ADDB: FPWRITE

연세대학교 컴퓨터과학과 박상현

2019년 5월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

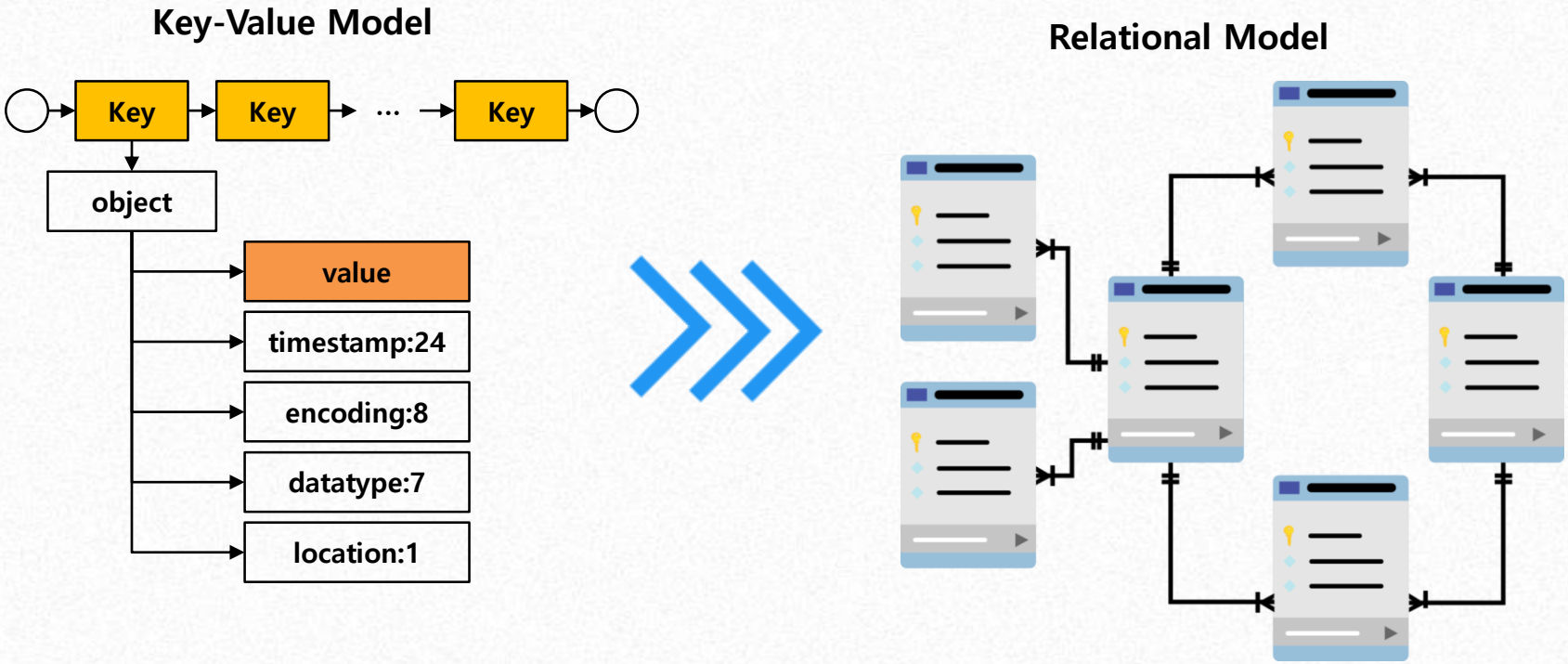
정보통신기술진흥센터
Institute for Information & communications Technology Promotion

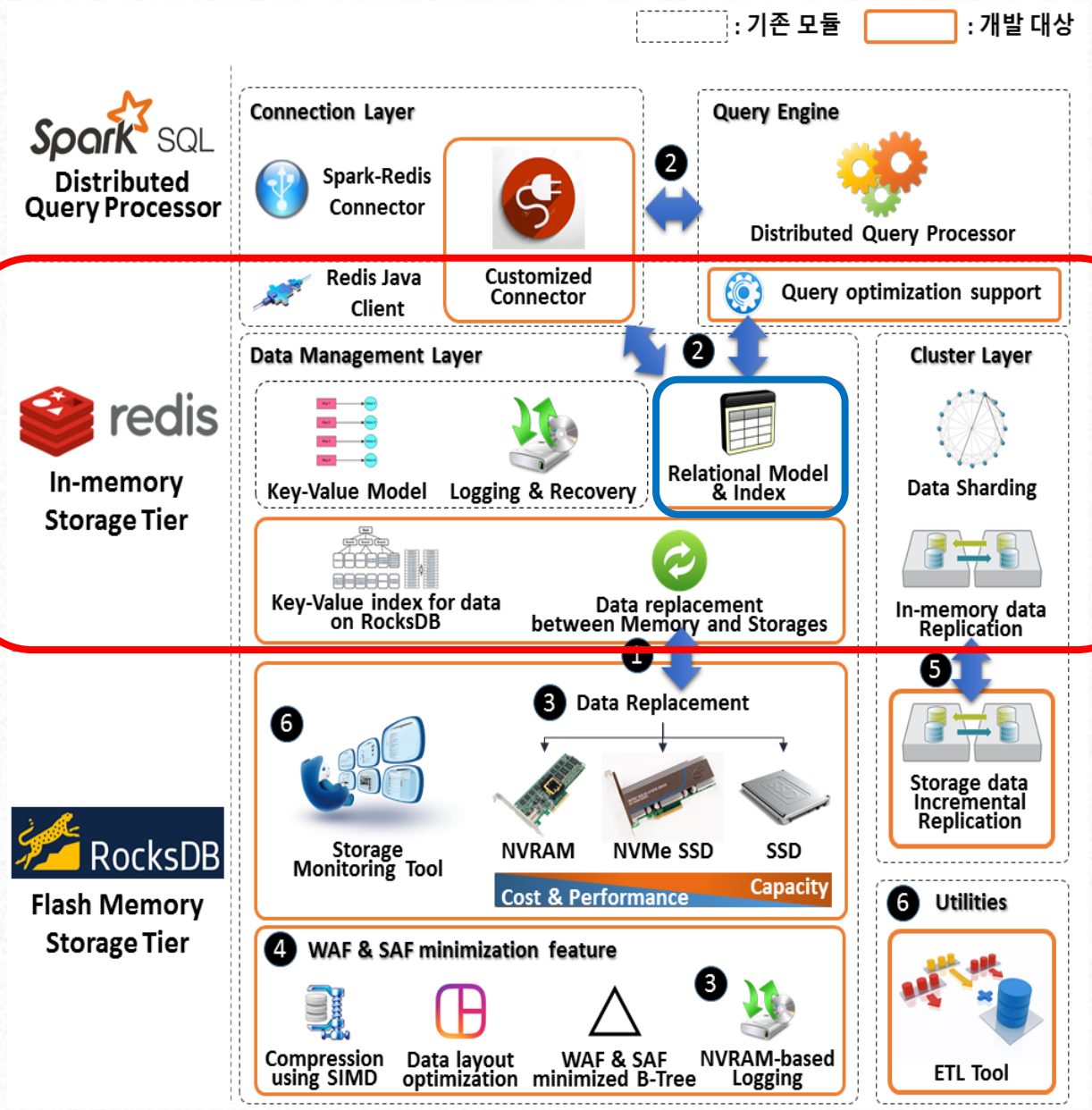
목차

01. Relational Model 설계

분산 DBMS의 관계형 모델 데이터 타입 구현

- Redis 내 관계형 모델 자료구조
  - Key-Value 형태의 저장 기반을 관계형 모델로 변경
  - Key-Value Store의 빠른 저장 속도를 활용하여 데이터를 적재



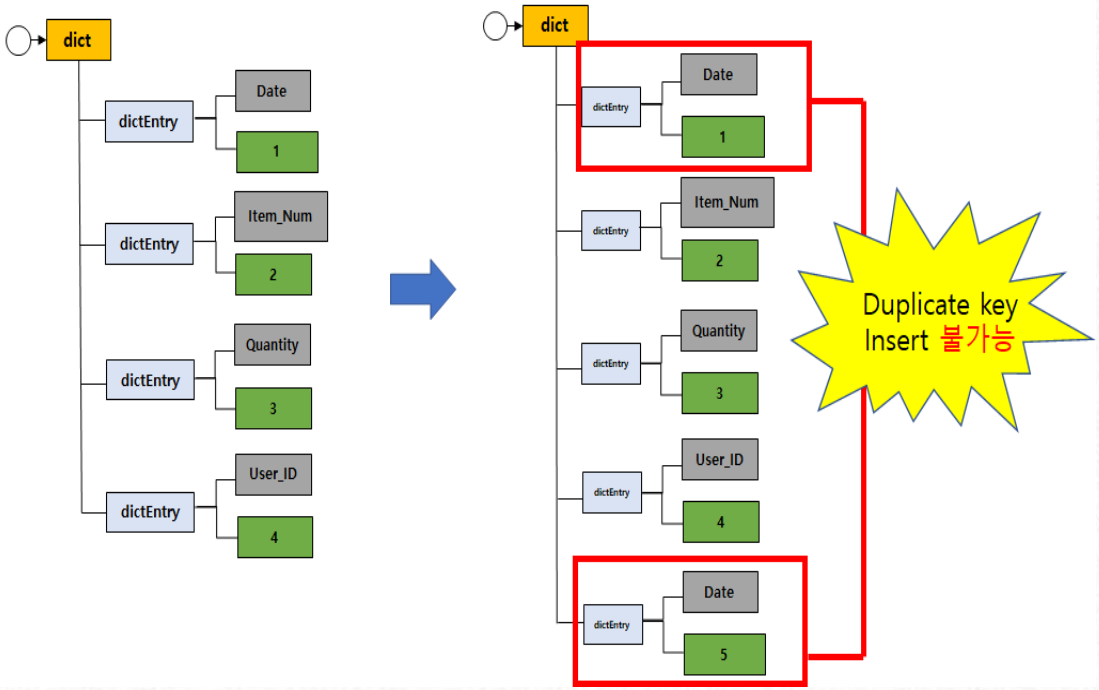

: 기존 모듈
: 개발 대상
Spark SQL
Distributed Query Processor
Connection Layer
Spark-Redis Connector
Redis Java Client
Customized Connector
Query Engine
Distributed Query Processor
Query optimization support
redis
In-memory Storage Tier
Data Management Layer
Key-Value Model
Logging & Recovery
Relational Model & Index
Key-Value index for data on RocksDB
Data replacement between Memory and Storages
Cluster Layer
Data Sharding
In-memory data Replication
Storage data Incremental Replication
RocksDB
Flash Memory Storage Tier
Storage Monitoring Tool
Data Replacement
NVRAM
NVMe SSD
SSD
Cost & Performance
Capacity
WAF & SAF minimization feature
Compression using SIMD
Data layout optimization
WAF & SAF minimized B-Tree
NVRAM-based Logging
Utilities
ETL Tool

분산 DBMS의 관계형 모델 데이터 타입 구현

- 관계형 모델 데이터 적재 문제
  - 키-값 중복문제로 인하여 관계형 모델 적재 불가능
  - 새로운 관계형 모델 설계가 필요

Date	Item_Num	Quantity	User_ID
1	2	3	4

Date	Item_Num	Quantity	User_ID
1	2	3	4
5	6	7	8

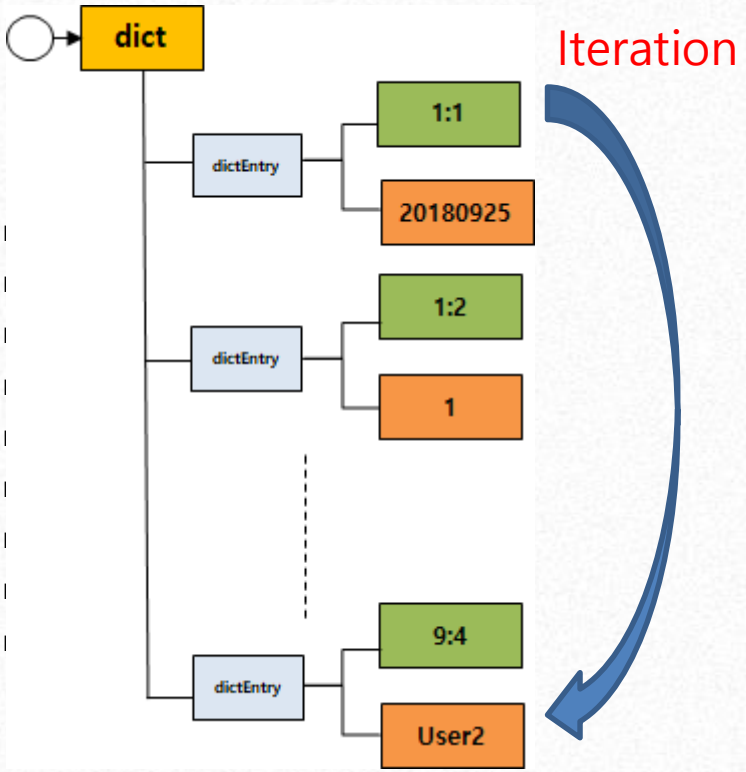


분산 DBMS의 관계형 모델 데이터 타입 구현

- 관계형 모델 설계

TableID = 1

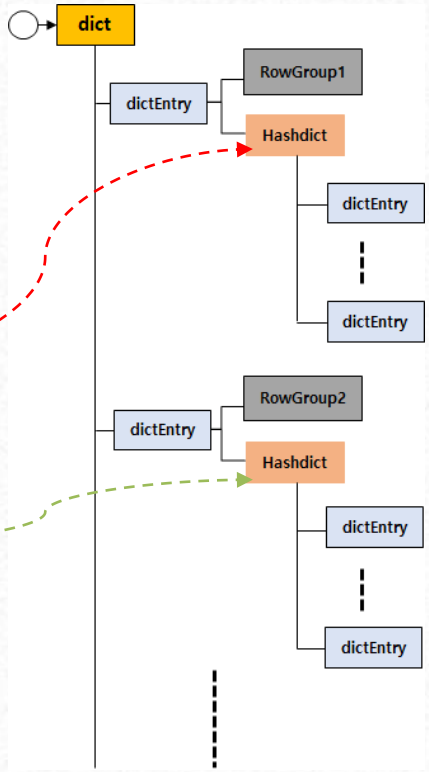
	ColumnID =1 Date	ColumnID =2 Item_Num	ColumnID =3 Quantity	ColumnID =4 User_ID
RowgroupID = 1 RowID = 1	20180925	1	4	User2
RowID = 2	20180925	1	2	User1
	20180925	1	6	User6
RowgroupID = 2	20180925	1	11	User8
	20180925	1	5	User5
	20180926	1	3	User1
RowgroupID = 3	20180926	2	3	User13
	20180926	2	7	User8
RowID = 9	20180926	2	15	User2

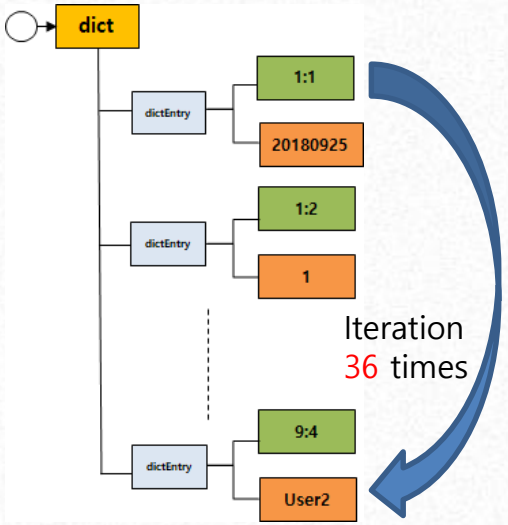


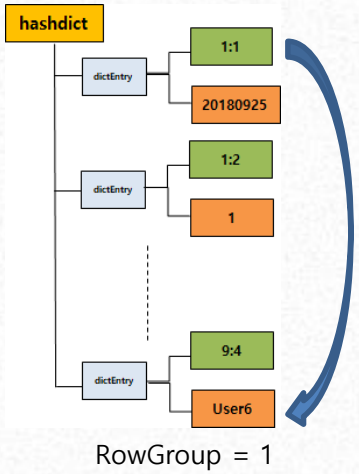
Iterator로 인한 읽기 성능 저하
과도한 dictEntry 생성 및 데이터 핸들링 단위 증가

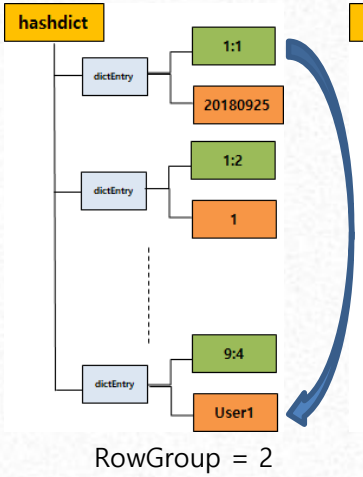
- Rowgroup
  - Access 빈도를 줄이고 데이터 관리를 편하게 하기 위해 Row를 Group 단위로 구분

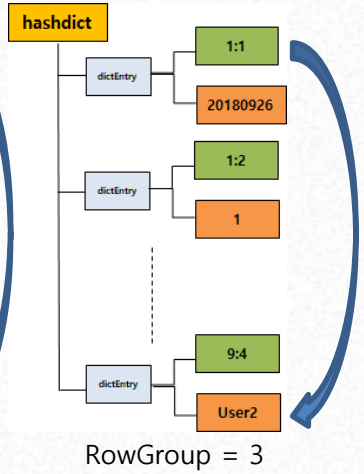
	Date	Item_Num	Quantity	User_ID
RowGroup = 1	20180925	1	4	User2
	20180925	1	2	User1
	20180925	1	6	User6
RowGroup = 2	20180925	1	11	User8
	20180925	1	5	User5
	20180926	1	3	User1
RowGroup = 3	20180926	2	3	User13
	20180926	2	7	User8
	20180926	2	15	User2











분산 DBMS의 관계형 모델 데이터 타입 구현

- 관계형 모델 설계

Date	Item_Num	Quantity	User_ID
20180925	1	4	User2
20180925	1	2	User1
20180925	1	6	User6
20180925	1	11	User8
20180925	1	5	User5
20180926	1	3	User1
20180926	2	3	User13
20180926	2	7	User8
20180926	2	15	User2

- TableID
- RowID
- ColumnID
- RowGroupID
- PartitionInfo ➔ ColumnID:Value

PartitionInfo = 2:1

Date	Item_Num	Quantity	User_ID
20180925	1	4	User2
20180925	1	2	User1
20180925	1	6	User6
20180925	1	11	User8
20180925	1	5	User5
20180926	1	3	User1

PartitionInfo = 2:2

Date	Item_Num	Quantity	User_ID
20180926	2	3	User13
20180926	2	7	User8
20180926	2	15	User2

02. FPWRITE 예시 (초기버전)

- Redis 관계형 모델 구조 적재 명령어(fpwrite)
  - fpwriteCommand : 데이터 적재 명령어
  - 데이터는 Row단위로 적재
  - Parameter
    - TableID
    - PartitionInfo
    - Column count
    - Index column
    - Value

- TableID ➔ 100
- Rowgroup size ➔ 1
- PartitionInfo ➔ 2:1

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7

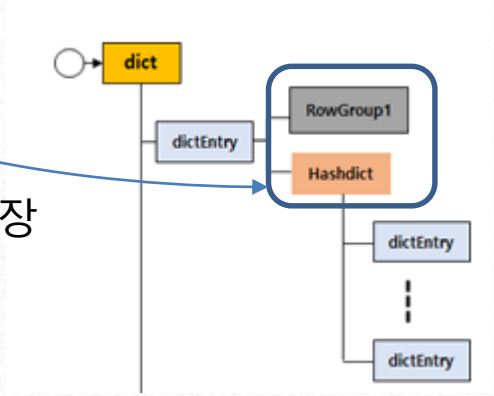
FPWRITE D:{TableID : PartitionInfo}:G:RowgroupID PartitionInfo ColumnCount IndexColumn Value…

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

Data prefix

• DataField

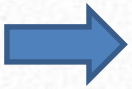
- Datakey – Hashdict pair에서 Hashdict에 저장된 value의 위치를 나타내는 Field
- Row : Column으로 구성
- Metadict에서 조회한 정보를 사용하여
  데이터 적재시 Field를 생성하여 데이터를 저장



Row : Column

- TableID ➔ 100
- Rowgroup size ➔3
- PartitionInfo ➔ 2:1

Column / Row	1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2



Column / Row	1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	1:2	1:3	1:4
2	2:1	2:2	2:3	2:4
3	3:1	3:2	3:3	3:4
4	1:1	1:2	1:3	1:4
5	2:1	2:2	2:3	2:4
6	3:1	3:2	3:3	3:4
7	1:1	1:2	1:3	1:4
8	2:1	2:2	2:3	2:4
9	3:1	3:2	3:3	3:4

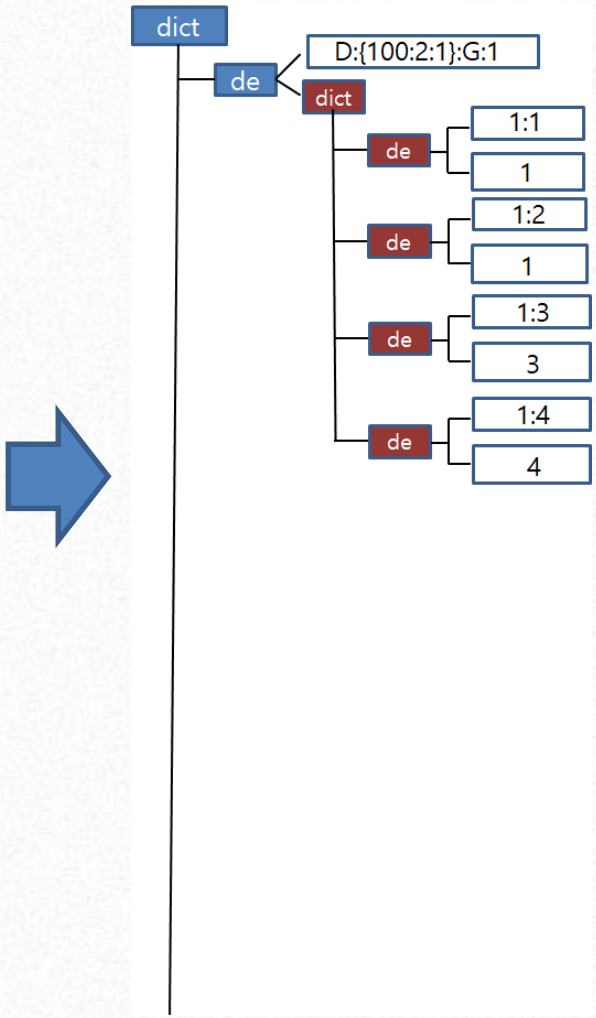
• Redis 관계형 모델 구조 적재 예시(fpwrite)

- TableID ➔ 100
- Rowgroup size ➔ 1
- PartitionInfo ➔ 2:1

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

FPWRITE D:{100:2:1} 2:1 4 0 5 1 6 7



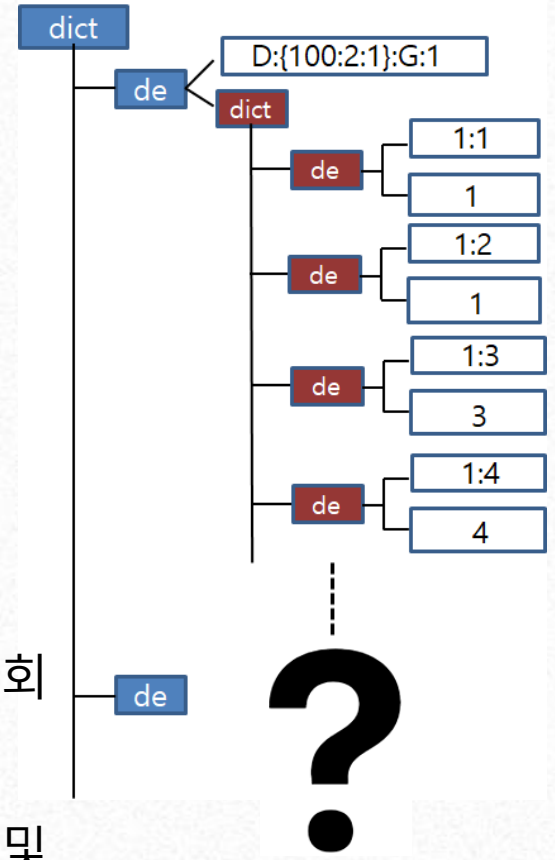
03. Metadict

분산 DBMS의 관계형 모델 인덱스 구현

Which Rowgroup?
Which Rownumber?

Date	Item_Num	Quantity	User_ID
1	1	3	4
5	1	7	8
9	1	11	12
13	1	15	16
17	1	19	20
21	1	23	24

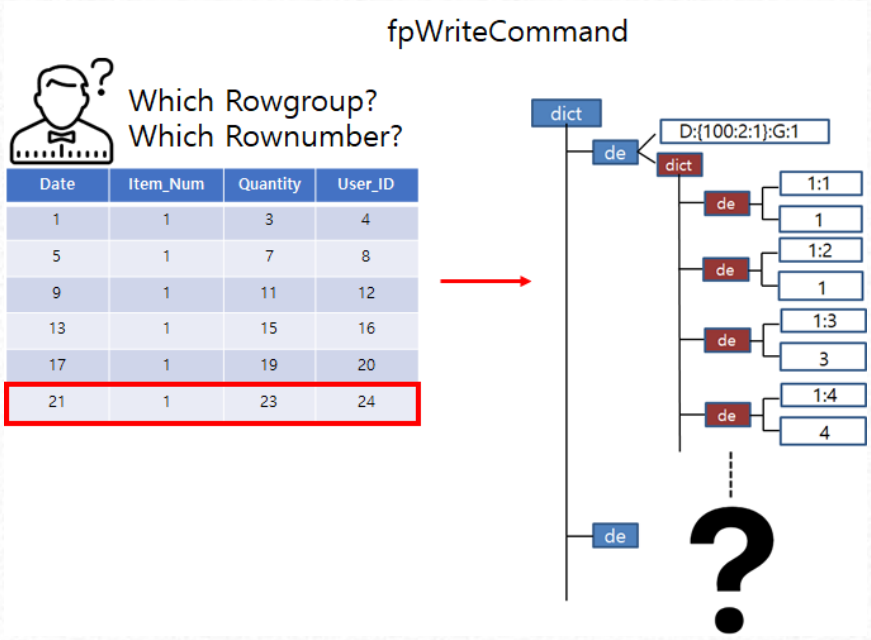
fpWriteCommand

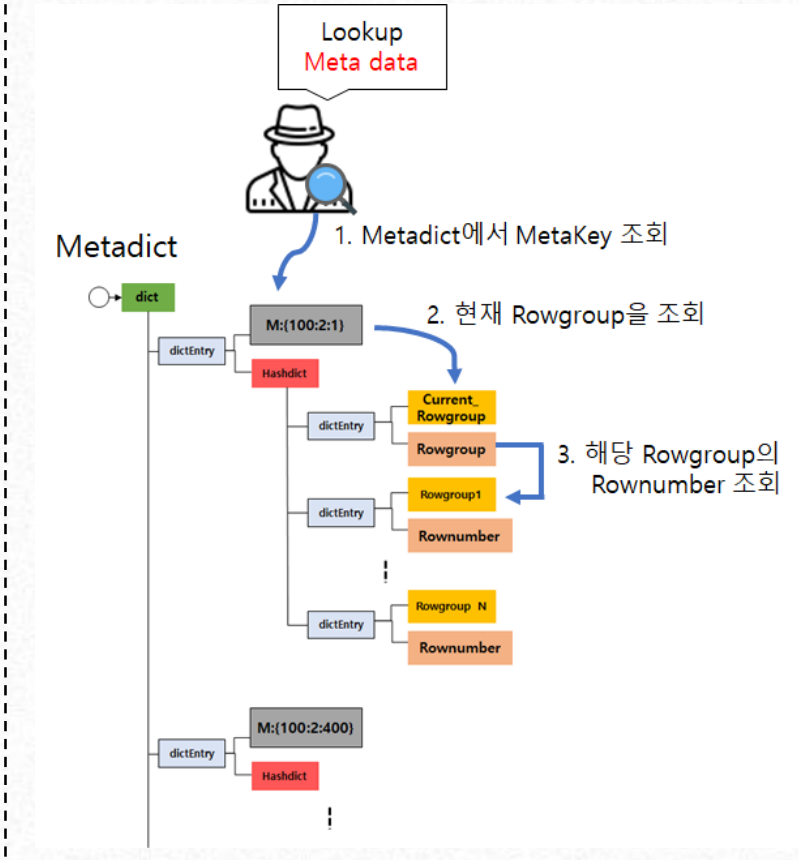


- 데이터적재 요청시, dict를 순회하여 정보를 조회할 경우 성능 저하 발생
- 데이터 적재를 위해 필요한 정보를 별도로 저장 및 관리하는 구조가 요구됨

분산 DBMS의 관계형 모델 인덱스 구현

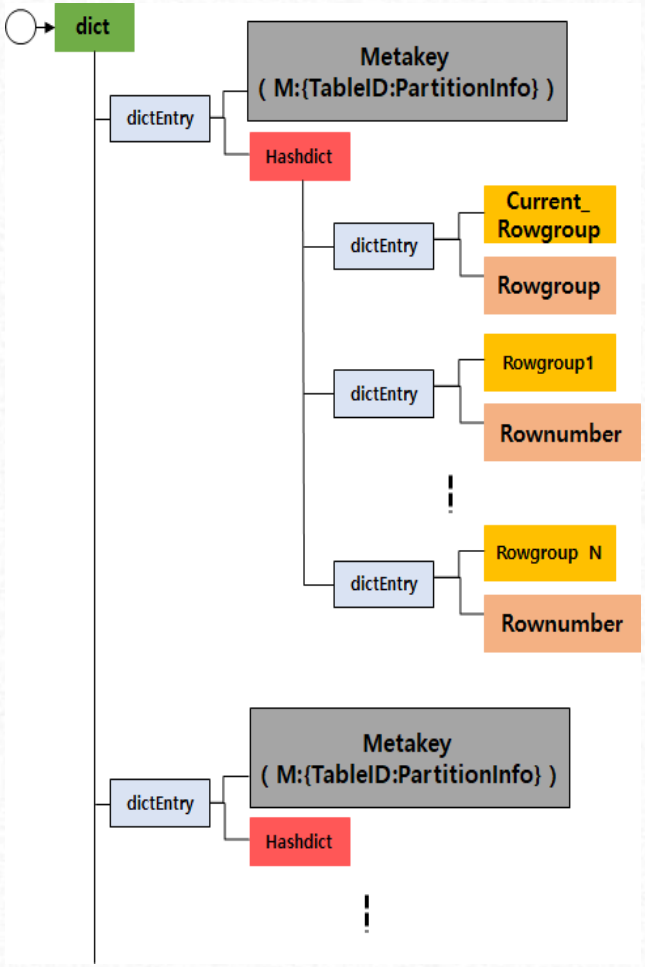
- Metadict
  - 데이터 적재에 필요한 정보를 저장
    - Current_RowgroupID
    - RowgroupID
    - Rownumber





분산 DBMS의 관계형 모델 인덱스 구현

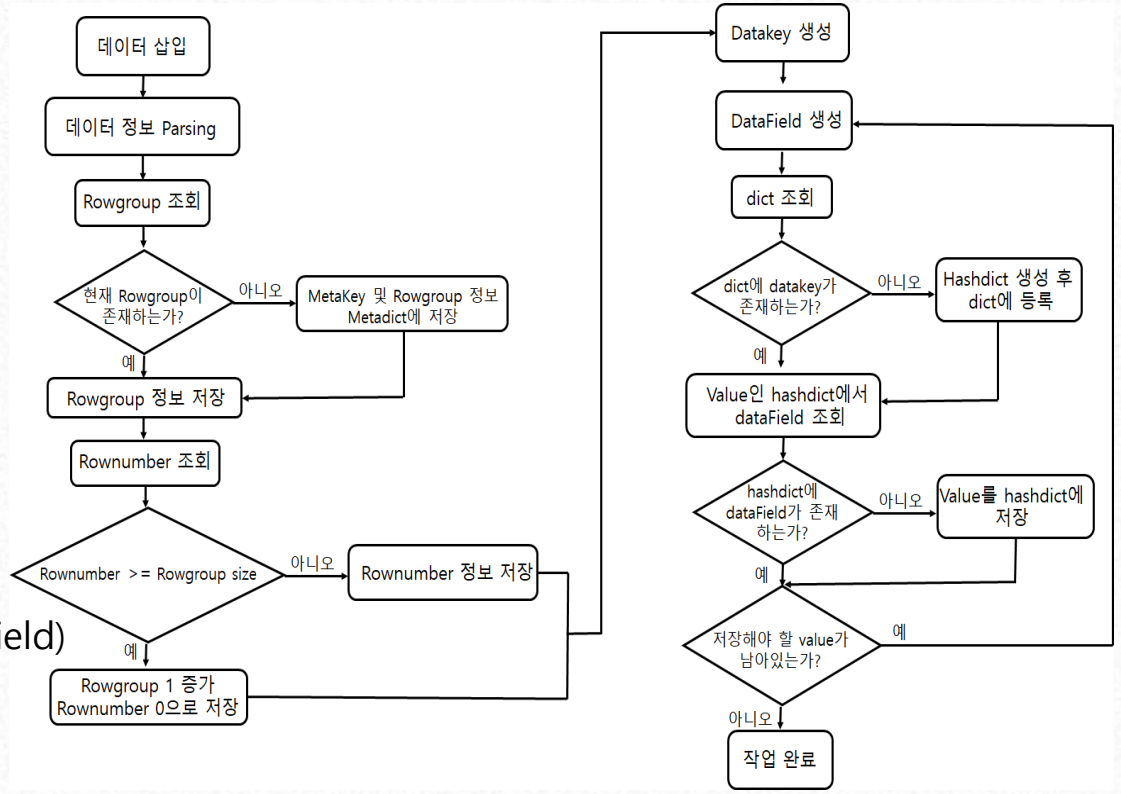
- Metadict 구조



- Metakey :**M:{TableID:PartitionInfo}**
  - ✓ Meta정보를 찾기 위한 Key
- Current_RowgroupID
  - ✓ 데이터가 적재되어야 할 RowgroupID 정보
- RowgroupID
  - ✓ 데이터 적재로 인하여 생성된 Rowgroup
- Rownumber
  - ✓ 각 Rowgroup에 저장된 Row 개수

FPWRITECOMMAND STEP
fpwriteCommand

1. Command Parsing
   - ✓ TableID
   - ✓ PartitionInfo
   - ✓ RowgroupID
2. Metadict 조회
   - ✓ RowgroupID
   - ✓ Rownumber
3. Datakey 생성
4. 데이터 적재
   1. DataField 생성
   2. Dict 조회(Datakey)
   3. Hashdict 조회(DataField)
   4. 데이터 저장
5. Metadict 갱신



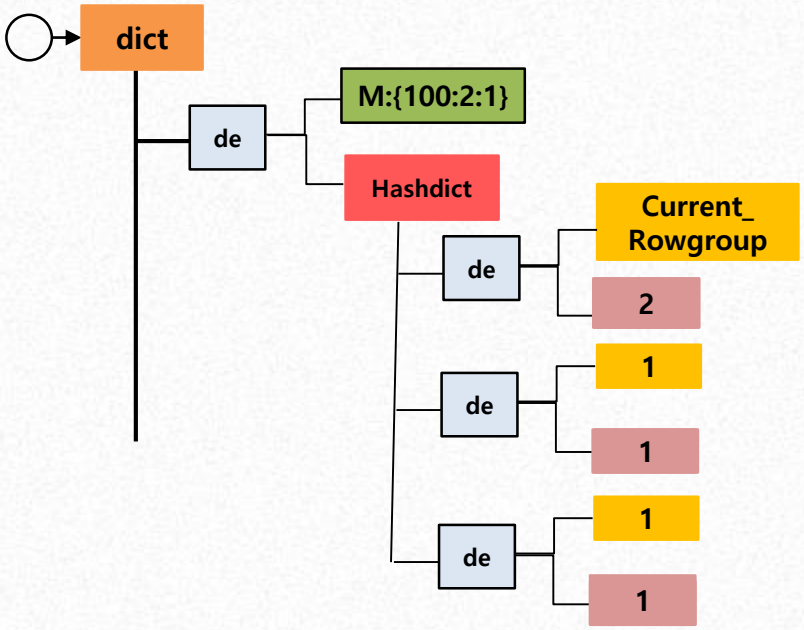
- Metadict를 활용한 데이터 적재 과정

- TableID ➔ 100
- Rowgroup size ➔ 1
- PartitionInfo ➔ 2:1

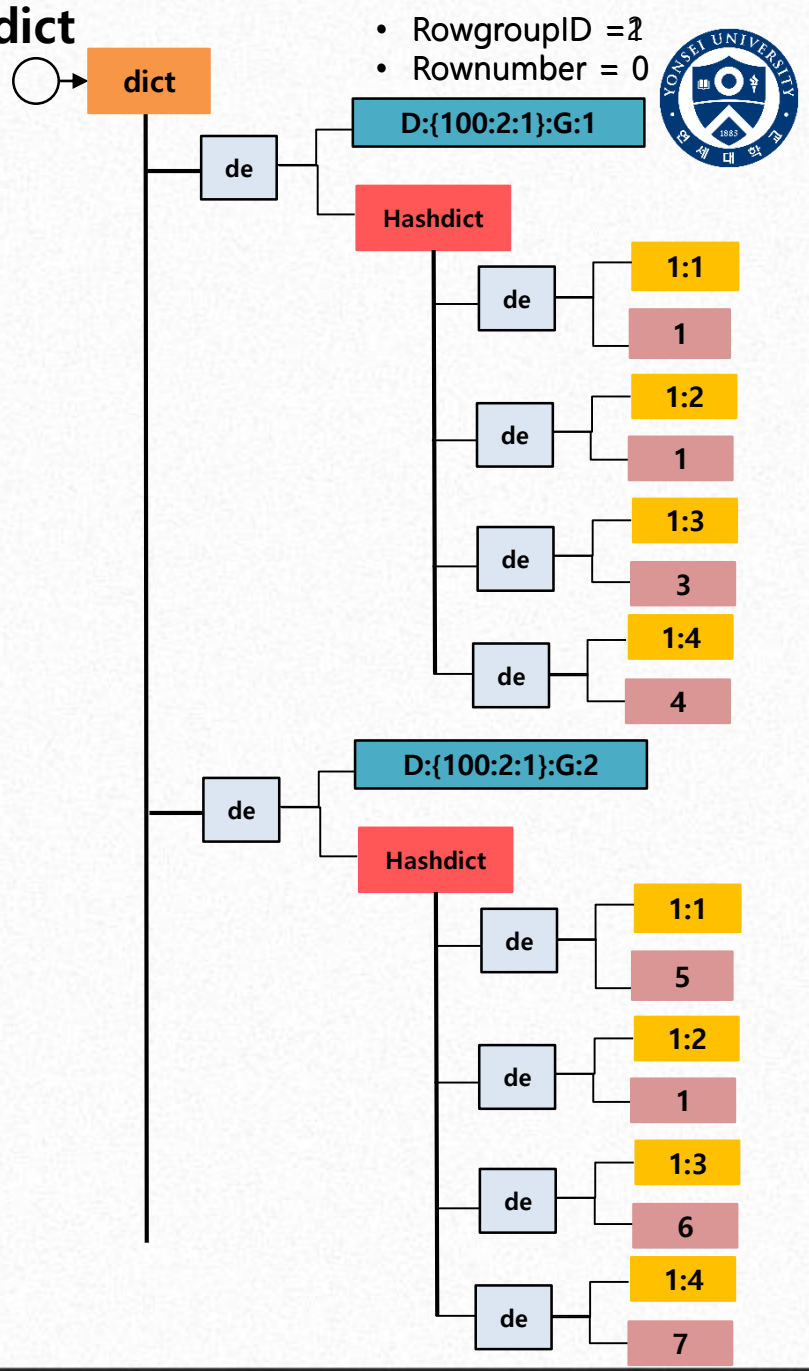
Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7

FPWRITE D:{100:2:1} 2:1 4 0 5 1 6 7

Metadict



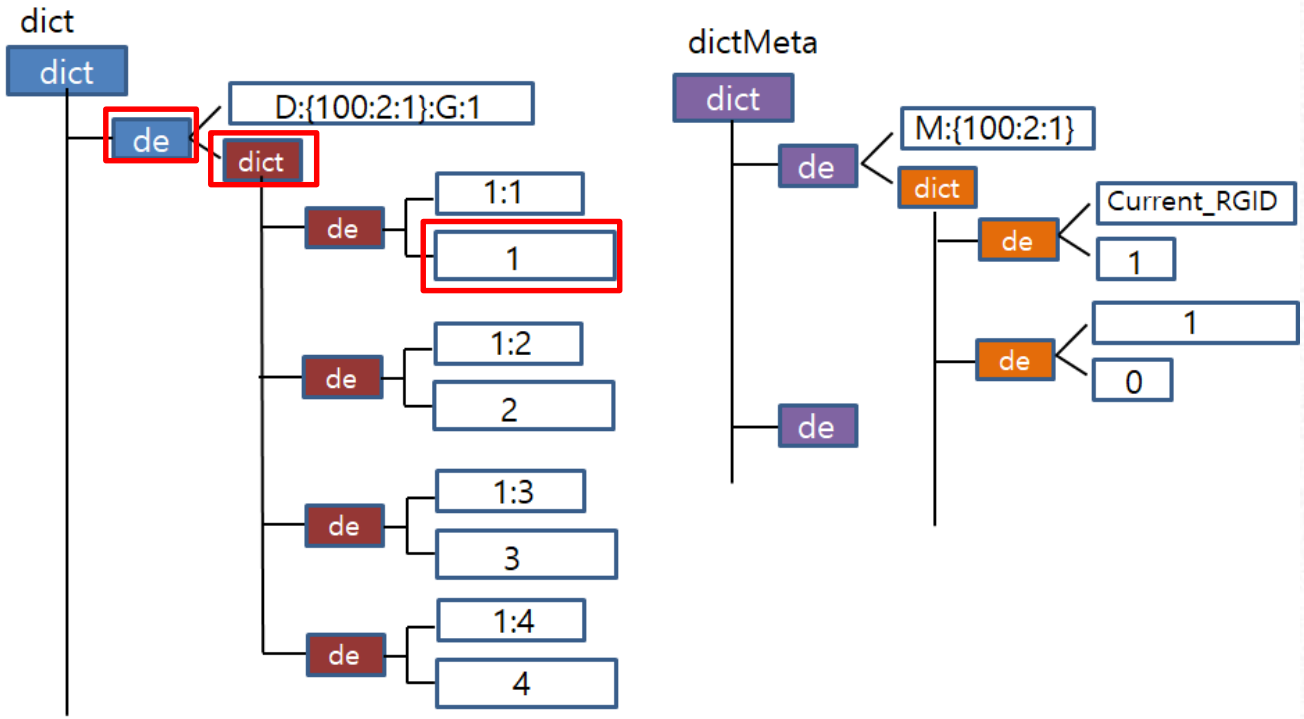
dict



04. Relational + Vector

관계형 모델 개선

- **초기 관계형 모델의 문제**
  - 과도한 메모리 사용

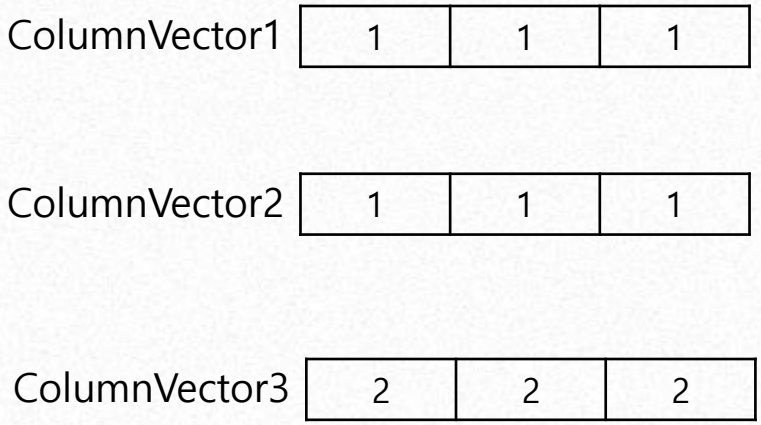


- 데이터 저장에 필요한 부가적인 요소로 인하여 메모리 사용량이 급격히 증가
  - dictEntry – 24B
  - Redis object – 16B

관계형 모델 개선

- Vector를 사용한 관계형 모델 개선
  - 저장된 데이터 타입을 고려하여 Column 단위 Vector를 적용
  - 생성되는 dictEntry와 Redis Object 수 감소

Date	Item_Num	Quantity	User_ID
20180925	1	4	User2
20180925	1	2	User1
20180925	1	6	User6
20180925	1	11	User8
20180925	1	5	User5
20180926	1	3	User1
20180926	2	3	User13
20180926	2	7	User8
20180926	2	15	User2



- DataField 변경
  - Vector 적용 전

Row : Column

- TableID ➔ 100
- Rowgroup size ➔3
- PartitionInfo ➔ 2:1

Row \ Column	1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2

Row \ Column	1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	1:2	1:3	1:4
2	2:1	2:2	2:3	2:4
3	3:1	3:2	3:3	3:4
4	1:1	1:2	1:3	1:4
5	2:1	2:2	2:3	2:4
6	3:1	3:2	3:3	3:4
7	1:1	1:2	1:3	1:4
8	2:1	2:2	2:3	2:4
9	3:1	3:2	3:3	3:4

  - Vector 적용 후

Column : VectorID

- TableID ➔ 100
- Rowgroup size ➔ 3
- PartitionInfo ➔ 2:1
- Vector size ➔ 3

Row \ Column	1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2

Row \ Column	1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	2:1	3:1	4:1
2	1:1	2:1	3:1	4:1
3	1:1	2:1	3:1	4:1
4	1:1	2:1	3:1	4:1
5	1:1	2:1	3:1	4:1
6	1:1	2:1	3:1	4:1
7	1:1	2:1	3:1	4:1
8	1:1	2:1	3:1	4:1
9	1:1	2:1	3:1	4:1

- Vector & Rowgroup size 변경에 따른 DataField 변경

Column : VectorID

- Vector size = 3
- Rowgroup size = 3

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	2:1	3:1	4:1
2	1:1	2:1	3:1	4:1
3	1:1	2:1	3:1	4:1
4	1:1	2:1	3:1	4:1
5	1:1	2:1	3:1	4:1
6	1:1	2:1	3:1	4:1
7	1:1	2:1	3:1	4:1
8	1:1	2:1	3:1	4:1
9	1:1	2:1	3:1	4:1

- Vector size = 3
- Rowgroup size = 6

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	2:1	3:1	4:1
2	1:1	2:1	3:1	4:1
3	1:1	2:1	3:1	4:1
4	1:2	2:2	3:2	4:2
5	1:2	2:2	3:2	4:2
6	1:2	2:2	3:2	4:2
7	1:1	2:1	3:1	4:1
8	1:1	2:1	3:1	4:1
9	1:1	2:1	3:1	4:1

- Vector & Rowgroup size 변경에 따른 DataField 변경2

Column : VectorID

- Vector size = 3
- Rowgroup size = 6

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	2:1	3:1	4:1
2	1:1	2:1	3:1	4:1
3	1:1	2:1	3:1	4:1
4	1:2	2:2	3:2	4:2
5	1:2	2:2	3:2	4:2
6	1:2	2:2	3:2	4:2
7	1:1	2:1	3:1	4:1
8	1:1	2:1	3:1	4:1
9	1:1	2:1	3:1	4:1

- Vector size = 2
- Rowgroup size = 6

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	20180925	1	4	User2
2	20180925	1	2	User1
3	20180925	1	6	User6
4	20180925	1	11	User8
5	20180925	1	5	User5
6	20180926	1	3	User1
7	20180926	2	3	User13
8	20180926	2	7	User8
9	20180926	2	15	User2

Row	Column 1	2	3	4
	Date	Item_Num	Quantity	User_ID
1	1:1	2:1	3:1	4:1
2	1:1	2:1	3:1	4:1
3	1:2	2:2	3:2	4:2
4	1:2	2:2	3:2	4:2
5	1:3	2:3	3:3	4:3
6	1:3	2:3	3:3	4:3
7	1:1	2:1	3:1	4:1
8	1:1	2:1	3:1	4:1
9	1:2	2:2	3:2	4:2

• Redis 관계형 모델 적재

- TableID = 100
  - Rowgroup Size = 2
  - Partition Size = 2 2:1
  - Vector Size = 2:1

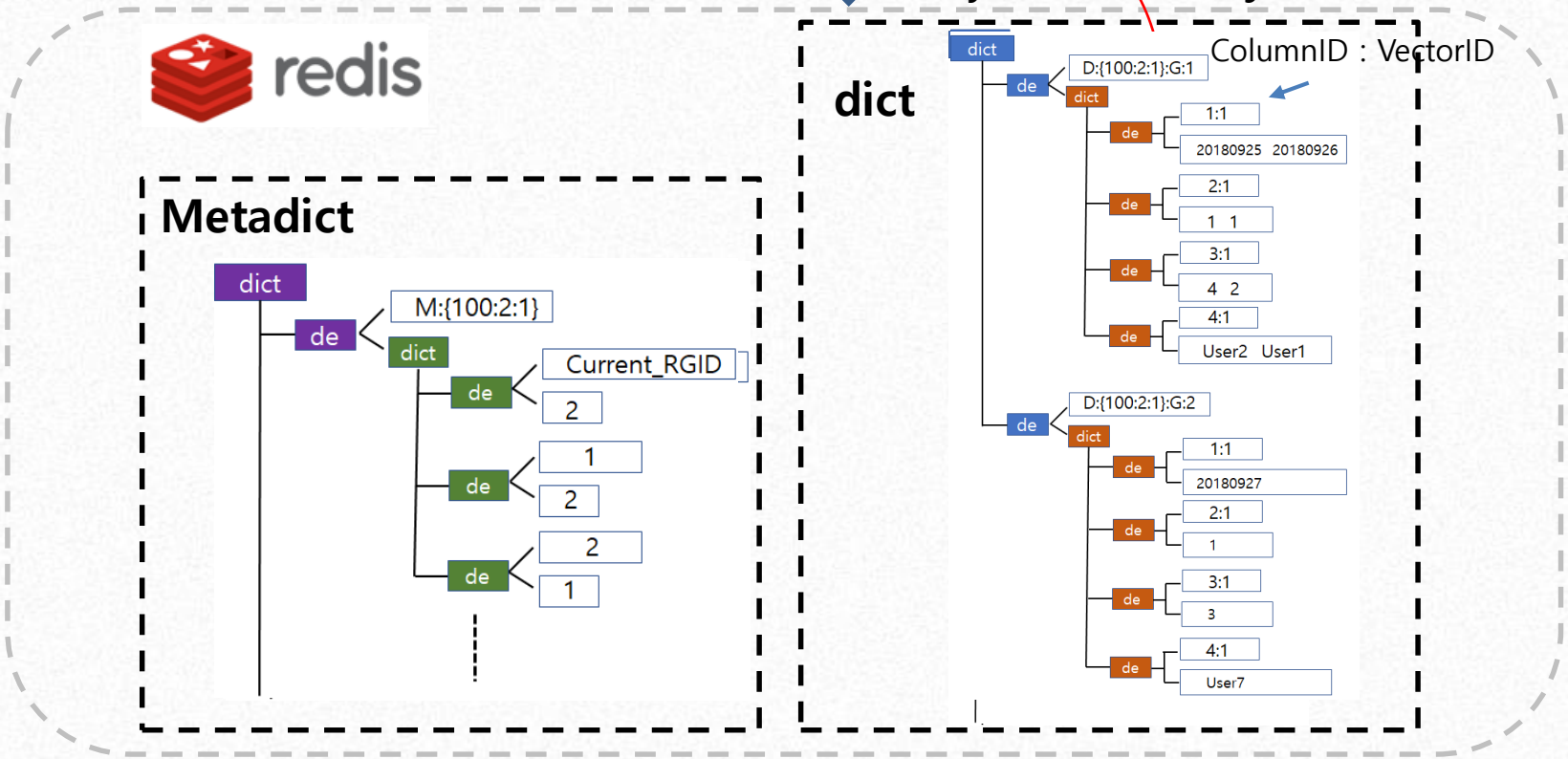
Date	Item_Num	Quantity	User_ID
20180925	1	4	User2
20180926	1	2	User1
20180927	1	3	User7

데이터 저장에 필요한 부가적인 요소로 인한 메모리 사용량 증가
✓ dictEntry – 24B
✓ Redis Object – 16B

Column : VectorID 단위 저장

WRITE D:{100:2:1} 2:1 4 0 20180926 1 2 User1
D:{100:2:1} 2:1 4 0 20180927 1 3 User7

dictEntry & Redis Object 수 감소



• FPWRITECOMMAND CODE

```
void fpWriteCommand(client *c){

    serverLog(LL_DEBUG,"FPWRITE COMMAND START");

    int fpWrite_result = C_OK;
    int i;
    long long insertedRow = 0;
    int Enroll_queue = 0;
    int partial_flag = 0;
    //struct redisClient *fakeClient = NULL;

    serverLog(LL_DEBUG, "fpWrite Param List ==> Key : %s, partition : %s, num_of_column : %s, indexColumn : %s",
            (char *) c->argv[1]->ptr,(char *) c->argv[2]->ptr, (char *) c->argv[3]->ptr , (char *) c->argv[4]->ptr);

    /*parsing dataInfo*/
    NewDataKeyInfo *dataKeyInfo = parsingDataKeyInfo((sds)c->argv[1]->ptr);

    /*get column number*/
    int column_number = atoi((char *) c->argv[3]->ptr);
    assert(column_number <= MAX_COLUMN_NUMBER);
    serverLog(LL_DEBUG, "fpWrite Column Number : %d", column_number);

    /*get value number*/
    int value_num = c->argc - 5;
    serverLog(LL_DEBUG ,"VALUE NUM : %d", value_num);

    /*compare with column number and arguments*/
    if((value_num % column_number) != 0 ){
     serverLog(LL_WARNING,"column number and args number do not match");
     addReplyError(c, "column_number Error");
     return;
  }

    serverLog(LL_DEBUG,"VALID DATAKEYSTRING ==> tableId : %d, partitionInfo : %s, rowgroup : %d",
              dataKeyInfo->tableId, dataKeyInfo->partitionInfo.partitionString, dataKeyInfo->rowGroupId);

    /*get rowgroup info from Metadict*/
    int rowGroupId = getRowgroupInfo(c->db, dataKeyInfo);
    serverLog(LL_DEBUG, "rowGroupId = %d", rowGroupId);

    /*get rownumber info from Metadict*/
    int row_number = getRowNumberInfoAndSetRowNumberInfo(c->db, dataKeyInfo);
    serverLog(LL_DEBUG, "rowNumber = %d", row_number);
    int prev_row = row_number;

    /*set rowNumber Info to Metadict*/
    if(row_number == 0 ){
     incRowNumber(c->db, dataKeyInfo, 0);
  }
    /*check rowgroup size*/
    if(row_number >= server.rowgroup_size){
     rowGroupId = IncRowgroupIdAndModifyInfo(c->db, dataKeyInfo, 1);
     row_number = 0;
     Enroll_queue = 1;
  }


    robj *dataKeyString = NULL;
    dataKeyString = generateDataKey(dataKeyInfo);
    //serverLog(LL_VERBOSE, "DATAKEY1 :  %s", (char *) dataKeyString->ptr);

    dictEntry *entryDict = dictFind(c->db->dict, dataKeyString->ptr);
       if (entryDict != NULL) {
         robj * val = dictGetVal(entryDict);
         //serverLog(LL_VERBOSE, "DATAKEYtest :  %d", val->location);

         __sync_synchronize();
         if (val->location != LOCATION_REDIS_ONLY &&  !Enroll_queue) {
           serverLog(LL_VERBOSE, "---PARTIAL ROWGROUP OCCURED---");
           serverLog(LL_VERBOSE, "PREV DATAKEY : %s", (char *)dataKeyString->ptr);
           int prev_rowgroup = rowGroupId;
           serverLog(LL_VERBOSE, "PREV ROWGROUPID : %d", prev_rowgroup);
           int prev_rownumber = row_number;
           serverLog(LL_VERBOSE, "PREV Rownumber : %d", prev_rownumber);
           serverLog(LL_VERBOSE, "-----------------------------");

           rowGroupId = IncRowgroupIdAndModifyInfo(c->db, dataKeyInfo, 1);;
          // incRowNumber(c->db, dataKeyInfo, 0);
           decrRefCount(dataKeyString);
           dataKeyString = generateDataKey(dataKeyInfo);
           row_number = 0;
           partial_flag = 1;


           serverLog(LL_VERBOSE, "---MAKE NEW ROWGROUP---");
           serverLog(LL_VERBOSE, "NEW DATAKEY : %s", (char *)dataKeyString->ptr);
           int rowGroupId2 = getRowgroupInfo(c->db, dataKeyInfo);
           serverLog(LL_VERBOSE, "NEW ROWGROUPID : %d", rowGroupId2);
           serverLog(LL_VERBOSE, "-----------------------------");


           //serverLog(LL_VERBOSE, "DATAKEY2 :  %s, rowGroupId : %d  rowgroupId : %d",
           //    (char *) dataKeyString->ptr, rowGroupId, dataKeyInfo->rowGroupId);
       }
     }
```

```
    int idx =0;
    int init =0;
    for(i = 5; i < c->argc; i++){

        /*TODO - pk column check & ROW MAX LIMIT, COLUMN MAX LIMIT, */

     robj *valueObj = getDecodedObject(c->argv[i]);

     //Create field Info
     int row_idx = row_number + (idx / column_number) + 1;
     int column_idx = (idx % column_number) + 1;
     int columnvector_idx = ((row_idx -1) / server.columnvector_size + 1);
     assert(column_idx <= MAX_COLUMN_NUMBER);

     /*Todo - change vector:column*/
     robj *dataField = getDataField(column_idx, columnvector_idx);
     serverLog(LL_DEBUG, "DATAFIELD KEY = %s", (char *)dataField->ptr);
     assert(dataField != NULL);


     /*check Value Type*/
     if(!(strcmp((char *)valueObj->ptr, NULLVALUE)))
         valueObj = shared.nullValue;


     serverLog(LL_DEBUG, "insertKVpairToRelational key : %s, field : %s, value : %s",
          (char *)dataKeyString->ptr, (char *)dataField->ptr, (char *)valueObj->ptr);

     /*insert data into dict with Relational model*/
     init = insertKVpairToRelational(c, dataKeyString, dataField, valueObj, partial_flag);

     if(init)
      Enroll_queue++;

     idx++;
     insertedRow++;
     decrRefCount(dataField);
     decrRefCount(valueObj);
  }

    /*addb update row number info*/
    insertedRow /= column_number;
    incRowNumber(c->db, dataKeyInfo, insertedRow);
    partial_flag=0;

    //check validate step
  {
      int valid_result = check_insert_result(c, dataKeyInfo, dataKeyString);

  }

  serverLog(LL_VERBOSE, "---FINISH INSERT DATA---");
  serverLog(LL_VERBOSE, "INSERTED DATAKEY : %s", (char *)dataKeyString->ptr);
    int Current_rowGroupId = getRowgroupInfo(c->db, dataKeyInfo);
  serverLog(LL_VERBOSE, "CURRENT ROWGROUPID : %d", Current_rowGroupId );
    int Current_row_number = getRowNumberInfoAndSetRowNumberInfo(c->db, dataKeyInfo);
  serverLog(LL_VERBOSE, "INSERTED ROWNUMBER : %d", Current_row_number);
  serverLog(LL_VERBOSE, "----------------------");
  serverLog(LL_VERBOSE, " ");
  serverLog(LL_VERBOSE, " ");
    serverLog(LL_DEBUG,"FPWRITE COMMAND END");

    serverLog(LL_DEBUG,"DictEntry Registration in a circular queue START");

    /*Enqueue Entry*/
    if(Enroll_queue){
     if(enqueue(c->db->EvictQueue, dictFind(c->db->dict, dataKeyString->ptr)) == 0) {
       serverLog(LL_VERBOSE, "Enqueue queue : %d --- prev_row : %d --- String : %s " ,Enroll_queue, prev_row,
           dataKeyString->ptr);
       serverAssert(0);
     }
  }

    decrRefCount(dataKeyString);
    zfree(dataKeyInfo);
    addReply(c, shared.ok);
}
```

- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

```
/*parsing dataInfo*/
NewDataKeyInfo *dataKeyInfo = parsingDataKeyInfo((sds)c->argv[1]->ptr);
```

Metadict

dict

```
// addb Parsing Key information from arg[1]
NewDataKeyInfo * parsingDataKeyInfo(sds dataKeyString){

  assert(dataKeyString != NULL);
  serverLog(LL_DEBUG,"Create DATAKEYINFO START");

  NewDataKeyInfo *ret = zcalloc(sizeof(NewDataKeyInfo));
  char copyStr[MAX_TMPBUF_SIZE];
  char* saveptr = NULL;
  char* token = NULL;
  size_t size = sdslen(dataKeyString) + 1;

  assert( size == (strlen(dataKeyString ) + 1));
  assert( size < MAX_TMPBUF_SIZE);

  memcpy(copyStr, dataKeyString, size);

  //parsing the prefix
  if ((token = strtok_r(copyStr, RELMODEL_DELIMITER, &saveptr)) == NULL){
    serverLog(LL_WARNING, "Fatal: DataKey broken Error: [%s]", dataKeyString);
    zfree(ret);
    return NULL;
  }
  /*skip IDX prefix ("I") */
  if (strcasecmp(token, RELMODEL_INDEX_PREFIX) == 0 ) {
    // skip idxType field
    strtok_r(NULL, RELMODEL_DELIMITER, &saveptr);
  }
  // parsing tableId
  if ((token = strtok_r(NULL, RELMODEL_DELIMITER, &saveptr)) == NULL){
    serverLog(LL_WARNING, "Fatal: DataKey broken Error: [%s]", dataKeyString);
    zfree(ret);
    return NULL;
  }
  ret->tableId = atoi(token + strlen(RELMODEL_BRACE_PREFIX));
  //parsing partitionInfo
  if ((token = strtok_r(NULL, RELMODEL_BRACE_SUFFIX, &saveptr)) == NULL){
    serverLog(LL_WARNING, "Fatal: DataKey broken Error: [%s]", dataKeyString);
    zfree(ret);
    return NULL;
  }
  size_t part_size = strlen(token)+1;
  assert(part_size <= PARTITION_KEY_MAX_SIZE);
  memcpy(ret->partitionInfo.partitionString, token, part_size);
  ret->isPartitionString = true;

  //parsing rowgroup number
  if (saveptr != NULL && (token = strtok_r(NULL, RELMODEL_ROWGROUPID_PREFIX, &saveptr)) != NULL)
    ret->rowGroupId = atoi(token);
  else
    ret->rowGroupId = 0;

  serverLog(LL_DEBUG,"TOKEN : %s, SAVEPTR: %s, table_number : %d, partitionInfo : %s, rowgroup : %d",
              token, saveptr, ret->tableId,ret->partitionInfo.partitionString, ret->rowGroupId);
  return ret;
}
```

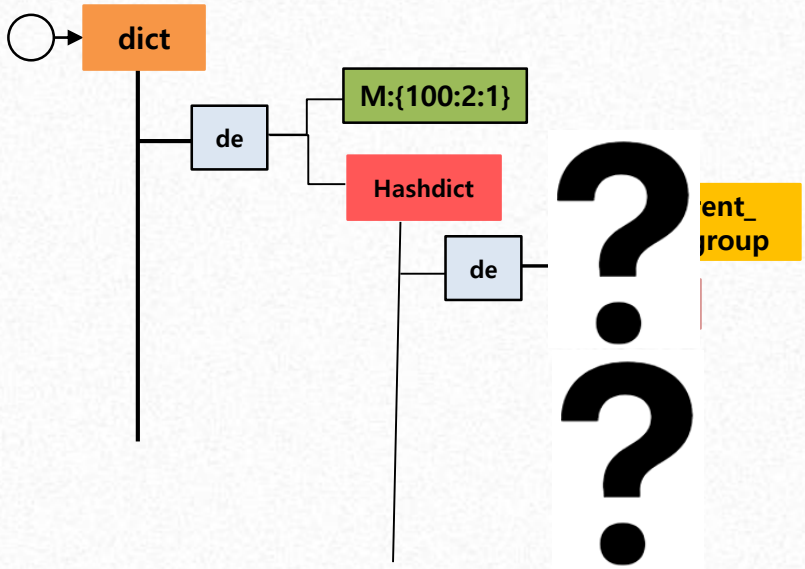
- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

```
/*get rowgroup info from Metadict*/
int rowGroupId = getRowgroupInfo(c->db, dataKeyInfo);
serverLog(LL_DEBUG, "rowGroupId = %d", rowGroupId);
```

Metadict



dict

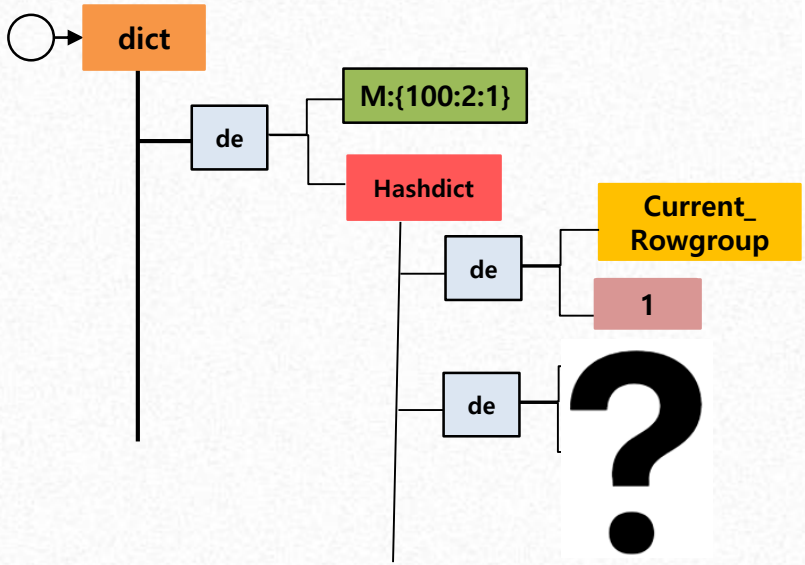
```
/*addb get RowgroupInfo from Metadict*/
int getRowGroupInfoAndSetRowGroupInfo(redisDb *db, NewDataKeyInfo *data
  char tmp[SDS_DATA_KEY_MAX];
  int rowgroup = 0;
  sds metaKey = sdsnewlen("", SDS_DATA_KEY_MAX);
  setMetaKeyForRowgroup(dataKeyInfo, metaKey);

  robj *metaHashdictObj = lookupSDSKeyForMetadict(db, metaKey);
  if(metaHashdictObj == NULL){
    serverLog(LL_DEBUG, "METAHASHDICT OBJ IS NULL");
  }
  else{
    serverLog(LL_DEBUG, "METAHASHDICT OBJ IS NOTNULL");
  }

  robj *metaField = shared.integers[0];
  rowgroup = lookupCompInfoForMeta(metaHashdictObj, metaField);
  dataKeyInfo->rowGroupId = rowgroup;
    sdsfree(metaKey);
  return rowgroup;
}


int getRowgroupInfo(redisDb *db, NewDataKeyInfo *dataKeyInfo){

    int rowgroup = getRowGroupInfoAndSetRowGroupInfo(db,dataKeyInfo);
    if(rowgroup == 0){
      serverLog(LL_DEBUG, "ROWGROUP 0 CASE OCCUR");
     rowgroup = IncRowgroupIdAndModifyInfo(db, dataKeyInfo, 1);

  }
    return rowgroup;
}
```

- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

```
/*get rownumber info from Metadict*/
int row_number = getRowNumberInfoAndSetRowNumberInfo(c->db, dataKeyInfo);
serverLog(LL_DEBUG, "rowNumber = %d", row_number);
int prev_row = row_number;

/*set rowNumber Info to Metadict*/
if(row_number == 0 ){
 incRowNumber(c->db, dataKeyInfo, 0);
}
```



dict

- RowgroupID =1
- Rownumber = 0

```
/*addb get RowNumberInfo from Metadict*/
int getRowNumberInfoAndSetRowNumberInfo(redisDb *db, NewDataKeyInfo *dataKeyInfo){
  char tmp[SDS_DATA_KEY_MAX];
  int rowNumber= 0;
  sds metaKey = sdsnewlen("", SDS_DATA_KEY_MAX);// sdsnewlen(tmp, sizeof(tmp) //sdsnew(tmp)
  setMetaKeyForRowgroup(dataKeyInfo, metaKey);

  robj *metaHashdictObj = lookupSDSKeyForMetadict(db, metaKey);
  if(metaHashdictObj == NULL){
    serverLog(LL_DEBUG, "METAHASHDICT OBJ IS NULL");
  }
  else{
    serverLog(LL_DEBUG, "METAHASHDICT OBJ IS NOTNULL");
  }

  robj *metaField  = createStringObjectFromLongLong((long long) dataKeyInfo->rowGroupId);
  rowNumber = lookupCompInfoForRowNumberInMeta(metaHashdictObj, metaField);
  dataKeyInfo->row_number = rowNumber;
  decrRefCount(metaField);
    sdsfree(metaKey);
  return rowNumber;
}
```

```
/*check rowgroup size*/
if(row_number >= server.rowgroup_size){
  rowGroupId = IncRowgroupIdAndModifyInfo(c->db, dataKeyInfo, 1);
  row_number = 0;
  Enroll_queue = 1;
}
```

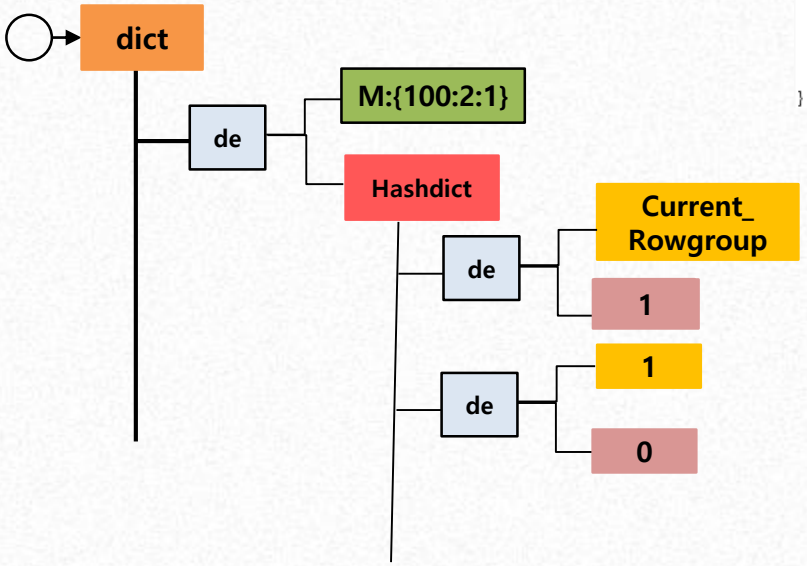
- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

```
robj *dataKeyString = NULL;
dataKeyString = generateDataKey(dataKeyInfo);
//serverLog(LL_VERBOSE, "DATAKEY1 :  %s", (char *) dataKeyString->ptr);
```

Metadict



dict

- RowgroupID =1
- Rownumber = 0

```
robj *generateDataKey(NewDataKeyInfo *dataKeyInfo) {
    sds dataKeySds = generateDataKeySds(dataKeyInfo);
    robj *dataKeyObj = createStringObject(dataKeySds, sdslen(dataKeySds));
    sdsfree(dataKeySds);
    return dataKeyObj;
}

sds generateDataKeySds(NewDataKeyInfo *dataKeyInfo) {
    char dataKey[DATA_KEY_MAX_SIZE];
  if(dataKeyInfo->isPartitionString){
    sprintf(dataKey, "%s:{%d:%s}:%s%d",
        RELMODEL_DATA_PREFIX, dataKeyInfo->tableId, dataKeyInfo->partitionInfo.partitionString,
        RELMODEL_ROWGROUPID_PREFIX,
        dataKeyInfo->rowGroupId);
  }else{
    char *p = (char *)dataKey;
    sprintf(p, "%s:{%d", RELMODEL_DATA_PREFIX, dataKeyInfo->tableId);
    p += strlen(RELMODEL_DATA_PREFIX);
    p += 2; // for :{
    p += digits10(dataKeyInfo->tableId);
    for(int i=0;i<dataKeyInfo->partitionCnt;i++){
      sprintf(p, ":%llu", dataKeyInfo->partitionInfo.partitionInt[i]);
      p += digits10(dataKeyInfo->partitionInfo.partitionInt[i]);
      p += 1; //for :
    }
    sprintf(p, "}:%s%d", RELMODEL_ROWGROUPID_PREFIX, dataKeyInfo->rowGroupId);
  }
  serverLog(LL_DEBUG, "DATAKEY :  %s", (char *)dataKey);
    return sdsnew(dataKey);
}
```

Datakey ➔ D:{100:2:1}:G:1

- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 1 1 3 4

dict

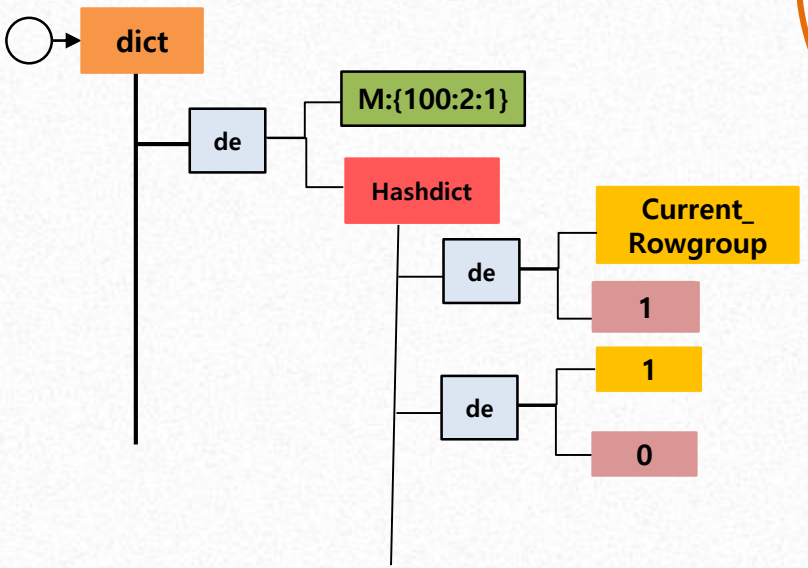
- RowgroupID =1
- Rownumber = 0

Datakey ➔ D:{100:2:1}:G:1

```
int idx =0;
int init =0;
for(i = 5; i < c->argc; i++){

    /*TODO - pk column check & ROW MAX LIMIT, COLUMN MAX LIMIT, */

 robj *valueObj = getDecodedObject(c->argv[i]);

 //Create field Info
 int row_idx = row_number + (idx / column_number) + 1;
 int column_idx = (idx % column_number) + 1;
 int columnvector_idx = ((row_idx -1) / server.columnvector_size + 1);
 assert(column_idx <= MAX_COLUMN_NUMBER);

 /*Todo - change vector:column*/
 robj *dataField = getDataField(column_idx, columnvector_idx);
 serverLog(LL_DEBUG, "DATAFIELD KEY = %s", (char *)dataField->ptr);
 assert(dataField != NULL);


 /*check Value Type*/
 if(!(strcmp((char *)valueObj->ptr, NULLVALUE)))
     valueObj = shared.nullValue;
```

Field ➔ 1:1

Metadict



- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

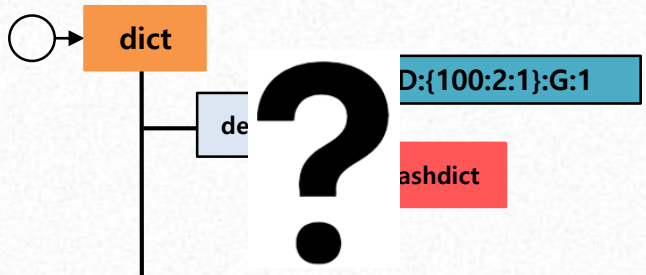
```
/*insert data into dict with Relational model*/
init = insertKVpairToRelational(c, dataKeyString, dataField, valueObj, partial_flag);
```

```
/*addb Data Insertion func*/
int insertKVpairToRelational(client *c, robj *dataKeyString, robj *dataField, robj *valueObj, int partial_flag){

  assert(dataKeyString != NULL);
  assert(dataField != NULL);

  robj *dataHashdictObj = NULL;
  int init = 0;

  if( (dataHashdictObj = lookupDictAndGetHashdictObj(c,dataKeyString, &init)) == NULL ){

    serverLog(LL_WARNING, "Can't Find dataHashdict in dict, Because of Creation Error");
    serverPanic("insertKVpairToRelational ERROR");
  }

  dict *hashDict = (dict *)dataHashdictObj->ptr;
  dictEntry *de = NULL;

  if( partial_flag == 1 || ((de = dictFind(hashDict, dataField->ptr)) == NULL)){


     //create vector
        Vector *v = zmalloc(sizeof(Vector));
        vectorTypeInit(v, STL_TYPE_SDS);
        assert(v->size == 0);
        assert(v->count == 0);

        vectorAdd(v, sdsdup(valueObj->ptr));
        robj *columnVectorObj = createObject(OBJ_VECTOR, v);

        int ret = dictAdd(hashDict,sdsdup(dataField->ptr),  columnVectorObj);

        if(!ret){
              serverLog(LL_DEBUG, "Create New Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
                  , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
        else {
              serverLog(LL_WARNING, "Create New Vector & DATA INSERTION FAIL");
              serverPanic("Create New Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  else {

    //get vector object & append value
    robj *VectorObj = dictGetVal(de);
    assert(VectorObj->type ==OBJ_VECTOR);
    Vector *v = (Vector *)VectorObj->ptr;
    vectorAdd(v, sdsdup(valueObj->ptr));

    int number = v->count;
    //check append result
    if(!strcmp(vectorGet(v, number-1), (sds)valueObj->ptr)){
      serverLog(LL_DEBUG, "Append Existed Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
          , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
    else {
      serverLog(LL_WARNING, "Append Vector & DATA INSERTION FAIL");
      serverPanic("Append Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  notifyKeyspaceEvent(NOTIFY_HASH,"hset", dataKeyString,c->db->id);
  server.dirty++;
  return init;
}
```

dict

- RowgroupID =1
- Rownumber = 0

Datakey ➔ D:{100:2:1}:G:1
Field ➔ 1:1



- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

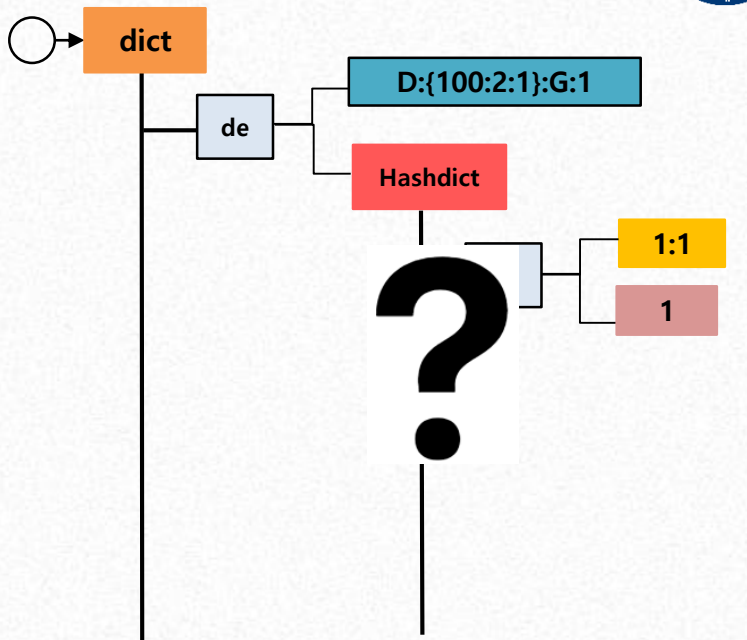
```
/*insert data into dict with Relational model*/
init = insertKVpairToRelational(c, dataKeyString, dataField, valueObj, partial_flag);
```

```
/*addb Data Insertion func*/
int insertKVpairToRelational(client *c, robj *dataKeyString, robj *dataField, robj *valueObj, int partial_flag){

  assert(dataKeyString != NULL);
  assert(dataField != NULL);

  robj *dataHashdictObj = NULL;
  int init = 0;

  if( (dataHashdictObj = lookupDictAndGetHashdictObj(c,dataKeyString, &init)) == NULL ){

    serverLog(LL_WARNING, "Can't Find dataHashdict in dict, Because of Creation Error");
    serverPanic("insertKVpairToRelational ERROR");
  }

  dict *hashDict = (dict *)dataHashdictObj->ptr;
  dictEntry *de = NULL;

  if( partial_flag == 1 || ((de = dictFind(hashDict, dataField->ptr)) == NULL)){


     //create vector
        Vector *v = zmalloc(sizeof(Vector));
        vectorTypeInit(v, STL_TYPE_SDS);
        assert(v->size == 0);
        assert(v->count == 0);

        vectorAdd(v, sdsdup(valueObj->ptr));
        robj *columnVectorObj = createObject(OBJ_VECTOR, v);

        int ret = dictAdd(hashDict,sdsdup(dataField->ptr),  columnVectorObj);

        if(!ret){
              serverLog(LL_DEBUG, "Create New Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
                  , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
        else {
              serverLog(LL_WARNING, "Create New Vector & DATA INSERTION FAIL");
              serverPanic("Create New Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  else {

    //get vector object & append value
    robj *VectorObj = dictGetVal(de);
    assert(VectorObj->type ==OBJ_VECTOR);
    Vector *v = (Vector *)VectorObj->ptr;
    vectorAdd(v, sdsdup(valueObj->ptr));

    int number = v->count;
    //check append result
    if(!strcmp(vectorGet(v, number-1), (sds)valueObj->ptr)){
      serverLog(LL_DEBUG, "Append Existed Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
          , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
    else {
      serverLog(LL_WARNING, "Append Vector & DATA INSERTION FAIL");
      serverPanic("Append Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  notifyKeyspaceEvent(NOTIFY_HASH,"hset", dataKeyString,c->db->id);
  server.dirty++;
  return init;
}
```

dict

- RowgroupID =1
- Rownumber = 0

Datakey ➔ D:{100:2:1}:G:1
Field ➔ 1:1



- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

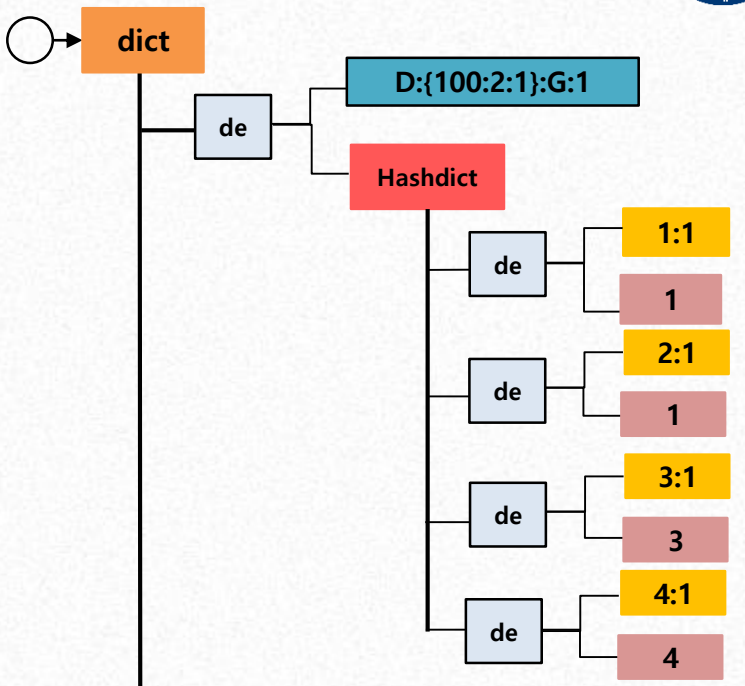
Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

```
/
i                                                                    g);
    /*addb update row number info*/
    insertedRow /= column_number;
    incRowNumber(c->db, dataKeyInfo, insertedRow);
```

```
/*addb Data Insertion func*/
int insertKVpairToRelational(client *c, robj *dataKeyString, robj *dataField, robj *valueObj, int partial_flag){

  assert(dataKeyString != NULL);
  assert(dataField != NULL);

  robj *dataHashdictObj = NULL;
  int init = 0;

  if( (dataHashdictObj = lookupDictAndGetHashdictObj(c,dataKeyString, &init)) == NULL ){

    serverLog(LL_WARNING, "Can't Find dataHashdict in dict, Because of Creation Error");
    serverPanic("insertKVpairToRelational ERROR");
  }

  dict *hashDict = (dict *)dataHashdictObj->ptr;
  dictEntry *de = NULL;

  if( partial_flag == 1 || ((de = dictFind(hashDict, dataField->ptr)) == NULL)){


     //create vector
        Vector *v = zmalloc(sizeof(Vector));
        vectorTypeInit(v, STL_TYPE_SDS);
        assert(v->size == 0);
        assert(v->count == 0);

        vectorAdd(v, sdsdup(valueObj->ptr));
        robj *columnVectorObj = createObject(OBJ_VECTOR, v);

        int ret = dictAdd(hashDict,sdsdup(dataField->ptr),  columnVectorObj);

        if(!ret){
              serverLog(LL_DEBUG, "Create New Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
                  , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
        else {
              serverLog(LL_WARNING, "Create New Vector & DATA INSERTION FAIL");
              serverPanic("Create New Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  else {

    //get vector object & append value
    robj *VectorObj = dictGetVal(de);
    assert(VectorObj->type ==OBJ_VECTOR);
    Vector *v = (Vector *)VectorObj->ptr;
    vectorAdd(v, sdsdup(valueObj->ptr));

    int number = v->count;
    //check append result
    if(!strcmp(vectorGet(v, number-1), (sds)valueObj->ptr)){
      serverLog(LL_DEBUG, "Append Existed Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
          , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
    else {
      serverLog(LL_WARNING, "Append Vector & DATA INSERTION FAIL");
      serverPanic("Append Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  notifyKeyspaceEvent(NOTIFY_HASH,"hset", dataKeyString,c->db->id);
  server.dirty++;
  return init;
}
```

dict

- RowgroupID =1
- Rownumber = 0



- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

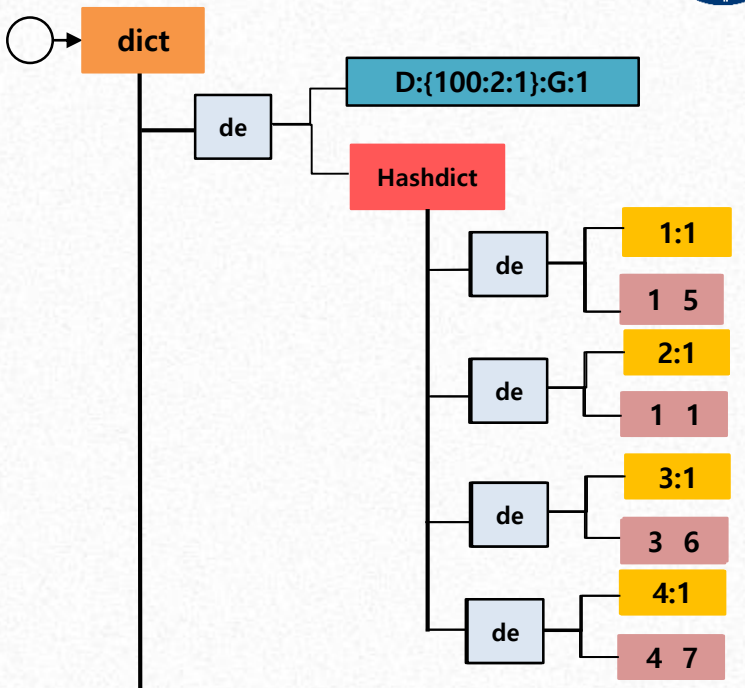
Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

```
/*insert data into dict with Relational model*/
init = insertKVpairToRelational(c, dataKeyString, dataField, valueObj, partial_flag);
```

```
/*addb Data Insertion func*/
int insertKVpairToRelational(client *c, robj *dataKeyString, robj *dataField, robj *valueObj, int partial_flag){

  assert(dataKeyString != NULL);
  assert(dataField != NULL);

  robj *dataHashdictObj = NULL;
  int init = 0;

  if( (dataHashdictObj = lookupDictAndGetHashdictObj(c,dataKeyString, &init)) == NULL ){

    serverLog(LL_WARNING, "Can't Find dataHashdict in dict, Because of Creation Error");
    serverPanic("insertKVpairToRelational ERROR");
  }

  dict *hashDict = (dict *)dataHashdictObj->ptr;
  dictEntry *de = NULL;

  if( partial_flag == 1 || ((de = dictFind(hashDict, dataField->ptr)) == NULL)){


     //create vector
        Vector *v = zmalloc(sizeof(Vector));
        vectorTypeInit(v, STL_TYPE_SDS);
        assert(v->size == 0);
        assert(v->count == 0);

        vectorAdd(v, sdsdup(valueObj->ptr));
        robj *columnVectorObj = createObject(OBJ_VECTOR, v);

        int ret = dictAdd(hashDict,sdsdup(dataField->ptr),  columnVectorObj);

        if(!ret){
              serverLog(LL_DEBUG, "Create New Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
                  , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
        else {
              serverLog(LL_WARNING, "Create New Vector & DATA INSERTION FAIL");
              serverPanic("Create New Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  else {

    //get vector object & append value
    robj *VectorObj = dictGetVal(de);
    assert(VectorObj->type ==OBJ_VECTOR);
    Vector *v = (Vector *)VectorObj->ptr;
    vectorAdd(v, sdsdup(valueObj->ptr));

    int number = v->count;
    //check append result
    if(!strcmp(vectorGet(v, number-1), (sds)valueObj->ptr)){
      serverLog(LL_DEBUG, "Append Existed Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
          , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
    else {
      serverLog(LL_WARNING, "Append Vector & DATA INSERTION FAIL");
      serverPanic("Append Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  notifyKeyspaceEvent(NOTIFY_HASH,"hset", dataKeyString,c->db->id);
  server.dirty++;
  return init;
}
```

dict

- RowgroupID =1
- Rownumber = 1

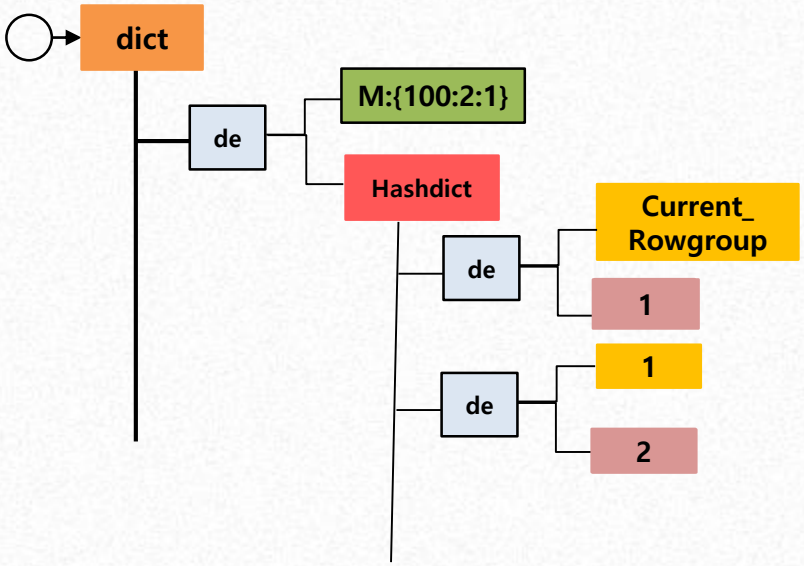


- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

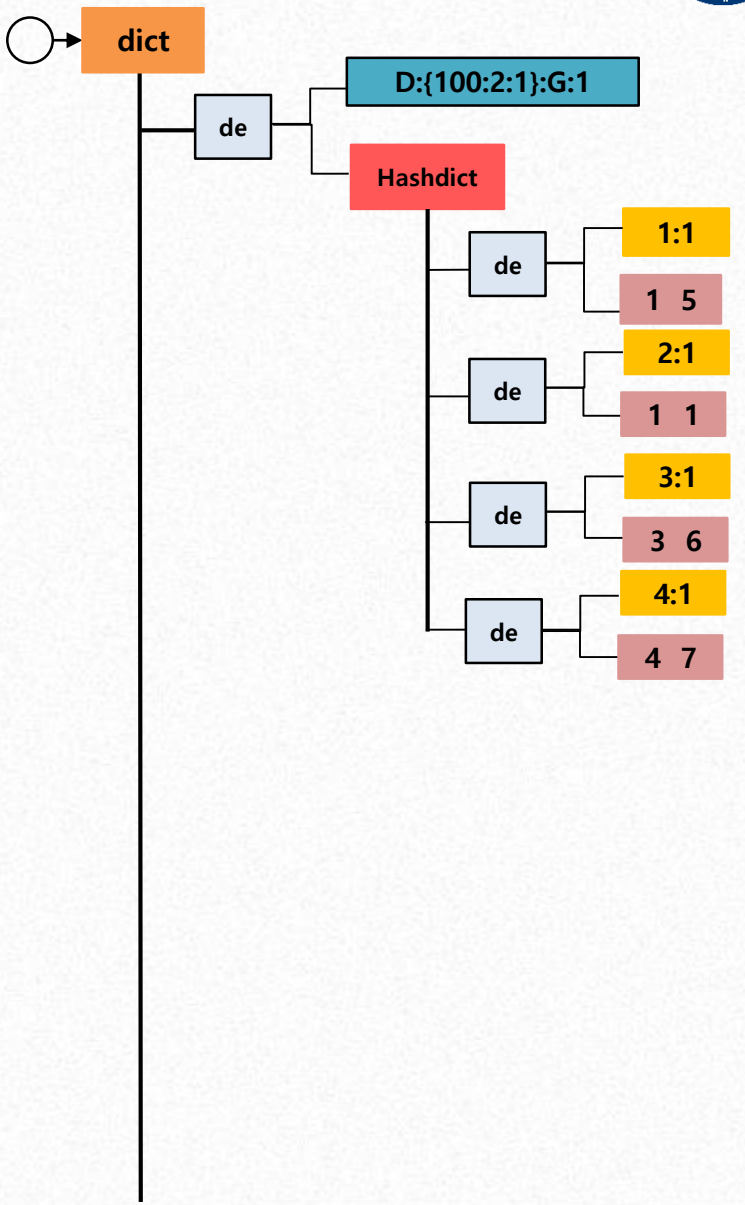
Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 5 1 6 7

Metadict



dict

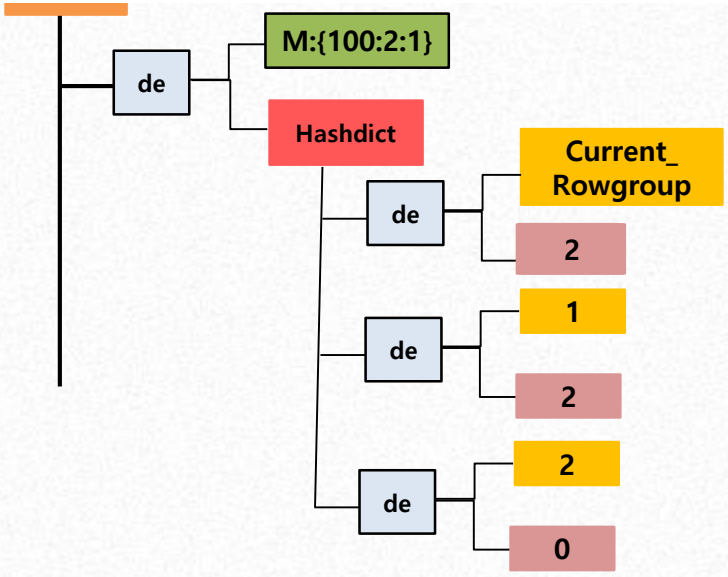


- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 8 1 9 10

```
/*check rowgroup size*/
if(row_number >= server.rowgroup_size){
  rowGroupId = IncRowgroupIdAndModifyInfo(c->db, dataKeyInfo, 1);
  row_number = 0;
  Enroll_queue = 1;
}
```



dict

- RowgroupID =2
- Rownumber = 0

dict

```
int IncRowgroupIdAndModifyInfo(redisDb *db, NewDataKeyInfo *dataKeyInfo, int cnt){
  int result = incRowgroupId(db, dataKeyInfo, cnt);
  dataKeyInfo->rowGroupId = result;
  return result;
}

int incRowgroupId(redisDb *db, NewDataKeyInfo *dataKeyInfo, int inc_number){
  robj *rowgroupKey= generateRgIdKeyForRowgroup(dataKeyInfo);
  robj *metaRgField = shared.integers[0];

  int ret = IncDecCount(db, rowgroupKey, metaRgField, (long long) inc_number);
  decrRefCount(rowgroupKey);
  return ret;
}

//TO-DO MODIFY LATER -HS
int IncDecCount(redisDb *db, robj *key, robj *field, long long cnt){  // int flags ??

    long long value, oldvalue;
    robj *o, *new, *cur_obj;

    o = lookupKeyWriteForMetadict(db,key);
    if(o == NULL){
     o = createHashObject();  //ziplist object
     dbAddForMetadict(db, key, o);
   }
    else{

     if(o->type != OBJ_HASH){
       serverLog(LL_ERROR, "The value of the Metadict, hashdictobj's type, must be hash ");
       assert(0);
     }
   }

    if((cur_obj = hashTypeGetValueRObject(o, field)) != NULL){
     if (getLongLongFromObject(cur_obj, &value) != C_OK) {
       serverLog(LL_ERROR, "the value of meta must be INTEGER");
       assert(0);
     }
    decrRefCount(cur_obj);
   }
    else{
     value = 0;
   }
    oldvalue = value;
    if ((cnt < 0 && oldvalue < 0 && cnt < (LLONG_MIN-oldvalue)) ||
        (cnt > 0 && oldvalue > 0 && cnt > (LLONG_MAX-oldvalue))) {
     serverLog(LL_ERROR,"increment or decrement would overflow");
     assert(0);
   }
    value += cnt;

    new = createStringObjectFromLongLong(value);
    hashTypeTryObjectEncoding(o,&field,NULL);
    hashTypeSetWithNoFlags(o, field, new);
    decrRefCount(new);

    signalModifiedKey(db,key);
    notifyKeyspaceEvent(NOTIFY_HASH,"Hash incrby",key,db->id);
    server.dirty++;

    return value;
}
```

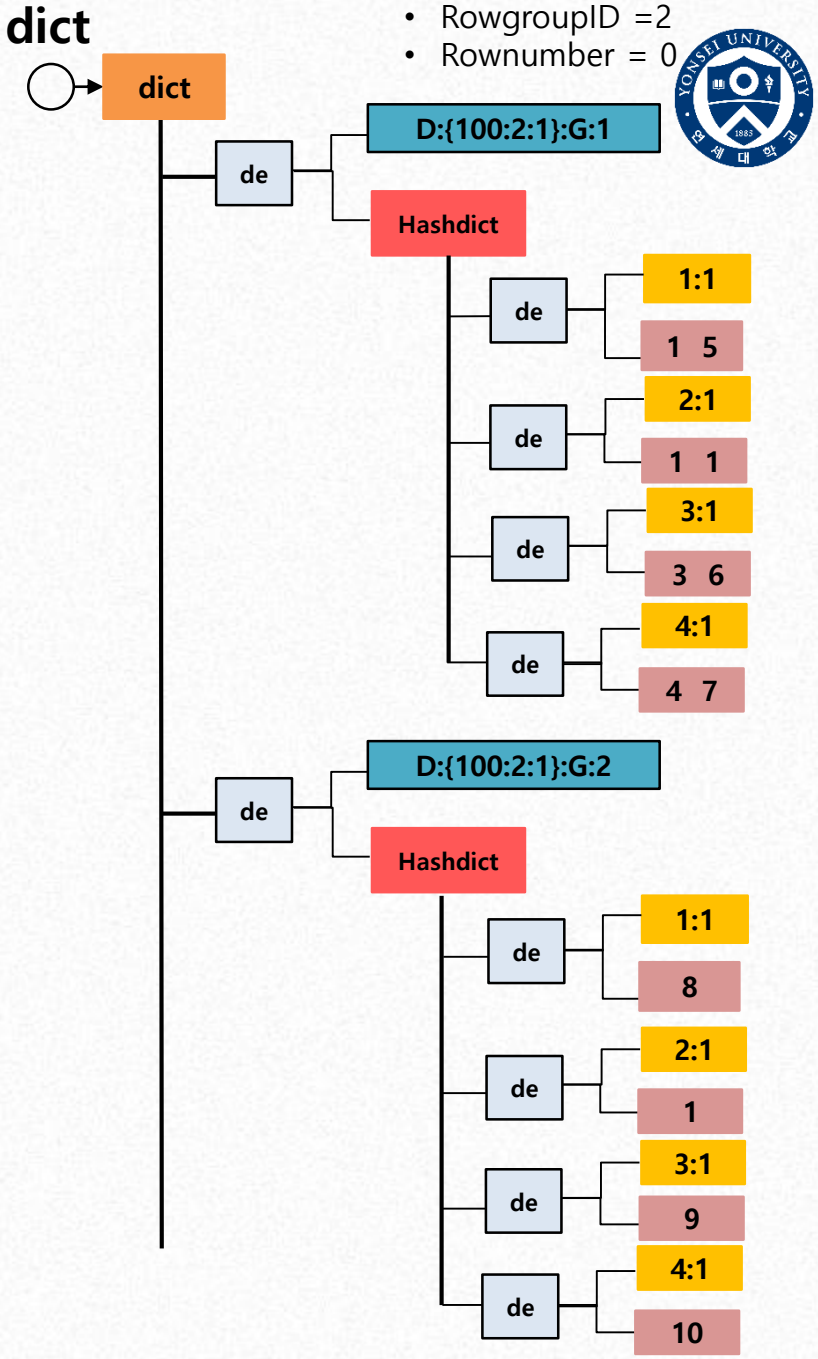
- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

Datakey ➔ D:{100:2:1}:G:2
Field ➔ 1:1

Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 8 1 9 10

```
/*addb Data Insertion func*/
int insertKVpairToRelational(client *c, robj *dataKeyString, robj *dataField, robj *valueObj, int partial_flag){

  assert(dataKeyString != NULL);
  assert(dataField != NULL);

  robj *dataHashdictObj = NULL;
  int init = 0;

  if( (dataHashdictObj = lookupDictAndGetHashdictObj(c,dataKeyString, &init)) == NULL ){

    serverLog(LL_WARNING, "Can't Find dataHashdict in dict, Because of Creation Error");
    serverPanic("insertKVpairToRelational ERROR");
  }

  dict *hashDict = (dict *)dataHashdictObj->ptr;
  dictEntry *de = NULL;

  if( partial_flag == 1 || ((de = dictFind(hashDict, dataField->ptr)) == NULL)){


     //create vector
        Vector *v = zmalloc(sizeof(Vector));
        vectorTypeInit(v, STL_TYPE_SDS);
        assert(v->size == 0);
        assert(v->count == 0);

        vectorAdd(v, sdsdup(valueObj->ptr));
        robj *columnVectorObj = createObject(OBJ_VECTOR, v);

        int ret = dictAdd(hashDict,sdsdup(dataField->ptr),  columnVectorObj);

        if(!ret){
              serverLog(LL_DEBUG, "Create New Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
                  , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
        else {
              serverLog(LL_WARNING, "Create New Vector & DATA INSERTION FAIL");
              serverPanic("Create New Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  else {

    //get vector object & append value
    robj *VectorObj = dictGetVal(de);
    assert(VectorObj->type ==OBJ_VECTOR);
    Vector *v = (Vector *)VectorObj->ptr;
    vectorAdd(v, sdsdup(valueObj->ptr));

    int number = v->count;
    //check append result
    if(!strcmp(vectorGet(v, number-1), (sds)valueObj->ptr)){
      serverLog(LL_DEBUG, "Append Existed Vector & DATA INSERTION SUCCESS. dataKey : %s, dataField : %s, value :%s"
          , (char *)dataKeyString->ptr, (char*)dataField->ptr, (char *)valueObj->ptr);
    }
    else {
      serverLog(LL_WARNING, "Append Vector & DATA INSERTION FAIL");
      serverPanic("Append Vector & DATA INSERTION ERROR in insertKVpairToRelational");
    }
  }
  notifyKeyspaceEvent(NOTIFY_HASH,"hset", dataKeyString,c->db->id);
  server.dirty++;
  return init;
}
```



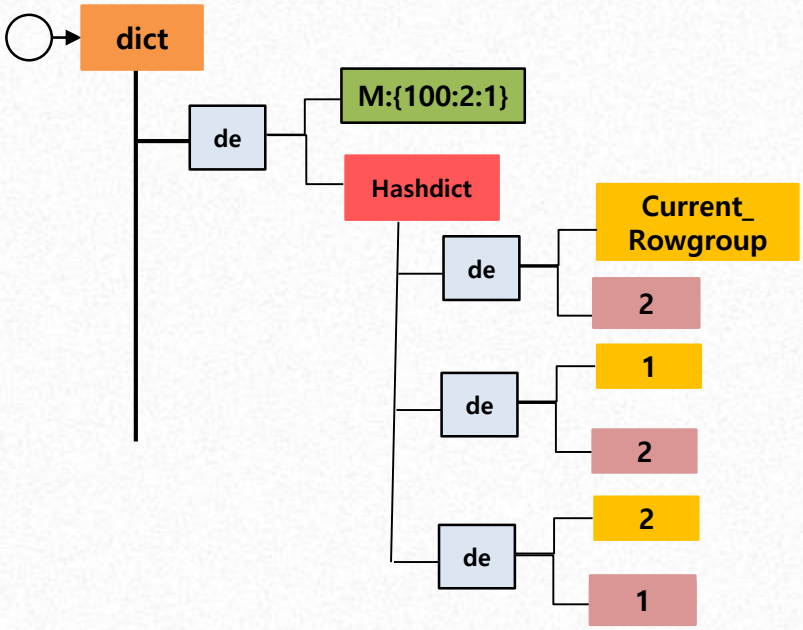
- RowgroupID =2
- Rownumber = 0

- TableID ➔ 100
- Rowgroup size ➔ 2
- PartitionInfo ➔ 2:1
- Vector size ➔ 2

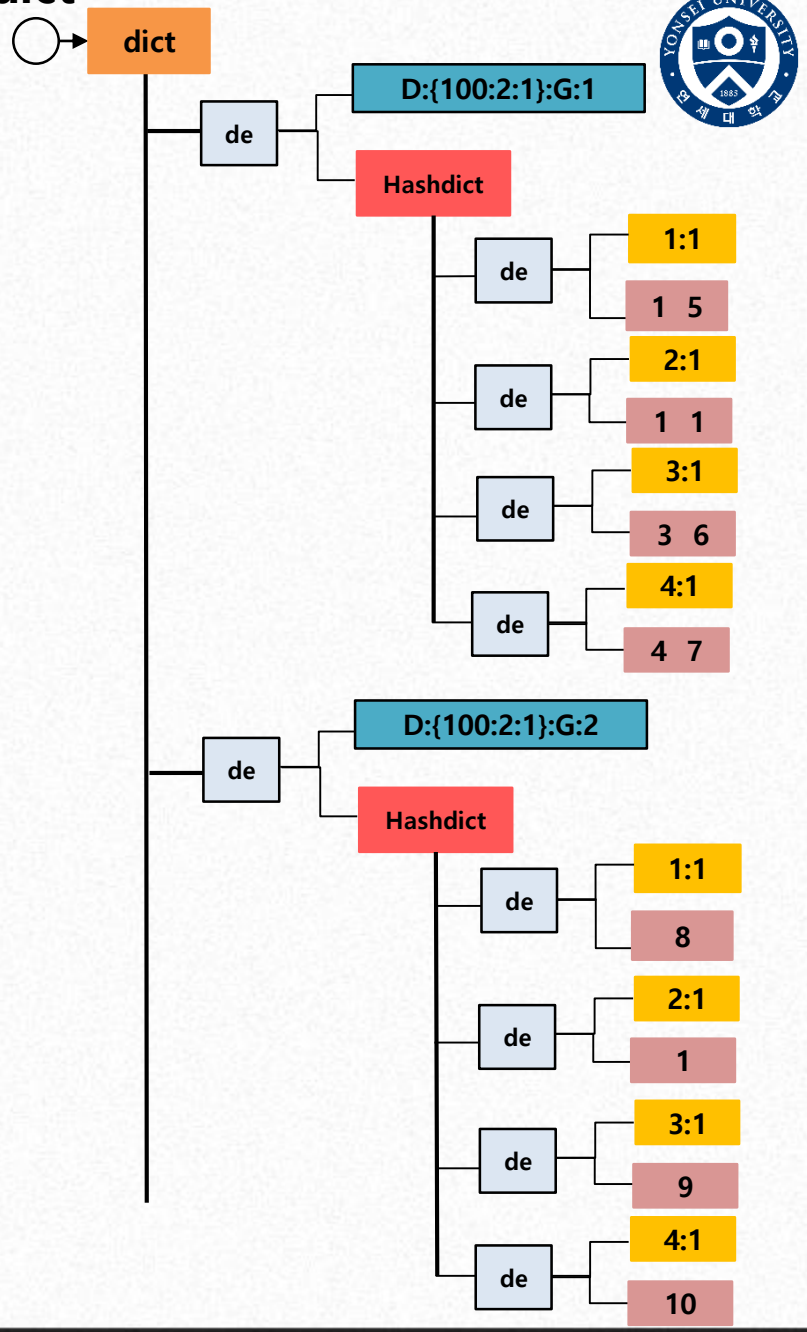
Column1	Column2	Column3	Column4
1	1	3	4
5	1	6	7
8	1	9	10

FPWRITE D:{100:2:1} 2:1 4 0 8 1 9 10

Metadict



dict



- **RocksDB 관계형 모델 구조 – Tiering에서 설명…**