Google

Indexes for file-based tables in F1 Query

panahi@

03/19/2023

About me

- I have survived Wisconsin winters:
  - Bachelors, Masters, PhD at University of Wisconsin-Madison
- Phd Thesis: Human-Centric Debugging of Entity Matching
  - Jeff Naughton and AnHai Doan
- Google, since August, 2016:
  - Storage and query execution for F1 Query and BigQuery:
    - F1 Query: Making queries fast using indexes on file-based tables

Google

Background: F1 Qurey*

- F1 Query is a federated query processing platform
    - Executes SQL queries against data stored in:
        - File-based tables
        - Storage systems (ex. Spanner)
- F1 Query is used for critical applications including **Advertising**, **Shopping**, **Analytics**, and **Payments**.

* Samwel, B., Cieslewicz, J., Handy, B., Govig, J., Venetis, P., Yang, C., Peters, K., Shute, J., Tenedorio, D., & Apte, H. (2018). F1 Query: Declarative Querying at Scale. Proceedings of the VLDB Endowment, 11(12), 1835-1848.

Google

Problem space: Schema

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

Problem space: File-based table

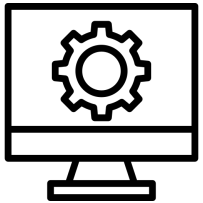


Table t

03/12/2023

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

03/11/2023

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

03/10/2023

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

F1 Query

```
SELECT user_id FROM t
WHERE 10 IN experiment_id
```

Constraint: Immutable write-once read-many files

Problem space: User-managed tables

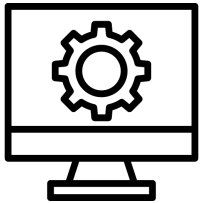


Table t

03/12/2023

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

03/11/2023

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

03/10/2023

user_id	Experiment id
111	[1, 10, 20]
222	[10, 25, 50]
333	[15, 20, 50]

```
SELECT user_id FROM t
WHERE 10 IN experiment_id
```

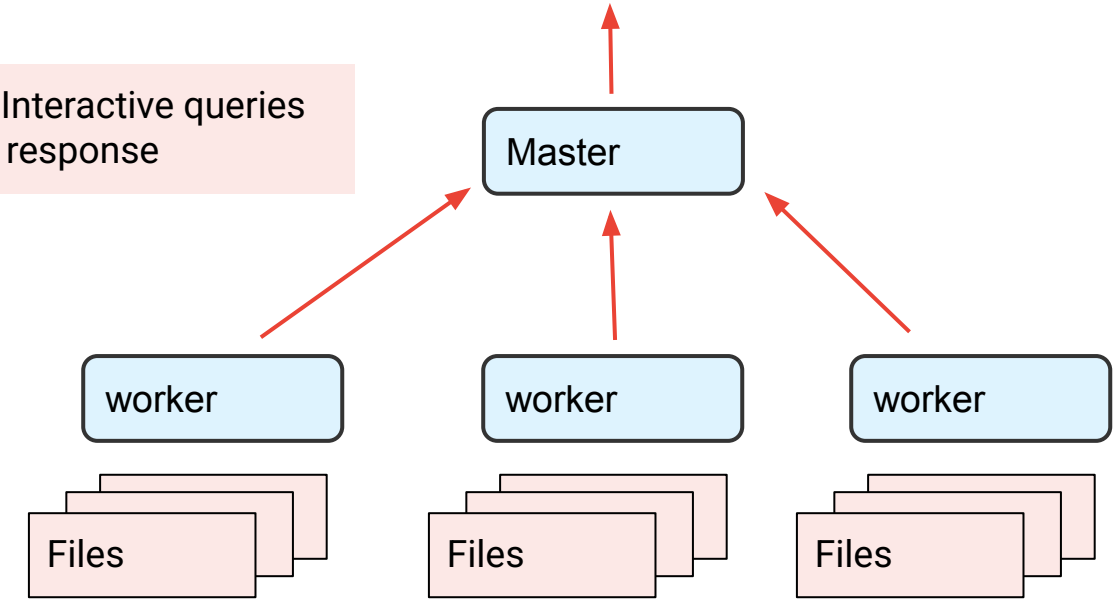
F1 Query

Constraint: User can change files without F1 knowing about it

Problem space: Distributed query processing

```
SELECT user_id FROM t WHERE 10 IN experiment_id
```

Constraint: Interactive queries require fast response



Hundreds of workers

100k files per day
PBs of data

Problem space: Evaluating the filter

15 IN experiment_id?

worker

user_id	Experiment id	
111	[1, 10, 20]	✗
222	[10, 25, 50]	✗
333	[15, 20, 50]	✓

Solution: Indexes

- **Key Idea:** Skip the full scan and skip the non-matching rows
- When is an index useful:
  - Only a few rows match
  - Expensive predicate (experiment_id analysis)

Google

Index format

Row number	Experiment id
1	[1, 10, 20]
2	[10, 25, 50]
3	[15, 20, 50]

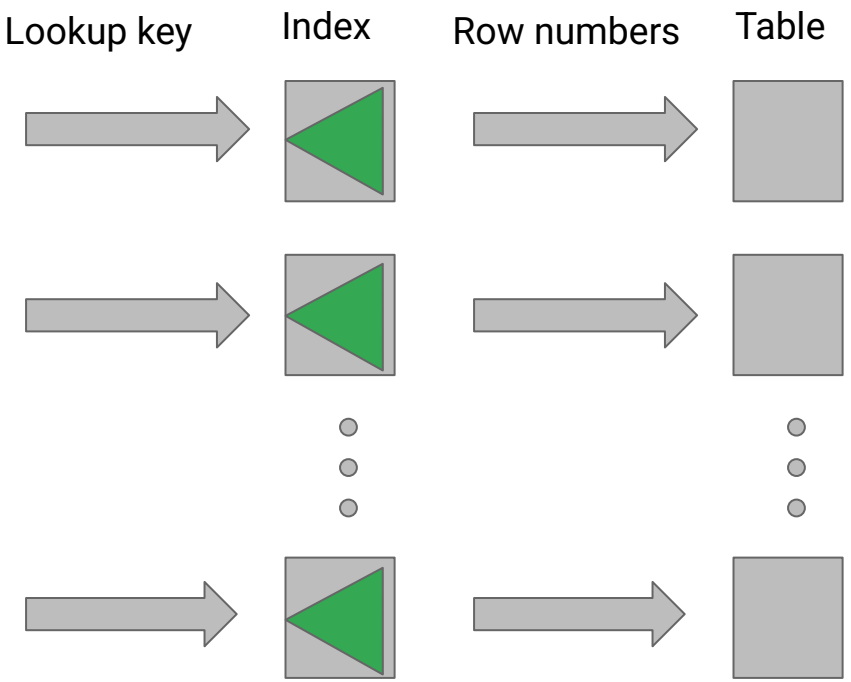
Experiment id	Row numbers
1	[1]
10	[1, 2]
15	[3]
20	[1, 3]
25	[2]
50	[2, 3]

B-Tree

- B-Tree is a self-balancing search tree
- Allows fast point and range lookups (Geo-spatial queries)
- The B-Tree size depends on data distribution:
  - In practice, ended up about 20% of the input file size
- Space saving strategies:
  - Each BTree page is zstd compressed
  - Roaring bitmaps for storing row numbers are space efficient and compressed
    - 1 bit - 2 bytes

Google

Version 1: Per-file index



Version 1: F1 syntax

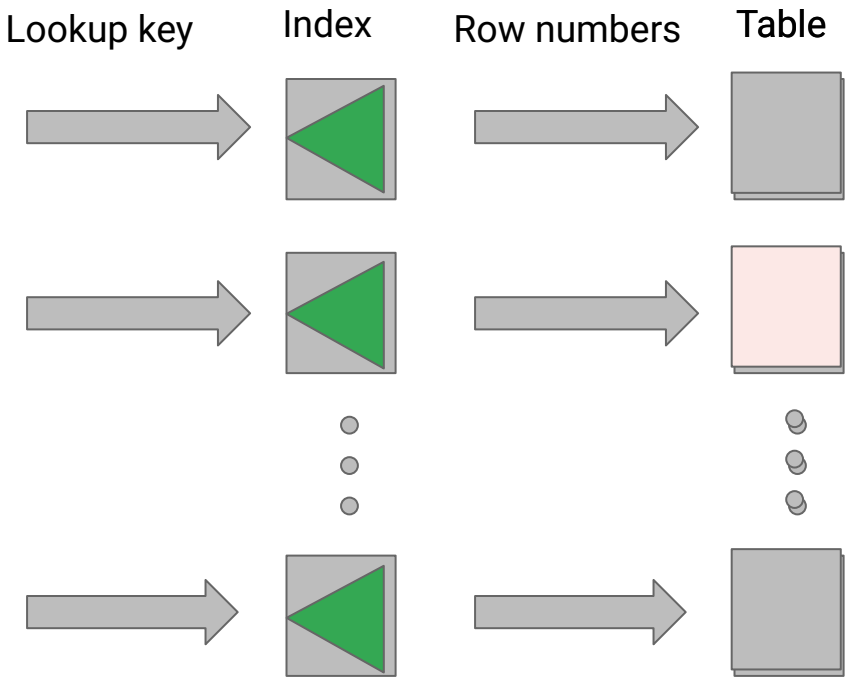
```
DEFINE TABLE t (format = 'capacitor', path = '<file_path>');

CREATE INDEX ON t (key_column);
```

Creates index and Registers metadata

```
SELECT * FROM t WHERE key_column = 10;
```


Google

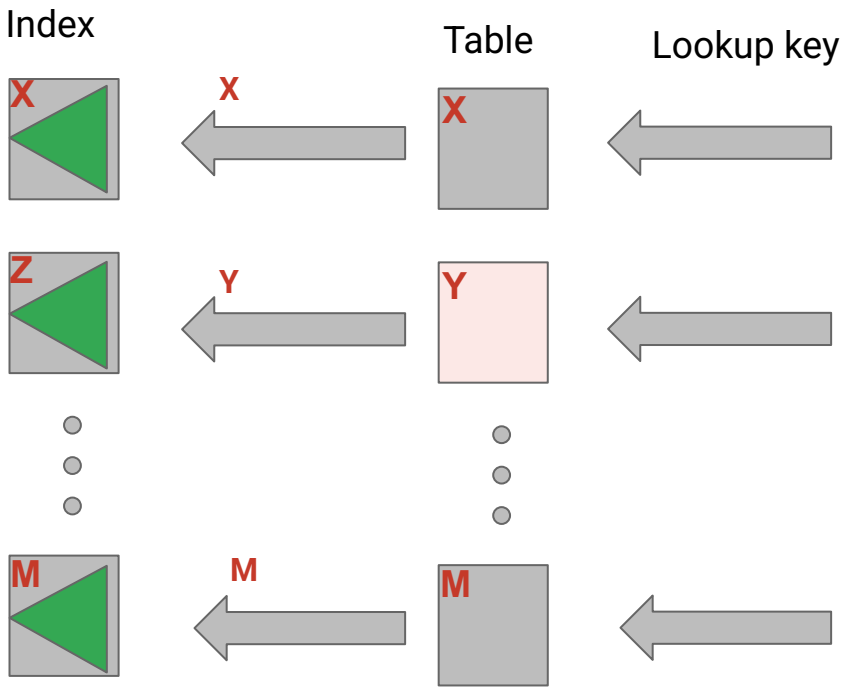
Synchronization issue



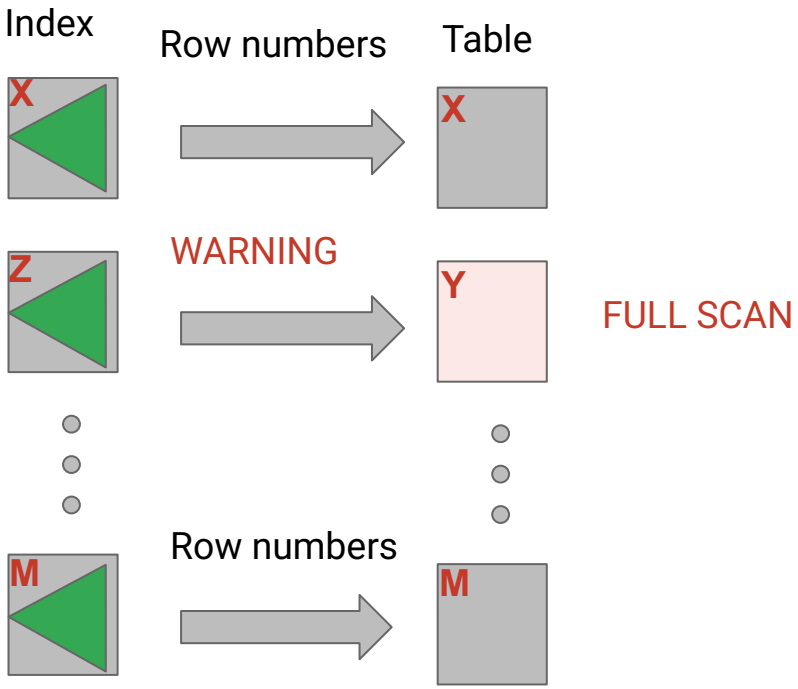
Incorrect Results

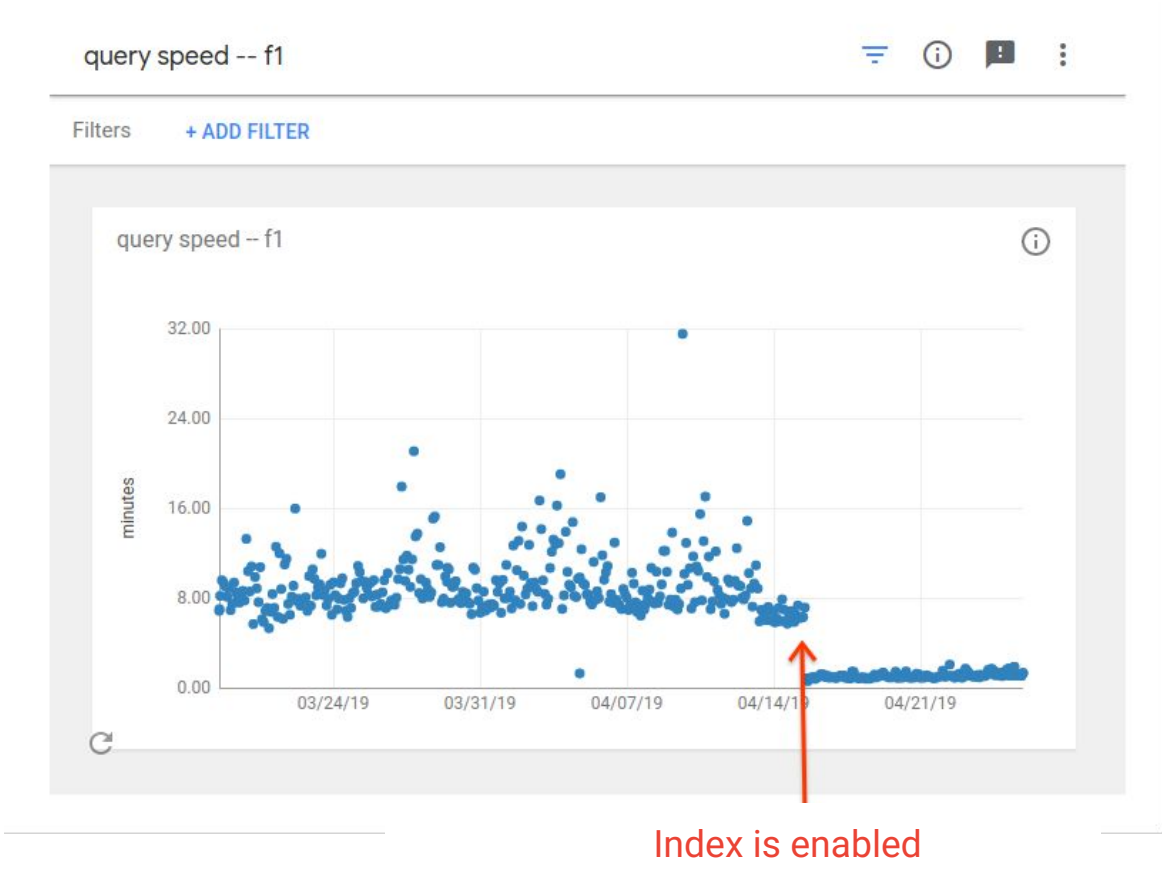
Constraint: We are not going to expect the user to be careful to keep things in sync.

Version 2: Base-driven lookup

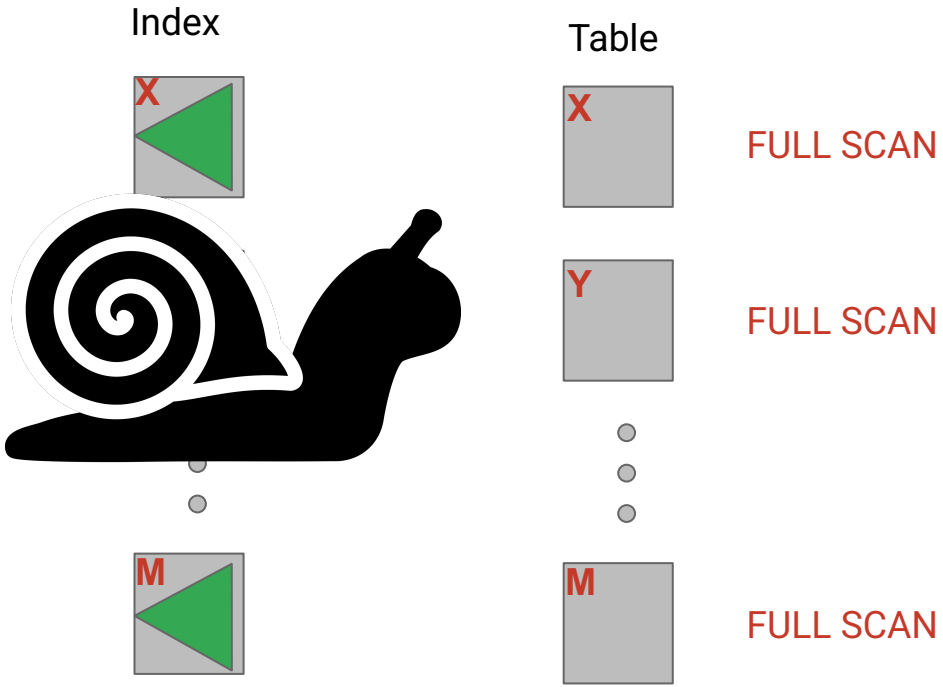


Version 2: Base-driven lookup





Issue: Index availability

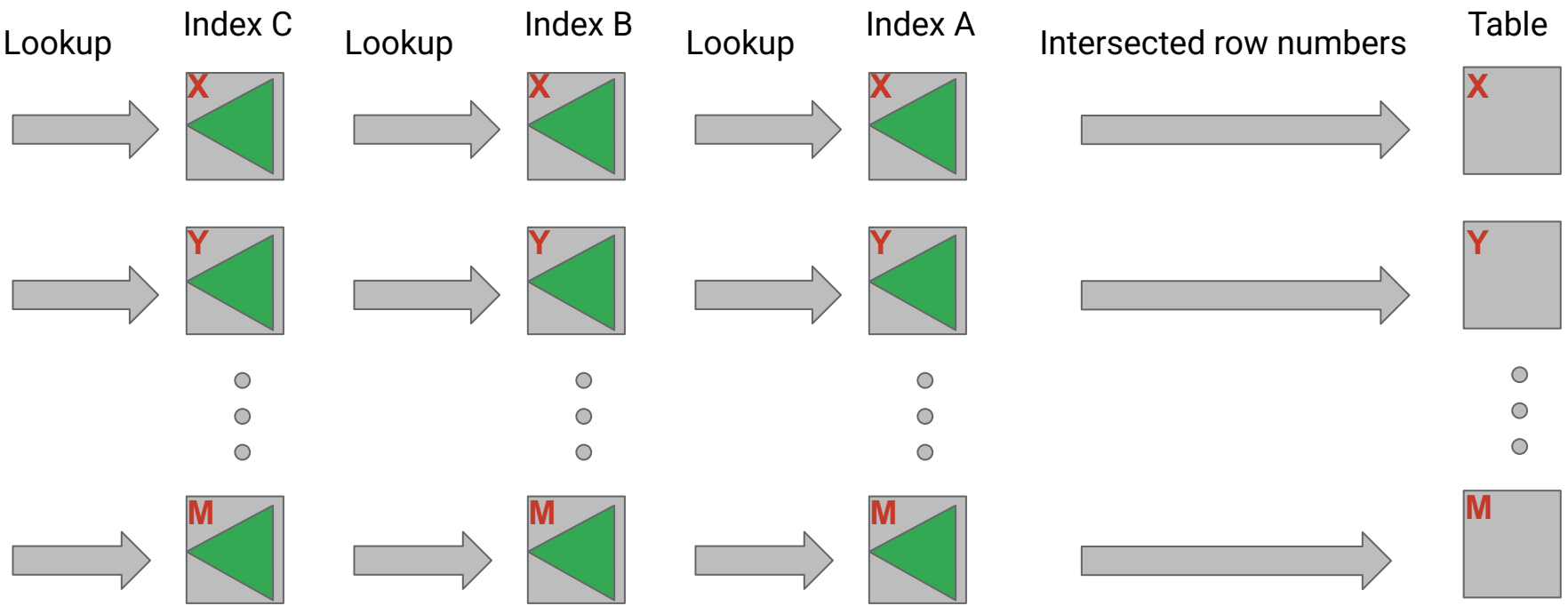


Problem: Multiple indexes

```
SELECT user_id FROM t WHERE A = 10 AND B = 20 AND C = 30
```


Google

Issue: Multiple indexes



Version 3: In-file indexes

- **Key idea:** Put the index inside the file itself
- **Convenience:**
  - Simple to understand
  - Easy to adopt
- Just name the indexes you want when creating the file
- No synchronization issue
- No delay between index and file generation
- Easily supports multiple indexes and index intersection

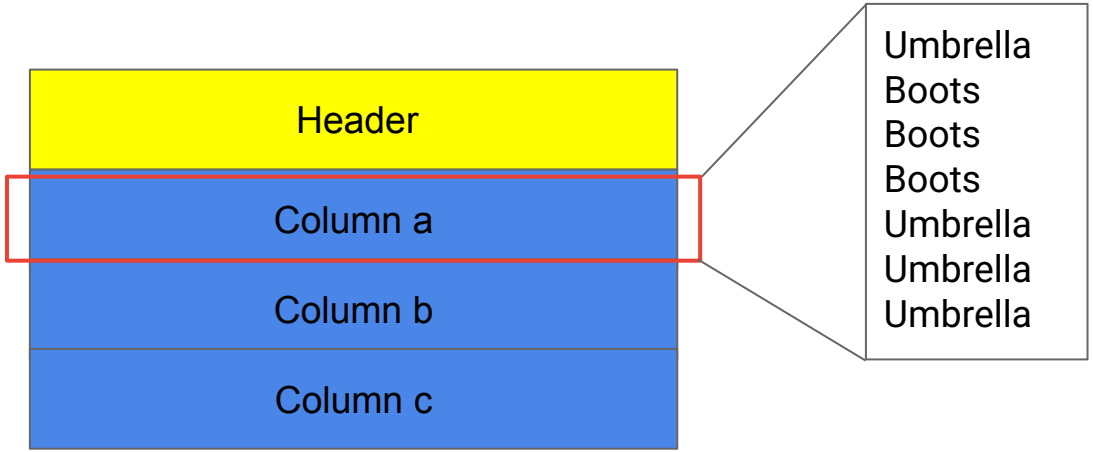
Background: Capacitor*

- Capacitor
  - Internal file format for BigQuery
  - Used by a wide array of users inside Google other than BigQuery

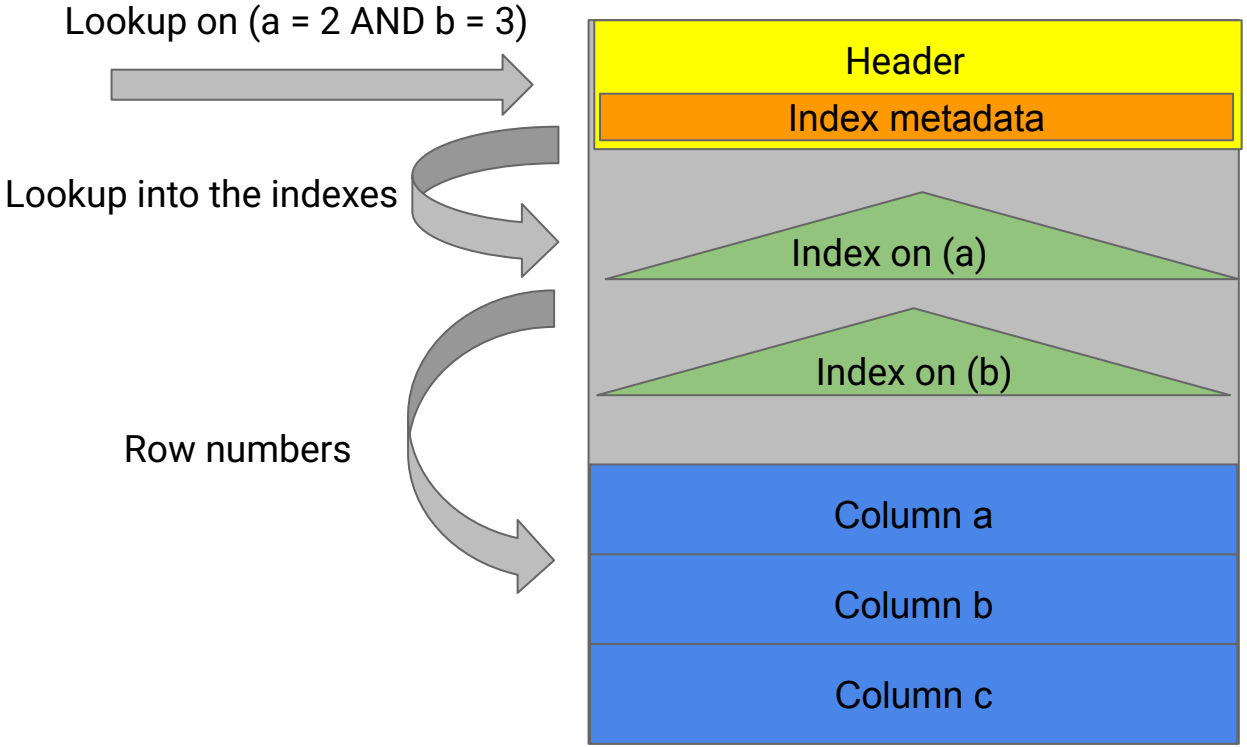
* M. Pasumansky. Inside Capacitor, BigQuery's Next-Generation Columnar Storage Format. Google Cloud Blog, Apr 2016.

Background: Capacitor

```
SELECT a FROM t;
```



Index lookup



Index format

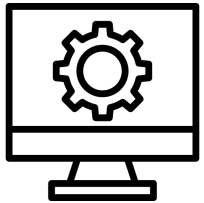
Row number	Experiment id
1	[1, 10, 20]
2	[10, 25, 50]
3	[15, 20, 50]

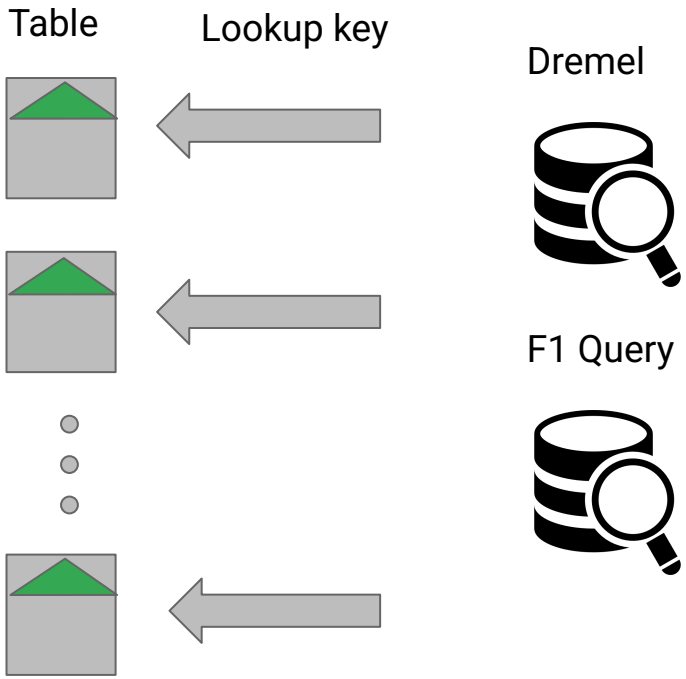
The same!

Experiment id	Row numbers
	[1]
	[1, 2]
15	[3]
20	[1, 3]
25	[2]
50	[2, 3]

How it works

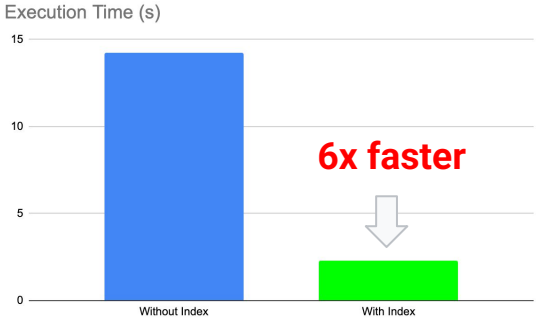
```
capacitor_writer.add_index(“experiment_id”)
```

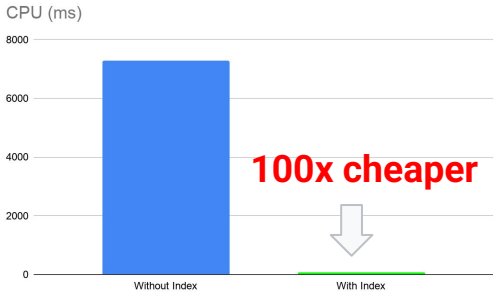


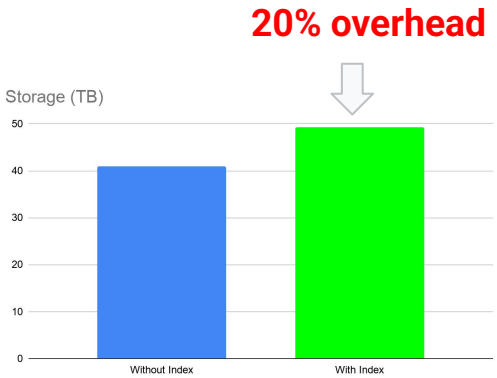


Example: Production data set

```
SELECT user_id FROM t WHERE 10 IN experiment_id
```

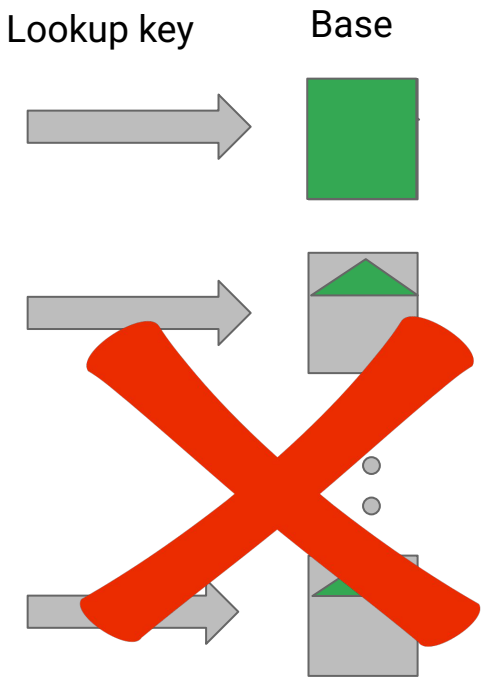




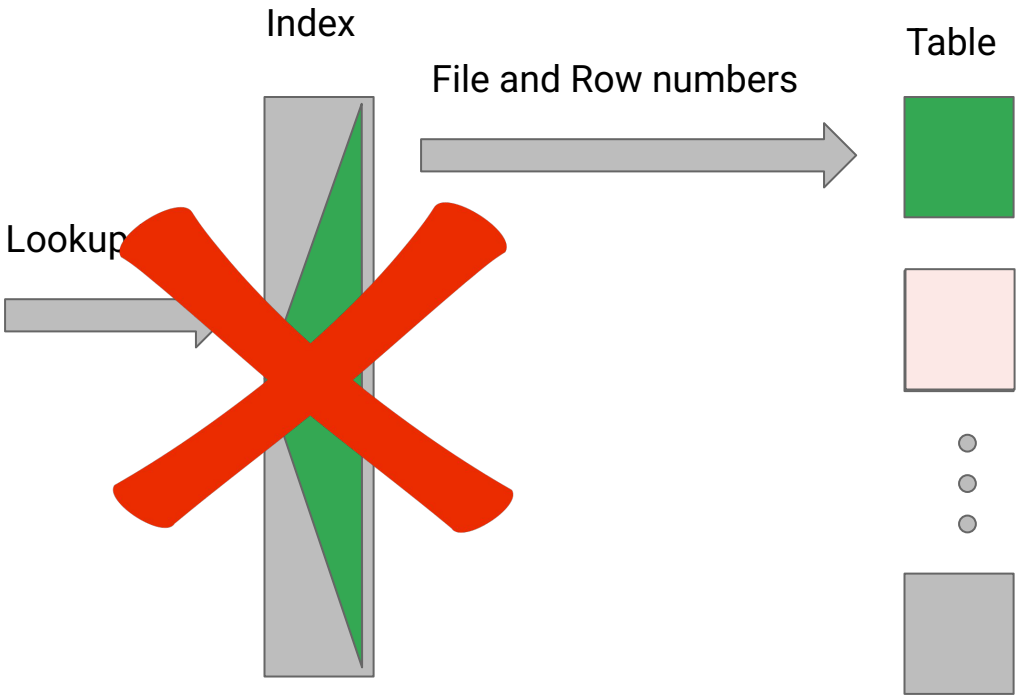


Customers started incorporating indexes into their data sets as soon as a couple of hours after launch with **one line code change**.

Issue: Opening too many files



Version 4: Global index



F1-managed tables

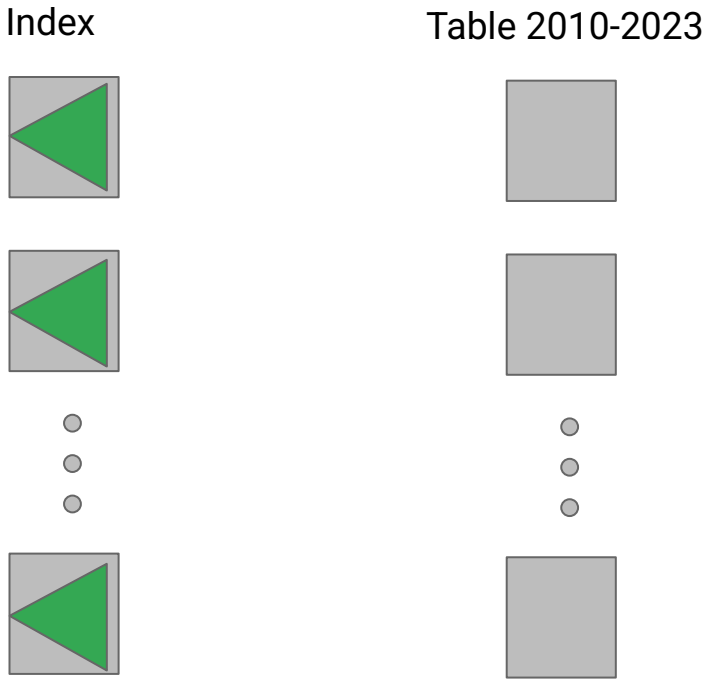
- **Key idea:** F1 manages the files and keeps the table and index in sync.

```
CREATE TABLE mytable OPTIONS
(format = 'capacitor', path = '<file_path>', indexes=["col1", "col2"]) AS ...
```

- F1 creates table and index files are created in a single atomic step
- Once table files are updated, F1 will automatically update the index files

Google

Dealing with legacy data



Separate-file vs. In-data-file indexes

Seperate-file (Version 1, 2, 4)	In-data-file indexes (Version 3)
Separate file	Inside the table file
Not file-format specific	Capacitor
Supports global indexes	Not applicable
Need to keep table and index file in-sync	No need for synchronization
Supports indexes over legacy data	Only applicable to newly generated files
Too many file opens for multiple indexes	Easily supports multiple indexes

Conclusions

- We talked about how to make queries run multiple orders of magnitude faster using:
  - Generic separate-file indexes
  - Capacitor-specific in-data-file indexes
- Discussed various workloads and constraints and how they lead us to multiple in-production solutions
- Stressed over the importance of:
  - Minimizing chance of errors from the users, always returning the correct result
  - Convenience of adopting a new feature
- **Hundreds of users/applications** in Google are using these indexes for **millions of queries** over **PBs of data** on a daily basis.