BigQuery STRUCT

TABLE OF CONTENTS

Overview

Objective

In an environment where BigQuery is used as a DB service,
proposing cost-saving measures
through the columnar storage features of BigQuery and its supported schemas.

BigQuery

Cloud service for storing and analyzing petabyte-scale data,
provided to external users by Google

Based on Dremel,
it features **Columnar Storage** and SQL tree architecture

BigQuery
Nested and Repeated Schema

BigQuery Nested and Repeated Schema

Denormalization

When BigQuery is denormalized, performance is optimized
Implementing Nested and Repeated to achieve denormalization

Advantages

Performance Improvement
Efficient data storage capacity
Flexible adaptation to schema changes

Columnar Storage

Storing tables at the denormalized column level to achieve storage efficiency

BigQuery Nested and Repeated Schema

Nested

BigQuery : RECORD

Container that have Every **Type(Required)** and field_name(Optional) sorted

STRUCT data structure

Similar to Dictionary(Python)**, but with fixed Key(Schema exists)**

→ **You can load data into a single column by bundling the data into one data struct**

Possible to nest up to depth of 15 levels

→ Same storage size does not matter of **Nest(Storage based Pricing)**

상품명	가격	수량
A	100	1
C	300	2
A	100	1
B	200	2
C	300	3

# of Column : 3

주문
상품명: 'A', 가격: 100, 수량: 1
상품명: 'C', 가격: 300, 수량: 2
상품명: 'A', 가격: 100, 수량: 1
상품명: 'B', 가격: 200, 수량: 2
상품명: 'C', 가격: 300, 수량: 3

# of Column : 1
Actual Form

주문		
주문.상품명	주문.가격	주문.수량
A	100	1
C	300	2
A	100	1
B	200	2
C	300	3

# of Column : 1
BigQuery Visualization

BigQuery Nested and Repeated Schema

Repeated

BigQuery : REPEATED

Array with same data type

Similar to Array(Python)

If ARRAY has any NULL value, it will Return NULL

→ IGNORE NULLS to avoid

BigQuery Visualization of Array is as follows

성	이름	연락처	나이	상품명
홍	은지	010-3333	30	A
홍	은지	010-3333	30	C
이	유나	010-4444	29	A
이	유나	010-4444	29	B
이	유나	010-4444	29	C

Original

성	이름	연락처	나이	상품명
홍	은지	010-3333	30	[A, C]
이	유나	010-4444	29	[A, B, C]

Actual Form

성	이름	연락처	나이	상품명
홍	은지	010-3333	30	A
				C
이	유나	010-4444	29	A
				B
				C

BigQuery Visualization

BigQuery Nested and Repeated Schema

Nested and Repeated

RECORD(STRUCT) + ARRAY

Denormalization using STRUCT with ARRAY

Apply array then use struct method

Original

성	이름	연락처	나이	상품명	가격	수량
홍	은지	010-3333	30	A	100	1
홍	은지	010-3333	30	C	300	2
이	유나	010-4444	29	A	100	1
이	유나	010-4444	29	B	200	2
이	유나	010-4444	29	C	300	3

Nested

성	이름	연락처	나이	주문		
				주문.상품명	주문.가격	주문.수량
홍	은지	010-3333	30	A	100	1
홍	은지	010-3333	30	C	300	2
이	유나	010-4444	29	A	100	1
이	유나	010-4444	29	B	200	2
이	유나	010-4444	29	C	300	3

Nested and Repeated

성	이름	연락처	나이	주문		
				주문.상품명	주문.가격	주문.수량
홍	은지	010-3333	30	A	100	1
				C	300	2
이	유나	010-4444	29	A	100	1
				B	200	2
				C	300	3

- Efficient Data Storage
- Form of GROUP BY

Effect of Use

Effect of Use

Test Data

DB data accumulated over 3 days
Duration :

Comparison

Applying a new schema to a total of 80 tables
22 Tables stay as current
37 Tables applying single STRUCT to achieve capacity efficiency
21 Tables applying several STRUCT to achieve capacity efficiency and table unification

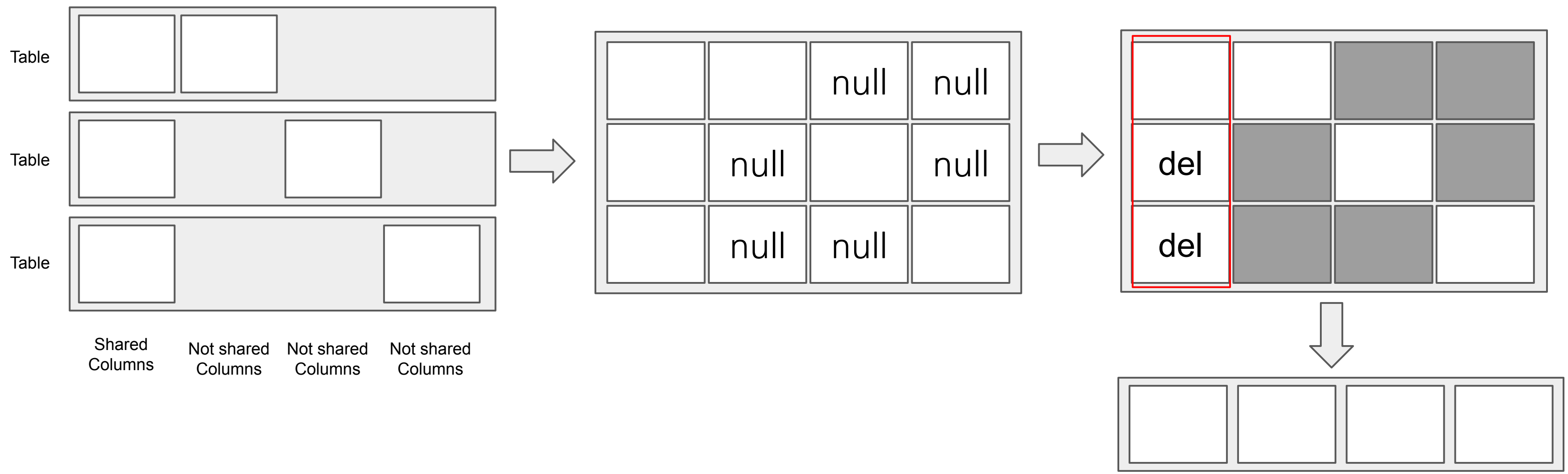
Unification	4 → 1	3 → 1	2 → 1
# of result Table	12 → 3	3 → 1	6 → 3

Total 66 Tables are created for capacity comparison

Effect of Use

Table Unification using STRUCT

- Table Unification based on Shared Columns
  - Make STRUCT with Not Shared Columns
- Null Value does not take up space, because of columnar storage
- Storage efficiency and table unification both can be achieved



Effect of Use

결과

Whole Tables	As Is	To Be	Reduction Amount
# of Tables	80	**66**	14
Capacity	TB (100%)	**TB (48.02%)**	TB (51.98%)
Price	₩	**₩**	₩

W/O same Tables	As Is	To Be	Reduction Amount
# of Tables	58	**44**	14
Capacity	TB (100%)	**TB (43.79%)**	TB (56.21%)
Price	₩	**₩**	₩

Effect of Use

Result

Average storage size decline rate of denormalized table : 47.66 %

Comparison between denormalization and unification table

Comparison between 3 source table and results

Table	Source Table	Denormalization	Unification(denormalized and unified)
Table 1	MB (100%)	MB (40.62%)	-
Table 2	MB (100%)	MB (36.06%)	
Table 3	GB (100%)	MB (35.26%)	
Total	GB (100%) -	GB Compared to Original (37.21%) -	GB Compared to Original (27.27%) Compared to Denormalization(73.28%)

※ 한 record에 여러 Not Shared Column들이 조합되면 압축 효율이 높아질 수 있음.

Usability

Usability

UNNEST

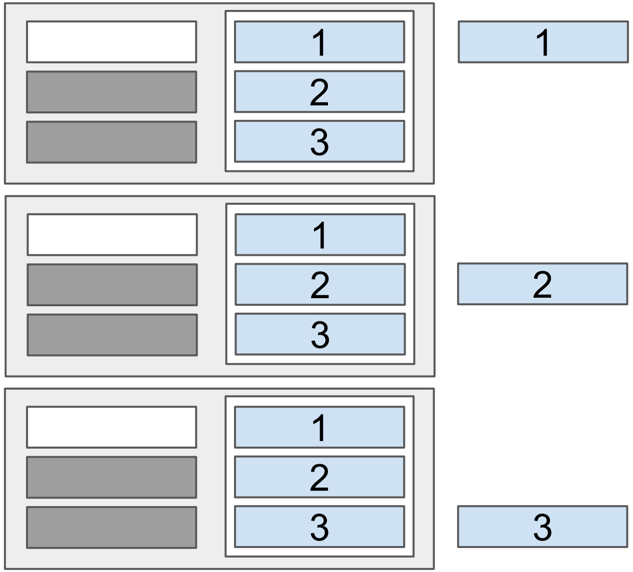
need to use UNNESET function when using RECORD(STRUCT) + ARRAY

※ Access is possible in the form of "Nested_field.*" to reference all fields within the nested structure

```
#1
SELECT *
FROM `project.dataset.table`
CROSS JOIN
    UNNEST(nested_field)

#2
SELECT * EXCEPT(nested_field)
FROM `project.dataset.table`, UNNEST(nested_field)

#3 access field with reference
SELECT ALIAS.target_nested_field
FROM `project.dataset.table`, UNNEST(nested_field) AS ALIAS
```



※ UNNEST 할 때 nested_field를 제외하지 않고 전체 SELECT 시 위 그림과 같이 SELECT 하는 경우가 있어 주의할 것

Usability

Criteria for RECORD Configuration

1. Exclusion of Partition Columns
   The leaf fields of a STRUCT cannot be used as partitions.
   Assumes partition usage for scan speed improvement

2. Conversion of Primary Key and Measure into STRUCT
   Measure fields are transformed into RECORD for non-code-based keys, considering the low likelihood of duplication.

3. Grouping Similar Types
   Aim to improve compression ratios through encoding by grouping similar types

4. Frequently Used Columns are placed outside the STRUCT for usability

Usability

Comparison by possible combinations of external columns from STRUCT

Should consider 1) Too many cases and 2) impossible cases for nothing

Comparison by DISTINCT combinations of external columns from STRUCT

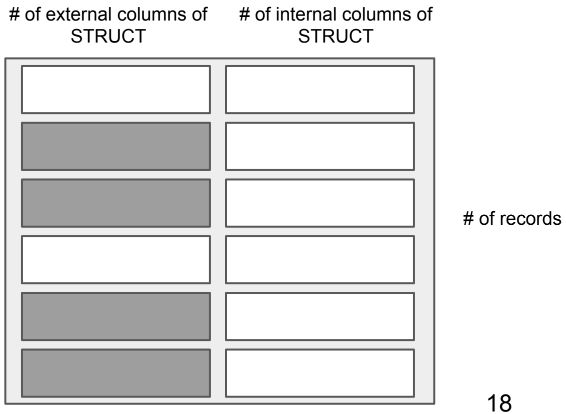
The compression ratio seems to follow a linear relationship with the number of distinct combinations of external columns from STRUCT.
However, there appears to be a phenomenon where the graph inverts after a certain level of compression.

# of external columns	possible combinations	DISTINCT combinations	record reduction rate	storage size	storage reduction rate
0 (Source)	X	408106	X	55.05MB	X
1	138,000,000,000	408106	0%	55.05MB	0%
2	1,270,000,000,000,000	357060	13%	50.03MB	9%
3	1,010,000,000,000,000,000	247168	39%	35.70MB	35%
4	13,158	6247	98%	28.45MB	48%
5	1	53	100%	33.83MB	39%

Usability

Optimization on Single STRUCT

min[possible combinations of external columns from STRUCT * # of external columns
+ # of every Records * internal columns]
→ Minimize the white area of graph
≈ max[(# of every Records - possible combinations of external columns from STRUCT)
*** # of external columns]**
→ Maximize the grey area of graph
→ Instead of possible combinations of external columns of STRUCT,
DISTINCT combinations of external columns of STRUCT
is expected to be more applicable for use



Limitation

- Combination
    - Dependency between variables are not considered

    → using DISTINCT case for real life applicatino
- Storage size : data type size need to be considered

    eg) text type with long length
- single STRUCT denormalization is only considered

Usability

Optimization in mathematical form

N = # of columns
X = external columns of STRUCT → N - X = internal columns of STRUCT
n = # of every records
A = # of possible combinations of external column from STRUCT
a = # of DISTINCT combinations of external column from STRUCT
Condition : a → A as n → ∞

min[A*X + n*(N-X)]

$= \min[A*X + n*N - n*X)] \approx \min[A*X - n*X]$

$\because$ n*N : Constant

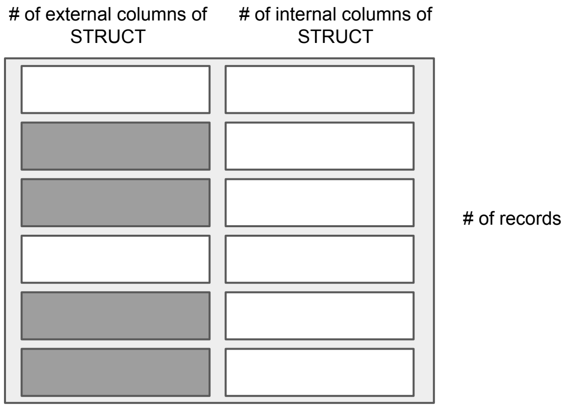
$= \mathbf{\min[(A - n)*X]} = -\max[(A - n)*X]$

$= \mathbf{\max[(n - A)*X]}$

$\therefore$ **min[A*X + n*(N-X)] ≈ max[(n - A)*X]**

$\because$ by Condition : a → A

$\therefore$ **min[a*X + n*(N-X)] ≈ max[(n - a)*X]**



Conclusion

Conclusion

Summary

- Depending on the table, the suitable form of nested columns may vary. Therefore, it is important to note that the criteria may differ for each table.

- Recommended to use a single STRUCT per table. If multiple STRUCTs are used, it is advised to configure them in a way that the STRUCTs are not interlinked

- For partitions and frequently used columns, placing them in external columns is recommended. Measures are recommended to be placed in internal columns of the STRUCT

- When structuring, it is recommended to consider the number of distinct combinations of external columns in a single STRUCT as a guideline for usage

Appendix

BigQuery Pricing

BigQuery Pricing

Storage based Pricing

Cost of storing data loaded into BigQuery

Active Storage

Every Table or Table Partitions modified in the last 90 days

Long-Term Storage

Every Table or Table Partitions that have not been modified in the last 90 days
50% price reduction automatically
No difference in performance, durability, availability between active and long-term

Storage	US Region	Tokyo Region	Details
Active	$0.020 per GB	$0.023 per GB	Free 10GB every month
Long-Term	$0.010 per GB	$0.016 per GB	Free 10GB every month

BigQuery Pricing

Analysis Pricing

cost to process queries, including SQL queries, user-defined functions, scripts, and certain data manipulation language (DML) and data definition language (DDL) statements

On-demand Pricing

charged for the number of bytes processed by each query
first 1 TiB of query data processed per month is free

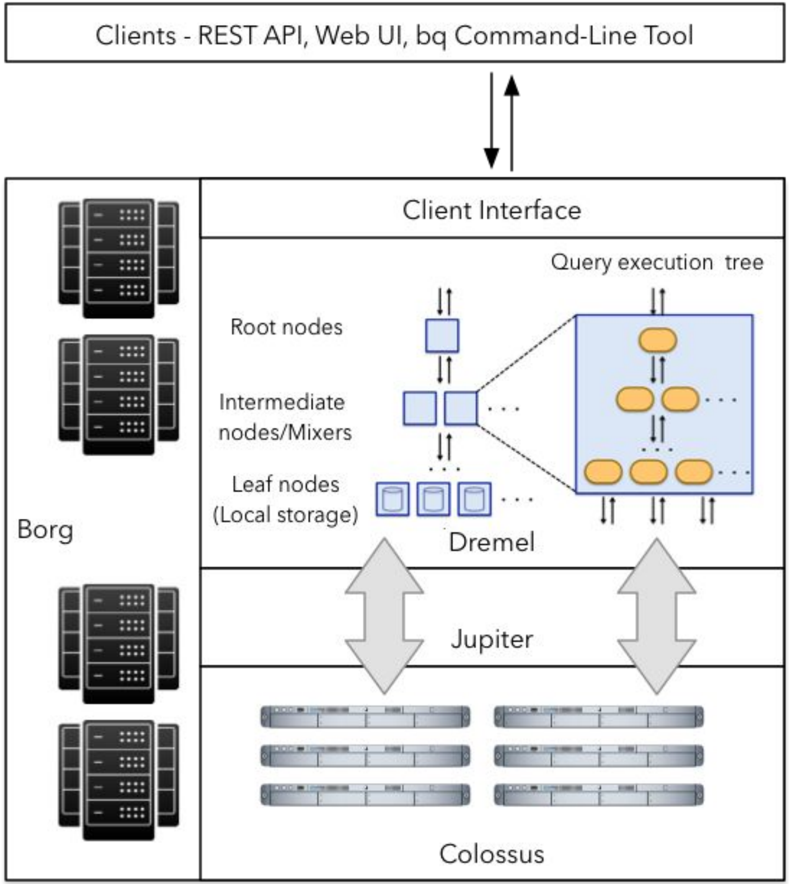
Flat-rate Pricing

Purchasing Virtual CPU ‘SLOT’
Variable slots: Committed for the first 60 seconds.
Monthly: Committed for the first 30 days.
Annual: Committed for 365 days.

Ref: https://cloud.google.com/bigquery/pricing?hl=en

BigQuery
Columnar Storage

BigQuery Columnar Storage



Borg

large-scale cluster management system
allocates server resources

Dremel

Columnar Storage
ree-based execution structure

Jupiter

Network Infrastructure
Bisectional Bandwidth
- 1 peta bit per second

→ Decreased importance of locality (region)
→ Reduced significance of lag considerations.

Colossus

Successor to the Google File System (GFS)
Capacitor : Columnar Storage Format

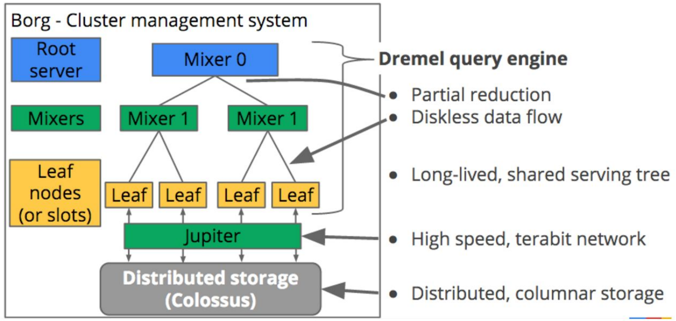
BigQuery Columnar Storage

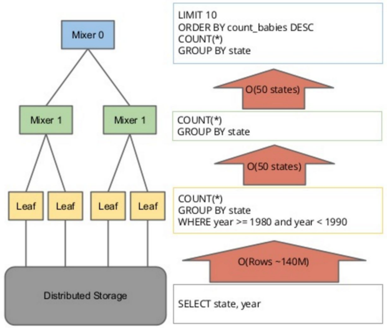
Dremel Engine

- database with the characteristic of storing data at the column level
- combining columnar storage and a tree architecture for SQL queries, achieving exceptionally fast speeds
- efficiency of data scanning is high due to the columnar storage method
- advantageous for read-centric OLAP tasks rather than OLTP

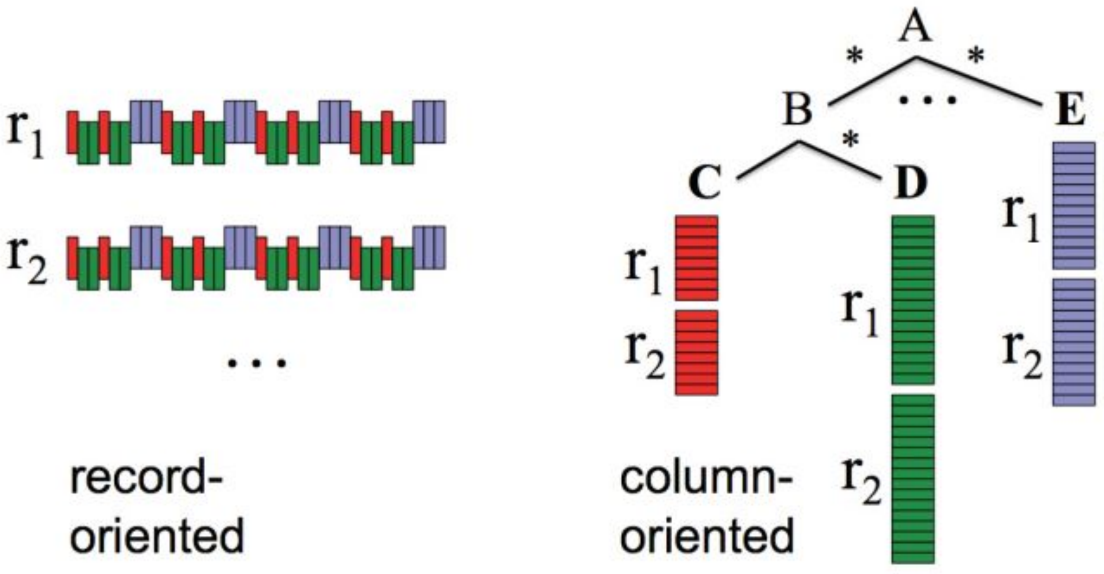
Tree Architecture Distribution(Execution Tree)

- Root Server(Mixer 0), Intermediate Server(Mixer1)......Leaf Server(Leaf)
- Distributes SQL queries in a tree structure to distributed machines and achieves fast processing.
- It processes data directly in RAM, without using disks except for leaf nodes, resulting in rapid speed.





BigQuery Columnar Storage



Record Oriented

FILE 1 : 1;Kim;F;32,
FILE 2 : 2;Nam;F;31,
FILE 3 : 3;Hong;M;30

Column Oriented

FILE 1 : 1:Kim, 2:Nam, 3:Hong
FILE 2 : 1:F, 2:F, 3:M
FILE 3 : 1:32, 2:31, 3:30

Ref: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/36632.pdf

BigQuery Columnar Storage

Properties

- Traffic Minimization
  - Cost savings occur because only the necessary columns are queried.
- High Compression Rate
  - Gathering and storing similar data types makes compression easier.
  - Run Length Encoding(RLE), Dictionary Encoding

Encoding

- Repetition and Definition Level
  - Concepts used when reconstructing a column storage record.
  - Combine by calling each columns→ Efficient query capacity achieved
- Stored together for each Block fragment (by column)

Capacitor

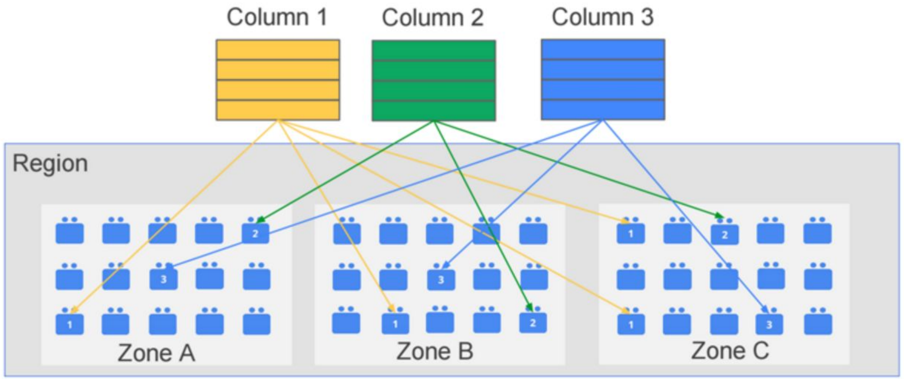
- BigQuery's Column-Based Storage Format
- Collects various statistical information and stores it in Colossus.
- Estimates and generates optimization models based on the gathered information.

BigQuery Columnar Storage

Other Characteristics

- No Key, No Index(Full Scan Only)
  Avoid full scan with Partitioning

- Eventual Consistency
  Replicating data to three data centers
  Typically, it is reflected immediately, but in some situations, it may take a few minutes.



출처: [링크]