



Going Deep -How to Optimise Queries

Optimization: Necessary columns only

Original code

```
SELECT  
  *  
FROM  
  `dataset.table`
```

Optimized

```
SELECT  
  * EXCEPT (dim1, dim2)  
FROM  
  `dataset.table`
```

Don't do "select * limit 10" to look at data, use the **preview** feature instead.

Optimization: Auto-pruning with Partitioning & Clustering

Table info

Table ID	bigquery-public-data:wikipedia.pageviews_2021
Table size	1.1 TB
Long-term storage size	497.28 GB
Number of rows	24,870,308,515
Created	Dec 31, 2020, 6:00:43 PM UTC-8
Last modified	Jun 11, 2021, 7:02:21 AM UTC-7
Table expiration	NEVER
Data location	US
Description	Wikipedia pageviews from http://dumps.wikimedia.you

Table Type	Partitioned
Partitioned by	DAY
Partitioned on field	datehour
Partition expiration	
Partition filter	Required

Clustered by	wiki
	title

Original code

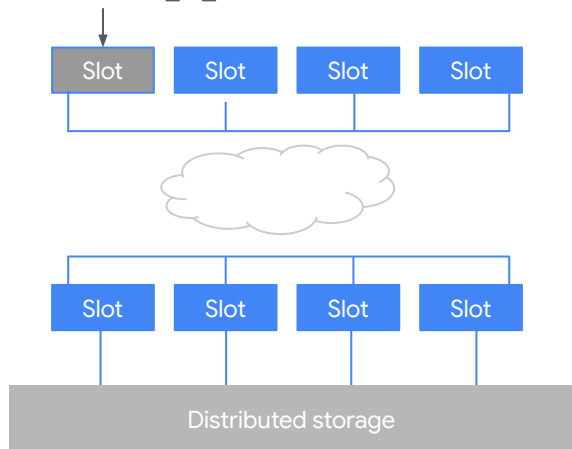
```
SELECT
...
FROM
`wikipedia.pageviews_2021`
```

Optimized

```
SELECT
...
FROM
`wikipedia.pageviews_2021`
WHERE
  DATE(datehour) = "2021-06-11"
  AND wiki="simple"
```

```
SELECT title, SUM(views)
FROM `wikipedia.pageviews_2021`
WHERE DATE(datehour) = "2021-06-11"
GROUP BY title
ORDER BY SUM(views) DESC
LIMIT 1000
```

All partial aggregations for
the title "Games_in_2020"



Shuffle - (bucketing using a hash function)

Stage 1: Partial
GROUP BY

```
SELECT title, SUM(views)
FROM `wikipedia.pageviews_2021`
WHERE DATE(datehour) = "2021-06-11"
GROUP BY title
ORDER BY SUM(views) DESC
LIMIT 1000
```



		Worker timing ⓘ					
Stages		Wait	Read	Compute	Write		Rows
✓ S00: Input ^	Avg:	<div><div></div></div> 236 ms	<div><div></div></div> 55 ms	<div><div></div></div> 723 ms	<div><div></div></div> 2 ms	Input:	81,888,146
	Max:	<div><div></div></div> 301 ms	<div><div></div></div> 205 ms	<div><div></div></div> 1571 ms	<div><div></div></div> 7 ms	Output:	2,635
READ	\$2:title, \$3:views, \$1:datehour FROM bigquery-public-data.wikipedia.pageviews_2021 WHERE and(equal(date(\$1), 18789), like(\$2, '%Deaths%'))						
AGGREGATE	GROUP BY \$30 := \$2 \$20 := SUM(\$3)						
WRITE	\$30, \$20 TO _stage00_output BY HASH(\$30)						
✓ S01: Sort+ v	Avg:	<div><div></div></div> 7 ms	<div><div></div></div> 0 ms	<div><div></div></div> 5 ms	<div><div></div></div> 1 ms	Input:	2,635
	Max:	<div><div></div></div> 8 ms	<div><div></div></div> 0 ms	<div><div></div></div> 7 ms	<div><div></div></div> 1 ms	Output:	868
✓ S02: Output v	Avg:	<div><div></div></div> 8 ms	<div><div></div></div> 0 ms	<div><div></div></div> 10 ms	<div><div></div></div> 10 ms	Input:	868
	Max:	<div><div></div></div> 8 ms	<div><div></div></div> 0 ms	<div><div></div></div> 10 ms	<div><div></div></div> 10 ms	Output:	868

Optimization: Late aggregation

Original code

```
SELECT
  t1.dim1,
  SUM(t1.m1)
  SUM(t2.m2)
FROM (SELECT
  dim1,
  SUM(metric1) m1
FROM `dataset.table1` GROUP BY 1) t1
JOIN (SELECT
  dim1,
  SUM(metric2) m2
FROM `dataset.table2` GROUP BY 1) t2
ON t1.dim1 = t2.dim1
GROUP BY 1;
```

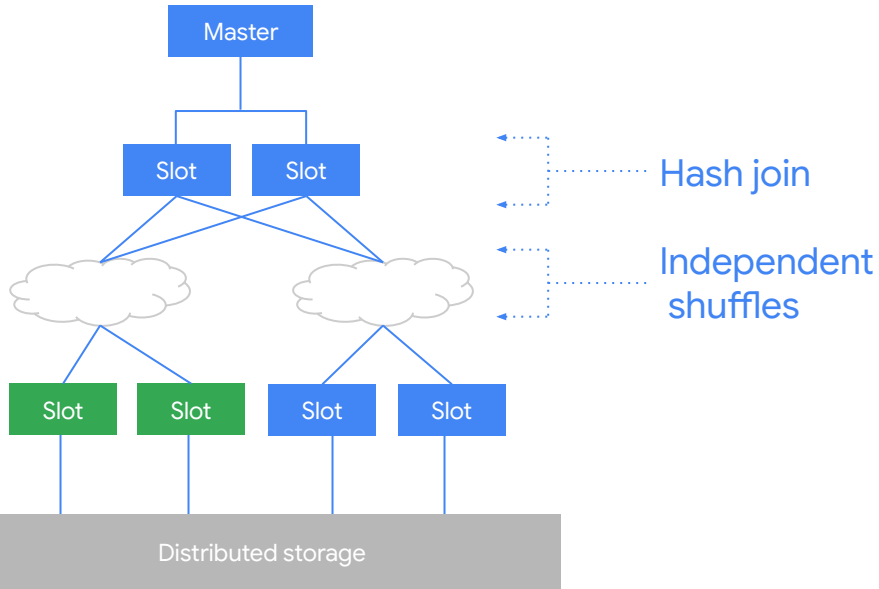
Optimized

```
SELECT
  t1.dim1,
  SUM(t1.m1)
  SUM(t2.m2)
FROM (SELECT
  dim1,
  metric1 m1
FROM `dataset.table1`) t1
JOIN (SELECT
  dim1,
  metric2 m2
FROM `dataset.table2`) t2
ON t1.dim1 = t2.dim1
GROUP BY 1;
```

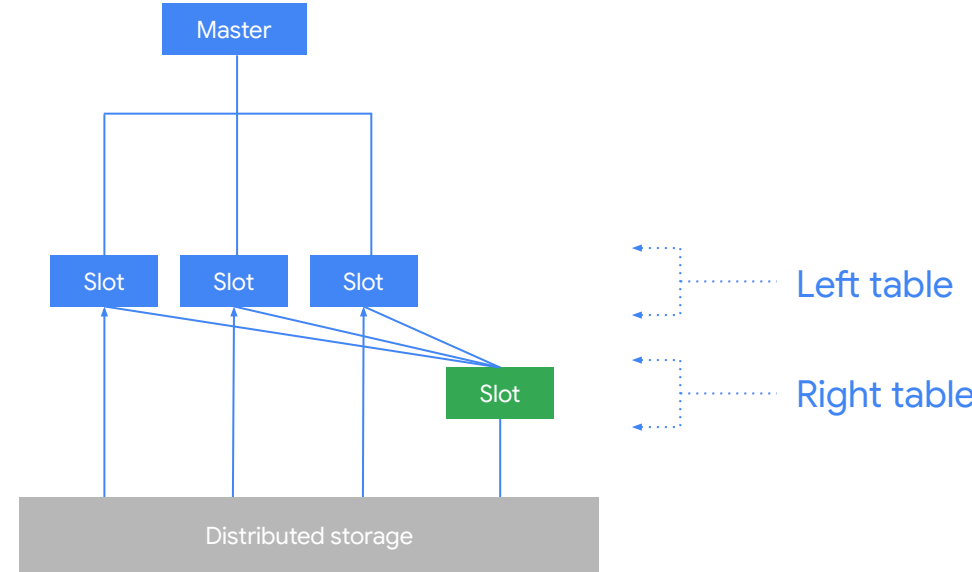
Avoid multiple group by executions, by moving them up.

Unless If a table can be reduced drastically by aggregating in preparation for being joined, then aggregate it early.

Large JOIN (shuffle)



Small JOIN (broadcast)



Optimization: JOIN pattern - largest table first

Original code

```
SELECT
  t1.dim1,
  SUM(t1.metric1),
  SUM(t2.metric2)
FROM
  `dataset.small_table` t1
JOIN
  `dataset.large_table` t2
ON
  t1.dim1 = t2.dim1
WHERE t1.dim1 = 'abc'
GROUP BY 1;
```

Optimized

```
SELECT
  t1.dim1,
  SUM(t1.metric1),
  SUM(t2.metric2)
FROM
  `dataset.large_table` t2
JOIN
  `dataset.small_table` t1
ON
  t1.dim1 = t2.dim1
WHERE t1.dim1 = 'abc'
GROUP BY 1;
```

Optimization: Filter before JOINS

Filter both tables before the join to reduce amount of data

Original code

```
SELECT
  t1.dim1,
  SUM(t1.metric1)
FROM
  `dataset.table1` t1
LEFT JOIN
  `dataset.table2` t2
ON
  t1.dim1 = t2.dim1
WHERE t2.dim2 = 'abc'
GROUP BY 1;
```

Optimized

```
SELECT
  t1.dim1,
  SUM(t1.metric1)
FROM
  `dataset.table1` t1
LEFT JOIN
  `dataset.table2` t2
ON
  t1.dim1 = t2.dim1
WHERE t2.dim2 = 'abc' AND t1.dim2 = 'abc'
GROUP BY 1;
```

WHERE clause: Expression order.

Original code

```
SELECT
  text
FROM
  `stackoverflow.comments`
WHERE
  text LIKE '%java%'
  AND user_display_name = 'anon'
```

Optimized

```
SELECT
  text
FROM
  `stackoverflow.comments`
WHERE
  user_display_name = 'anon'
  AND text LIKE '%java%'
```

The first part of your where clause should always contain the filter that will eliminate the most data.

Optimization: ORDER BY with LIMIT

Original code

```
SELECT
  t.dim1,
  t.dim2,
  t.metric1
FROM
  `dataset.table` t
ORDER BY t.metric1 DESC
```

Optimized

```
SELECT
  t.dim1,
  t.dim2,
  t.metric1
FROM
  `dataset.table` t
ORDER BY t.metric1 DESC
LIMIT 1000
```

