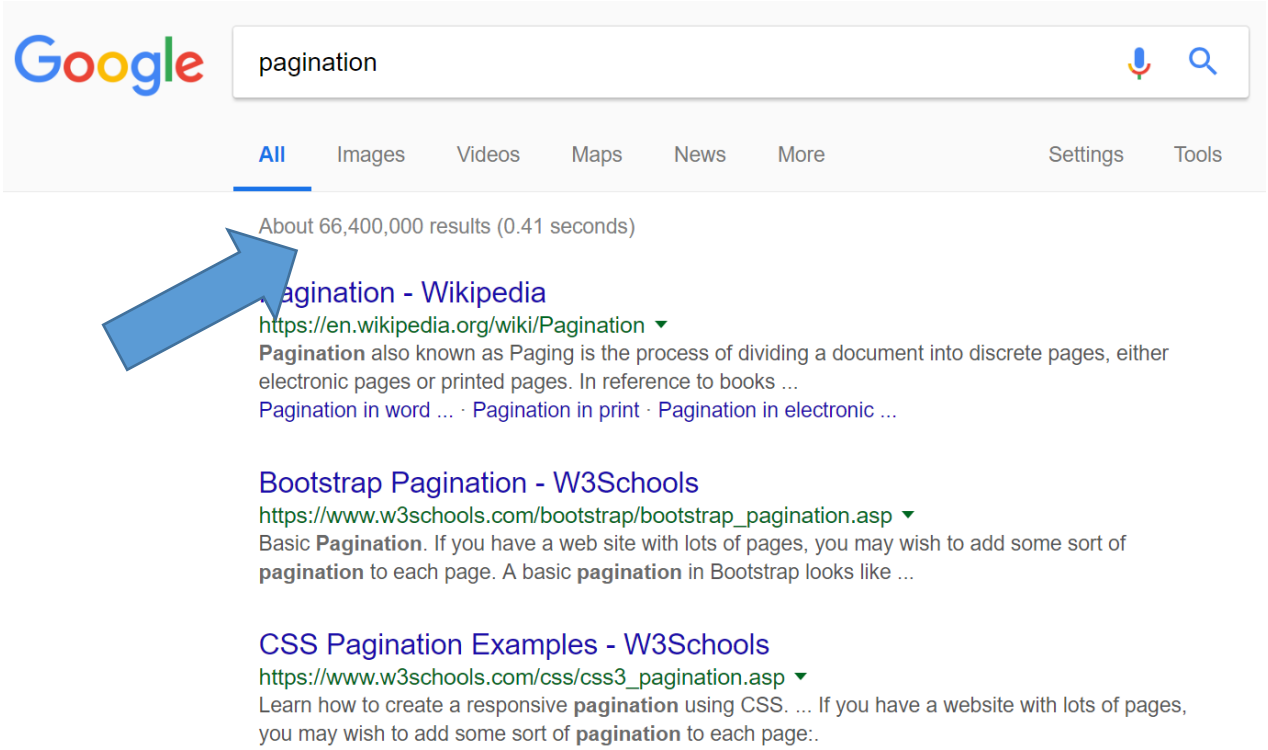
- Can have different exception handlers by using annotation *@ExceptionHandler* on different methods

Pagination

Amount of Data

- Example: **GET /news**
- Return all news in database, marshalling into JSON
- But what if the database has 2 billion news???
- You do not want to return terabytes of data for a single GET…
- It would end up in a easy to exploit Denial-Of-Service (DOS) attack

Searches


Google
pagination
All
Images
Videos
Maps
News
More
Settings
Tools
About 66,400,000 results (0.41 seconds)
agination - Wikipedia
https://en.wikipedia.org/wiki/Pagination
Pagination also known as Paging is the process of dividing a document into discrete pages, either electronic pages or printed pages. In reference to books ...
Pagination in word ... · Pagination in print · Pagination in electronic ...
Bootstrap Pagination - W3Schools
https://www.w3schools.com/bootstrap/bootstrap_pagination.asp
Basic Pagination. If you have a web site with lots of pages, you may wish to add some sort of pagination to each page. A basic pagination in Bootstrap looks like ...
CSS Pagination Examples - W3Schools
https://www.w3schools.com/css/css3_pagination.asp
Learn how to create a responsive pagination using CSS. ... If you have a website with lots of pages, you may wish to add some sort of pagination to each page:.


Pagination | GraphQL
https://graphql.org/learn/pagination/
Pagination. Different pagination models enable different client capabilities. A common use case in GraphQL is traversing the relationship between sets of objects ...
GitHub - vapor-community/pagination: Simple Vapor 3 Pagination
https://github.com/vapor-community/pagination
Mar 7, 2018 - Simple Vapor 3 Pagination. Contribute to vapor-community/pagination development by creating an account on GitHub.
GitHub - react-component/pagination: React Pagination
https://github.com/react-component/pagination
React Pagination. Contribute to react-component/pagination development by creating an account on GitHub.
Searches related to pagination
pagination example
pagination javascript
pagination html
pagination bootstrap
pagination website
pagination php
pagination design
pagination jquery
Gooooooooooogle
Previous
1 2 3 4 5 6 7 8 9 10
Next

Page

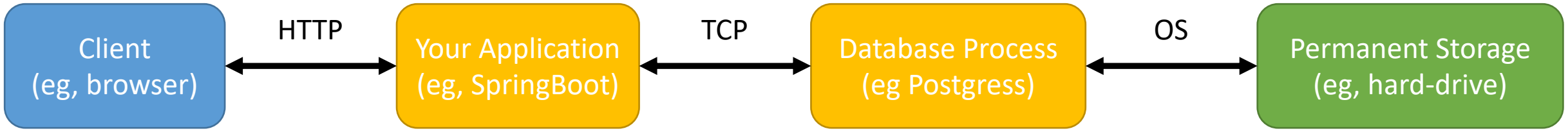
- Instead of billions of elements, just return a single *Page*
- A *Page* will contain *n* elements (eg 10 or 20) from the collection
- It will have information on the *previous* and the *next* page
- If you want, can iterate over the whole collection by checking one page at a time, following the *next* links

Two Types of Pagination

- **Offset Pagination**
  - likely the most common
  - can jump directly to a specific page
  - VERY INEFFICIENT, unless limited to read only a few pages from start
  - issues when database is manipulated while iterating over it
- **Keyset Pagination**
  - most efficient
  - bit more complex to implement
  - can’t jump directly to a specific page, without reading previous pages
    - not a problem if we just care of “*next*” page, or jumps ahead of only few pages (eg 5-10)

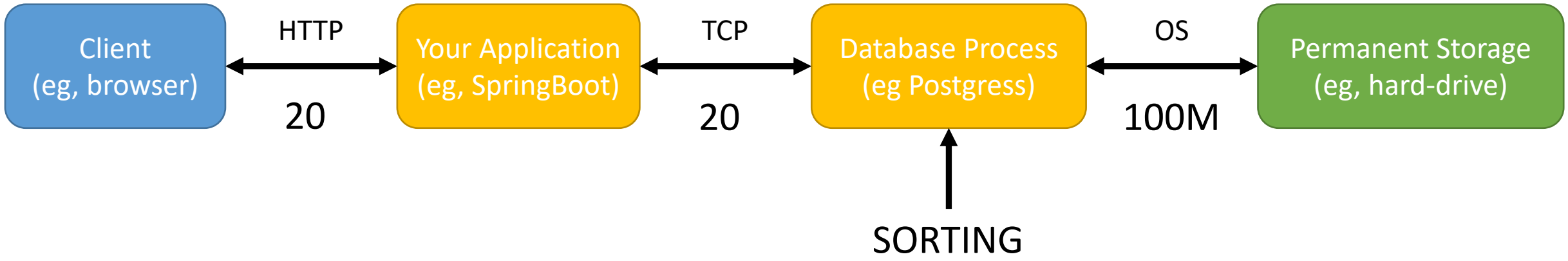
Efficiency

- Assume for example *100 millions* items in database
- Pages of small fixed size, eg *n=20*
- Data is sent through 3 steps



Ordering

- To iterate over collection of data in deterministic order, data must be sorted, eg with ORDER BY in SQL
- Sorting MUST be done on database, and NOT in the application
  - otherwise need to send all data to the application



Offset Pagination

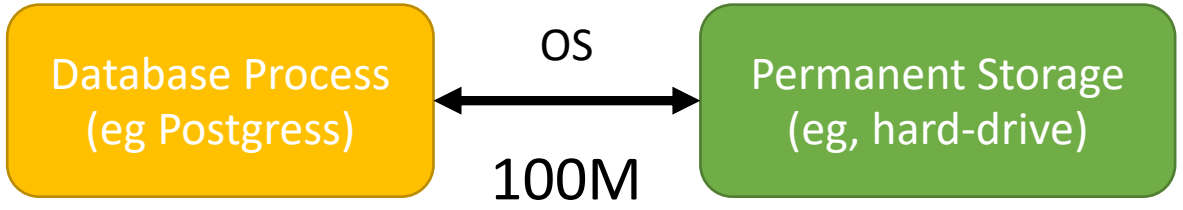
Offset/Limit

- When dealing with large collections, need a way to specify the boundaries of a *Page*
- Example: *GET /news?**offset**=40&**limit**=10*
- *Offset*: given the collection sorted like an array, this would be the starting index *i*
- *Limit:* starting from the offset, how many elements to return

Offset in SQL

- SQL has direct support for OFFSET and LIMIT
- Eg: *select * from News* ***ORDER BY*** *id desc* ***LIMIT*** *20* ***OFFSET*** *0*
- LIMIT: how many rows to return
- OFFSET: how many rows to skip before returning data
- Page: defined by OFFSET and LIMIT
  - 1st page: *LIMIT 20 OFFSET 0*
  - 2nd page: *LIMIT 20 OFFSET 20*
  - 3rd page: *LIMIT 20 OFFSET 40*
  - *etc.*

Why Inefficient?



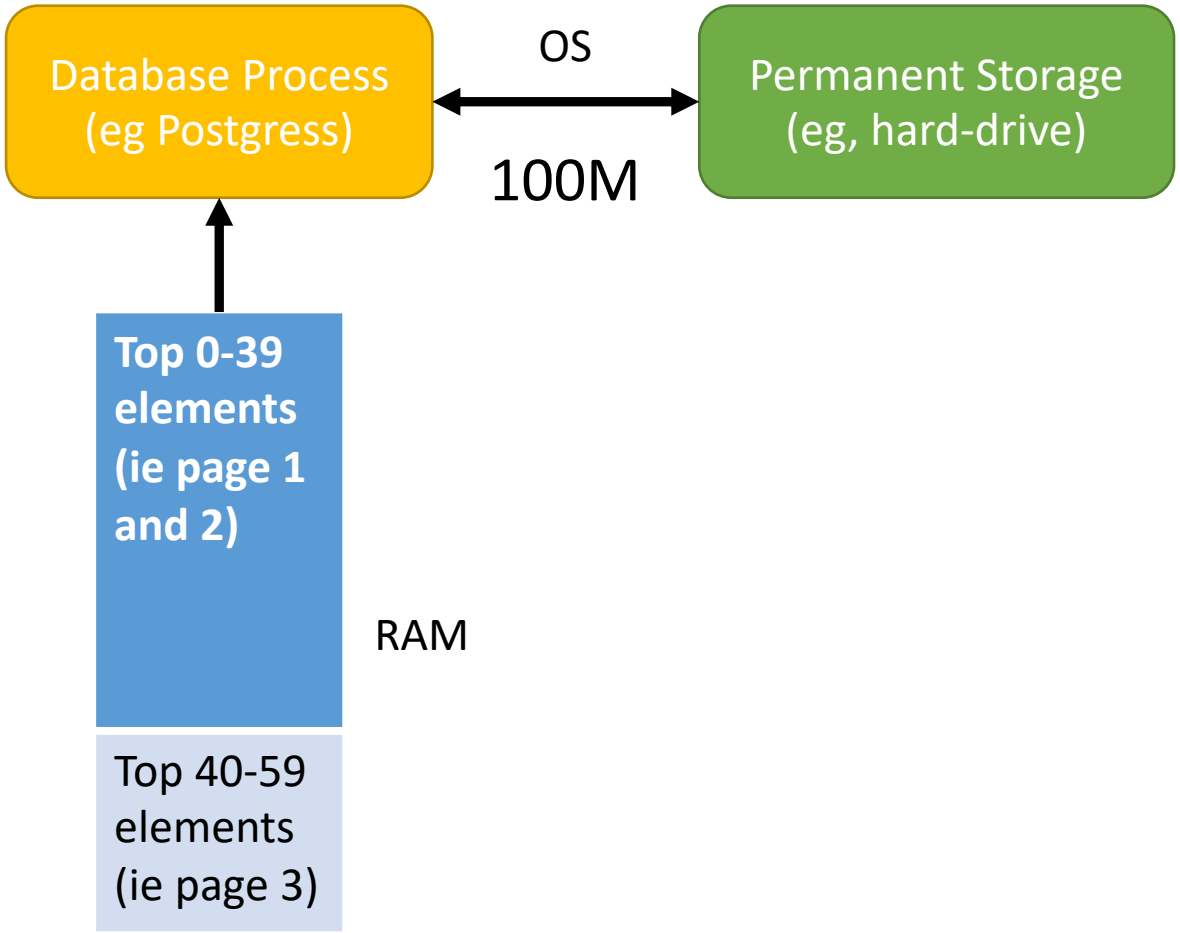
- To read page N, need to know the previous N-1 pages to skip
- To be able to sort, database has to keep a buffer of size OFFSET+LIMIT (which represent all pages till N)
- Have to read all data (eg 100M rows), but for sorting no need to keep in RAM all of that, as at most returning LIMIT rows
- If sorting on descending ID, only need to keep track of highest K=OFFSET+LIMIT ids
  - if when reading new row out of 100m, no need to store it in RAM if id smaller than the smallest in current K buffer

Reading Pages

- First page is fast: only 20 rows in RAM, which are fast to sort
- N*th* page requires a buffer of *20*N*, which needs to be sorted to determine the first *N-1* pages to skip
  - recall that sorting has complexity *O(n log n)*
- Last page? Database (eg Postgres) has to keep in RAM 100m rows, and sort them, to decide which are the first *n-1* pages to skip
  - if reading first page could take just a few *milliseconds*, reading last page could take *minutes*…
- **The higher the page number, the slower it will be**

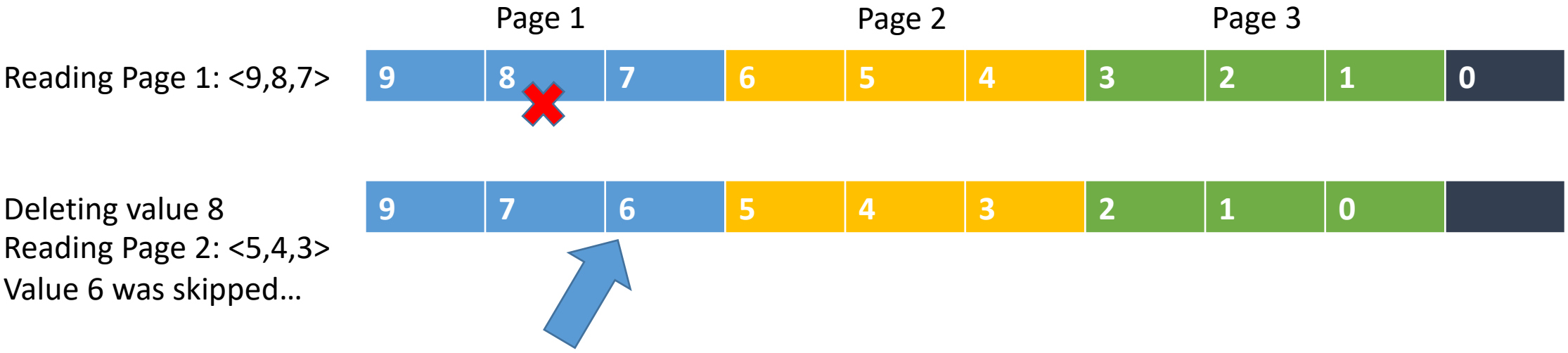
Ex. Reading Page 3

- OFFSET=40, LIMIT=20
- Only need a buffer *B* of size 60, and sort 60 elements
- If in current *B* the lowest id is X, then ignore all elements read from storage with id lower than X, as anyway impossible it will be part of the top 60
- Even if returning top 40-59, still need to determine which are the top 0-39 ☹

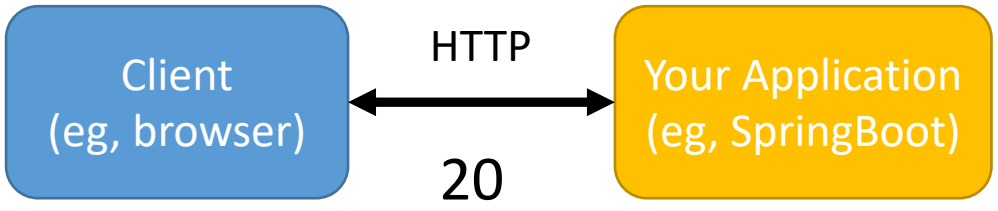


Problem: Database Modifications

- *Question*: what if, after reading Page 1, someone else does a DELETE of an element in first page?
- *Answer*: an element will skipped when reading Page 2
- Eg, assume page size n=3, and someone removes id 8



Links



- Database data will be converted into JSON, to send over HTTP to the client
- To access the next/previous pages, client could compute the needed offsets/limits
- Or, we could just provide valid URLs in the JSON responses with "*links*" to those pages
- This is an instance of HATEOAS
  - *Hypermedia as the Engine of Application State*
- Easier to navigate

dto = { “**list**”: […], *//the actual payload*

“**rangeMin**”: 40,

“**rangeMax**”: 49, *//so, 10 element pages*

*“**totalSize**”: 66400000,*

*“**_links**”:[*

*“**next**”: {“href”: “/news?**offset**=50&**limit**=10”},*

*“**self**”: {“href”: “/news?**offset**=40&**limit**=10”},*

*“**previous**” : {“href”: “/news?**offset**=30&**limit**=10”},*

]

}

Standard

- There is no official standard to define pages and links
- In the past, there were some attempts like HAL, but they look like abandoned

Keyset Pagination

Goal: Performance

- Given a page of size N, cost of reading it should be constant
  - ie, reading last page should not take much, much more time than first
- Deleting items in read pages should not impact the reading of the next pages

SQL WHERE on Last Read Item

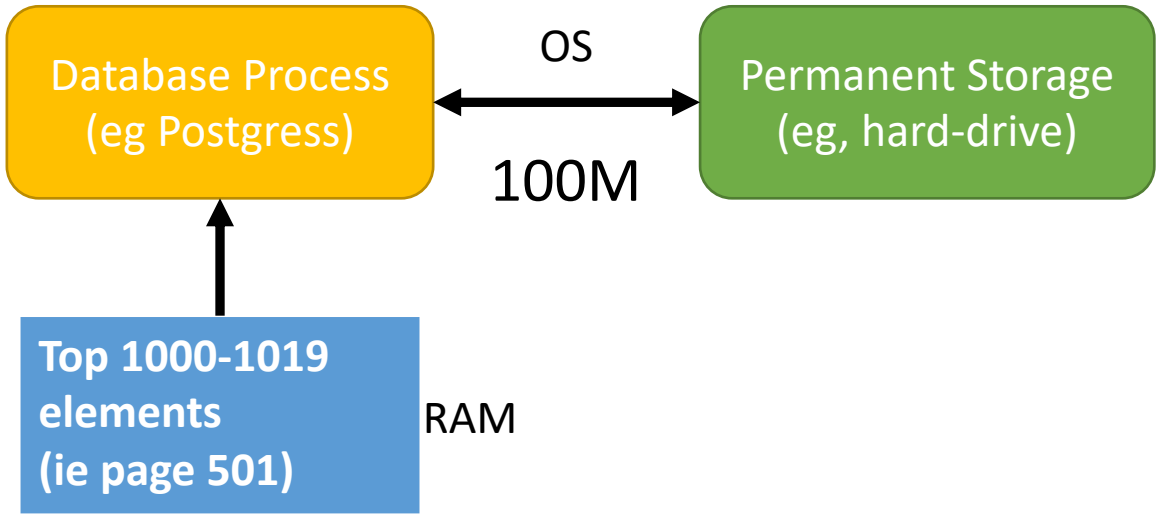
- Assume sorting by ID in descending order
- Page 1: *select * from News* ***ORDER BY*** *id desc* ***LIMIT*** *3*
- Page 2: *select * from News* ***ORDER BY*** *id desc* ***LIMIT*** *3* ***WHERE*** *id < 7*
  - 7 is the id of lowest item in previous page
- Page 3: *select * from News* ***ORDER BY*** *id desc* ***LIMIT*** *3* ***WHERE*** *id < 4*

Downside

- To read page *N*, we first must read the previous *N-1* pages, to know lowest id of page *N-1*
  - can’t jump directly to a specific page, but no problem if we iterate over them in order
- Need to keep track of last read item
  - all of its columns involved in the ORDER BY, more on this later…

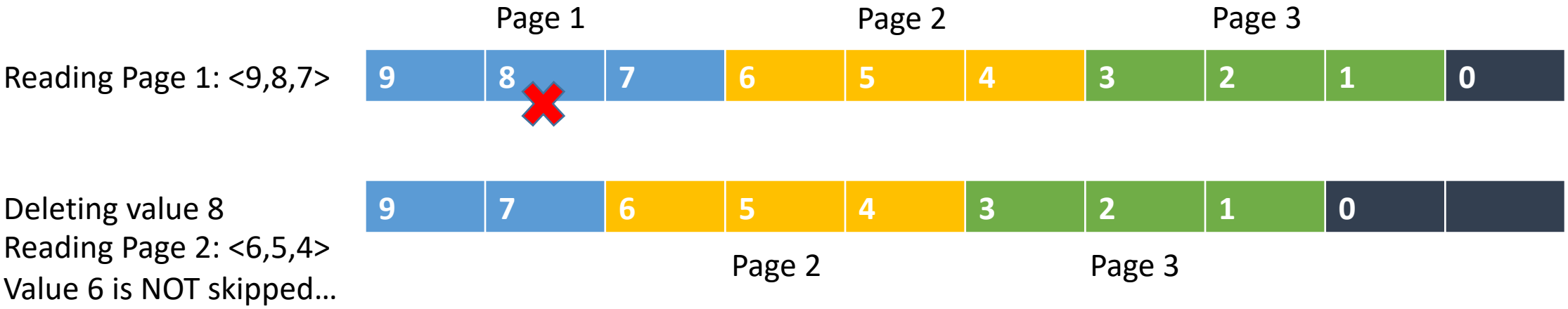
Benefit 1: Performance

- Example, N=20, reading Page 501
- Assume read all previous pages, where X is lowest id in Page 500
- Keep buffer B of only size 20
- No need to keep track of top 1-1000, as *"WHERE id < X"* takes care of it



Benefit 2: Data Consistency

- *Question*: what if, after reading Page 1, someone else does a DELETE of an element in first page?
- *Answer*: no problem!
- Eg, assume page size *N=3*, and someone removes id 8
- Due to "*WHERE id < 7*", the value 6 is not skipped when reading Page 2



Multi-Column Ordering

- There might be different ways to order data
  - eg, posts in a discussion forum sorted in chronological order
- Ordering MUST be *deterministic* and *strict*
  - no 2 elements must be equal for the ordering relations
  - otherwise, multiple SELECTs on different pages could give inconsistent results if the actual returned values are in different order when equivalent
- Given some properties (eg Time) which do not guarantee strictness (ie 2 or more elements with same value), can sort by a unique second parameter (eg, the primary key) when ties

Example

- Books sorted only by YEAR in the ORDER BY
- Asking for first page twice could give 2 totally different sets
  - reading 2nd page then could lead to re-read some rows while skipping some others
- *SELECT * from Book order by YEAR desc limit 3*

YEAR	ID
2019	3
2019	5
2019	1
2019	7
1969	9

SELECT

YEAR	ID
2019	1
2019	5
2019	3

SELECT

YEAR	ID
2019	5
2019	7
2019	1

Fixing Ordering

- If ordering is strict, then 2 SELECTs will have exactly same result
- *SELECT * from Book order by YEAR desc, ID desc limit 3*

YEAR	ID
2019	3
2019	5
2019	1
2019	7
1969	9

YEAR	ID
2019	7
2019	5
2019	3

What About 2nd Page?

- IF WHERE only based on YEAR, we could lose data when iterating
- WHERE must be based on all columns in the ORDER BY
- *SELECT * from Book order by YEAR desc, ID desc limit 3* ***where YEAR < 2019***
- The row <2019,1> would be wrongly skipped

YEAR	ID
2019	3
2019	5
2019	1
2019	7
1969	9

SELECT

YEAR	ID
1969	9

Fix: Handle Matches in WHERE

- Last element in first page: <2019,3>
- Page 2: *SELECT * from Book order by YEAR desc, ID desc limit 3 where **YEAR < 2019 or (YEAR = 2019 and ID < 3)***

YEAR	ID
2019	3
2019	5
2019	1
2019	7
1969	9

YEAR	ID
2019	1
1969	9

JSON Marshalling

- In dto, can simply have a "*next*" link URL
- Need 2 query parameters to keep track of last item in current page

dto = {

"**list**": […], *//the actual payload*

*"**next**": "/books?**keysetId**=3&**keysetYear**=2019"*

}

Data Expansion

Expansion

- A "*news*" might have a *list* of "*comments*"
- A "*news*" might also have a *list* of "*users*" that liked it
- When retrieving a single item, might not want to download as well the hundreds/thousands of other items related to it
- As returning those lists can be very expensive, can have special query parameters to choose if downloaded or not
- Eg.: *GET /news?**expand**=NONE* (no lists)
- Eg.: *GET /news?**expand**=COMMENTS* (include comments)

Tradeoff

- Option 1: never return those lists
  - But, then, need further HTTP calls to retrieve those lists if needed
- Option 2: create "*expand*" query parameters to control what returned
  - Good: can return everything needed in a single HTTP request
  - Bad: needs to implement all the needed cases *manually*
- *GraphQL*: a selling point compared to REST is its ability to exactly specify what to return
  - we will see GraphQL later in the course

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **advanced/rest/wrapper**
- **advanced/rest/rest-dto**
- **advanced/rest/exception-handling**
- **advanced/rest/rest-exception**
- **advanced/rest/pagination-offset**
- **advanced/rest/pagination-keyset-gui-v2**
- Study relevant sections in *RESTful Service Best Practices*
- Study relevant sections in RFC-7230 and RFC-7231