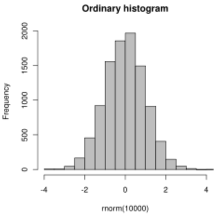
Indexes, Query Optimization

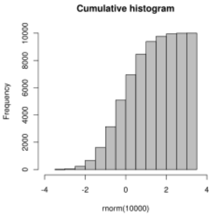
PDBM 7.5, 12.3.5

Dr. Chris Mayfield

Department of Computer Science
James Madison University

Feb 17, 2022





Part 1: SQL Indexes

How data is stored

Heap File
(table: `movie`)

RID	id	title	year
1	282375	Inception	2010
2	293178	Jane Eyre	2011
3	246720	Harry Potter	2010
4	57126	Avatar	2009
5	662387	TRON: Legacy	2010
...	...	...	...

Index File
(column: `movie.year`)

year	RID
2009	4
2010	1
2010	3
2010	5
2011	2
...	...

Does this query have to look at every movie?

```
SELECT * FROM movie
WHERE year >= 2011;
```

Creating indexes

Single column

```
CREATE INDEX ON movie (title);
CREATE INDEX ON movie (year);
```

Multiple columns

```
CREATE INDEX ON movie (title, year);
CREATE INDEX ON movie (year, title);
```

These are just examples; don't do all of them!

A LOT happens behind the scenes:

- http://en.wikipedia.org/wiki/B-tree

Selection of indexes

Why not just index every column?

- They take up disk space
- They take time to build
- Expensive to maintain

Deciding which columns to index:

- PRIMARY KEYs (automatic)
- UNIQUE attributes (automatic)
- Attributes in WHERE clauses

Example

StarsIn(movieTitle, movieYear, starName)

What should be indexed?

```
-- Query #1
SELECT movieTitle, movieYear FROM StarsIn
WHERE starName = s;

-- Query #2
SELECT starName FROM StarsIn
WHERE movieTitle = t AND movieYear = y;

-- Query #3
INSERT INTO StarsIn VALUES (t, y, s);
```

Part 2: Query Optimization

Evaluating query plans

In psql, type `EXPLAIN` in front of any query

In pgAdmin, press F7 instead of F5

- Shows graphical version of `EXPLAIN` output

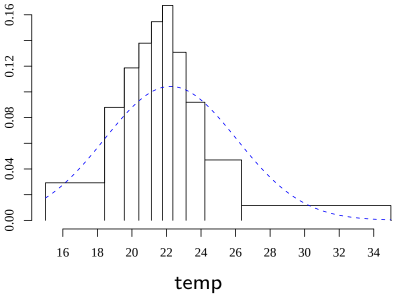
Demo: analyzing HW4 query performance

- If possible, avoid sequential scans
- Where should you create indexes?

Selectivity estimation

DBs maintain statistics for each relation / attribute:

- Histograms (if numeric)
- Most common values
- % NULL attributes
- Average size (in bytes)
- Physical correlation
- ...



```
SELECT * FROM sensor WHERE temp > 25; -- Index Scan

SELECT * FROM sensor WHERE temp < 25; -- Seq Scan
```

Summary of indexes

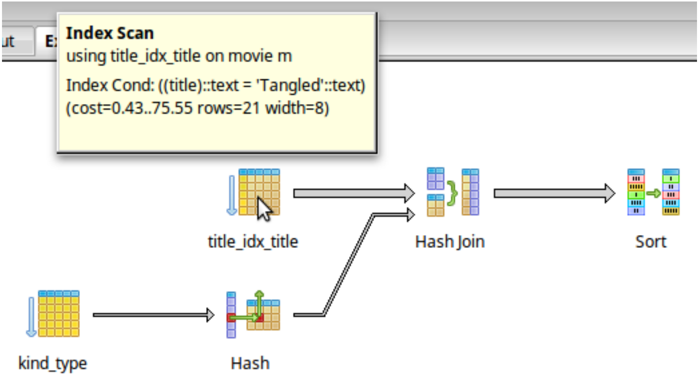
Data *distribution* affects plan choice

- For example, query 10,000 rows
- `SELECT * FROM t WHERE a = 1;`
  - [Plan A] when 90% have a = 1
  - [Plan B] when a = 1..10000, 1 time each
  - [Plan C] when a = 1..10, 1000 times each

Performance tips

- Rebuild optimizer statistics after updates
- Use `EXPLAIN ANALYZE` to profile your queries
- Be aware of which indexes are being used

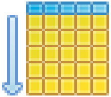
pgAdmin demo



Index Scan
using title_idx_title on movie m
Index Cond: ((title)::text = 'Tangled'::text)
(cost=0.43..75.55 rows=21 width=8)

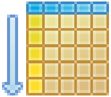
- `EXPLAIN` = show estimated cost/rows
- `EXPLAIN ANALYZE` = show actual time/rows

Basic scans



Sequential Scan

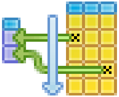
Most basic physical operation that reads an entire table from beginning to end.



Index Scan

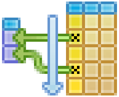
Scans a table using an (unclustered) index, most often with search criteria or other stopping conditions.

Bitmap scans



Bitmap Heap Scan

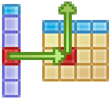
Combines the output of multiple index scans by constructing a bitmap, and then reads the resulting table rows in physical order.



Bitmap Index Scan

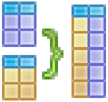
Similar to a bitmap heap scan, but uses an additional index scan at the end.

Hash joins



Hash

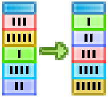
Constructs a temporary hash table over the given rows.



Hash Join

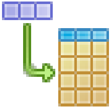
Constructs a hash of the inner table, scans the outer table sequentially, and joins rows via hash lookups.

Merge joins



SORT

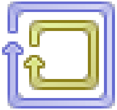
Orders the given rows by one or more values (using *external sorting*).



MERGE JOIN

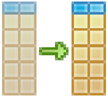
Used when both tables are sorted; scans each table simultaneously and merges any matching rows.

Other joins



Nested Loops

Most basic join technique: for each row in the outer table, compare with each row in the inner table.



Materialize

Saves the current query results to memory as a new (but temporary) table.

Aggregation

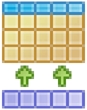
GROUP

Organizes rows into groups by sorting or hashing their values.

AGGREGATE

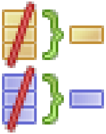
Combines each group of rows into a single row by applying a function.

Miscellaneous



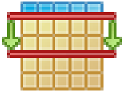
Append

Adds additional rows to the current result (e.g., `UNION ALL` queries).



Unique

Removes any duplicate rows (using sorting or hashing).



Limit

Returns only the top k rows. Often changes the entire plan.