Performance and Query Processing

Same Query, Same Data, Two Runs

- demo

Logical vs Physical

- so far we've focused on the logical view of the database
  - tables, columns, constraints, SQL
- now we look at the physical level
  - how tables are stored on disk
  - how queries are run
- physical data independence means that how things are managed at the physical level changes speed, not results
- our focus is on the tools a DBA can use to improve performance

Road Map

- storage and indexes
  - how tables are stored on disk
  - types of indexes
- indexes in MySQL
  - creating indexes and choosing which columns to index
- from SQL to query trees
  - rewriting a query into an equivalent but faster form
- execution plans and the optimizer
  - join algorithms, statistics, and how MySQL picks a plan
- tuning and denormalization
  - choosing indexes for a workload
  - rewriting queries
  - when to bend normalization rules

Records and Pages

- each row is stored as a *record*
- records are grouped into *pages* (also called *blocks*)
  - page is the DBMS' unit of transfer between disk and memory
  - InnoDB (MySQL) pages are 16KB
- a table is stored as a collection of pages

Measuring Cost

- the cost of a query is measured in the number of pages accessed (read and written)
  - assumes every page must go to and from disk – no caching
  - CPU time is not considered
- this is a simplification, but it is still useful for comparing query plans
  - produces the right answer most of the time
  - predictable – what is cached depends on what else has run recently
  - disk time dominates

	access time
memory	~0.0001 ms
SSD	~0.1 ms
HDD	~5-10ms

- real optimizers do take some steps to account for caching

Our Example

- library
  - LOANS = 1,000,000 rows
  - 100 rows per page
  - 10,000 pages

LOANS

field	type	bytes
loan_id	INT UNSIGNED	4
book_id	INT UNSIGNED	4
member_id	INT UNSIGNED	4
checked_out_on	DATE	3
due_on	DATE	3
returned_on	DATE NULL	3 (0 if NULL)

record size (min–max)	**18–21 bytes**
average record (actual data) returned_on almost never NULL	**≈ 21 bytes**
records per page	**16,384 / 21 ≈ 780**
actual records per page measured in MySQL after loading	**≈ 367** ≈ 45 bytes per record

- record size alone doesn't give the full picture of records per page
  - per-record overhead (e.g. transaction ID, rollback pointer)
  - partly empty pages – space left deliberately open for future inserts
  - data structure overhead (B+ tree)

Unordered (Heap) Files

- records are stored wherever there's space, in no particular order
- insert
  - cheap – put wherever there is space, e.g. at the end
- search
  - expensive – have to check every page (*full table scan*)
- `SELECT … FROM LOANS WHERE member_id = 42`
  - ~10,000 pages
  - is this bad?
    - on an SSD, 10,000 reads takes ~1 second
    - multiply by thousands of users

Sorted Files

- records are stored in sorted order
- search
  - binary search – fast
    - binary search on 10,000 pages requires $\log_2(10{,}000)$ = ~14 reads
- insert
  - binary search to find the right page for insertion
  - insert is fast if there's room on that page, potentially not so fast otherwise
- but records can only be sorted on one thing
  - if sorted by member_id, searching for book_id = 100 still requires a full table scan