ADDB: FPSCAN

연세대학교 컴퓨터과학과 박상현

2019년 7월

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

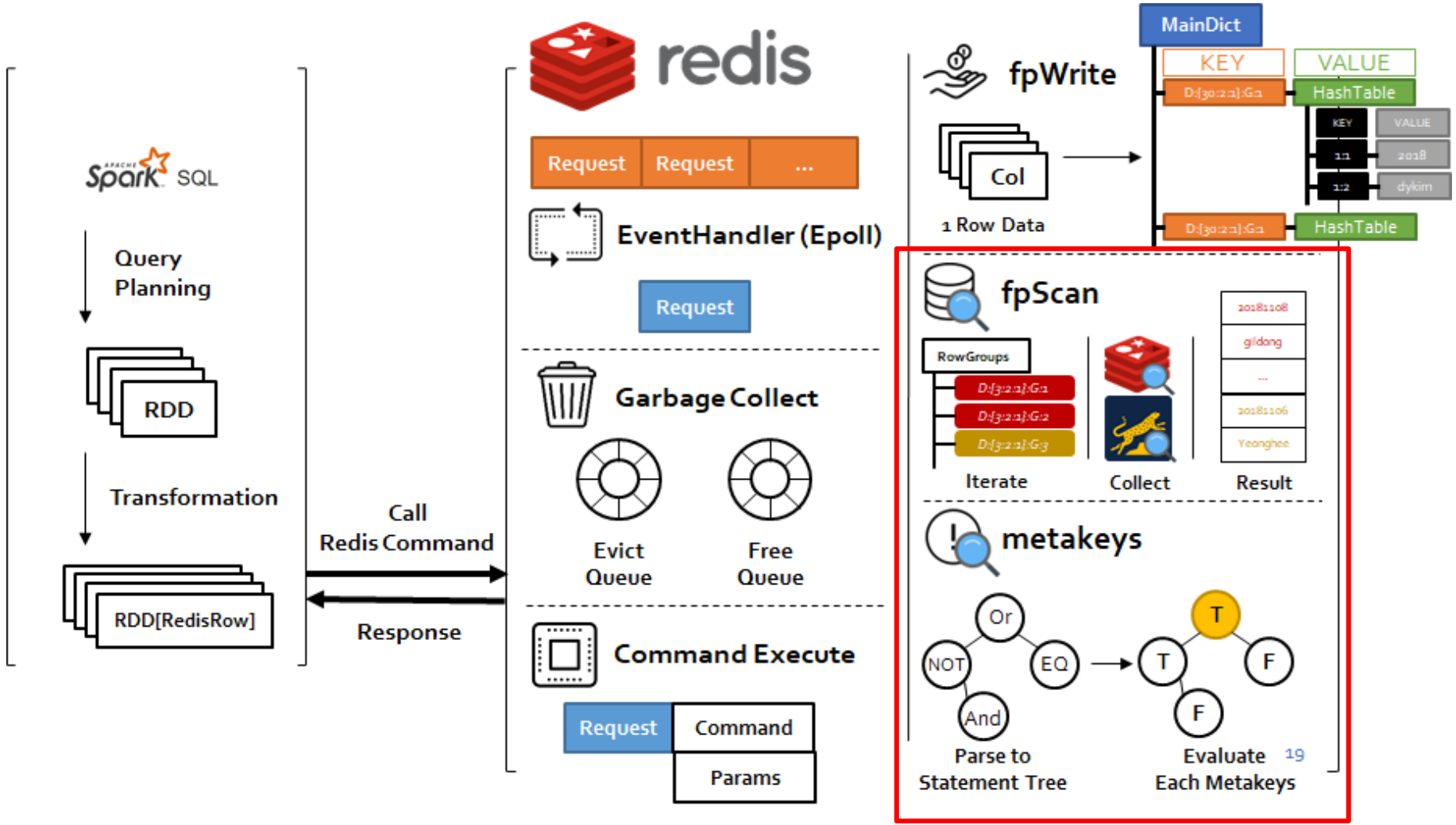
과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

정보통신기술진흥센터
Institute for Information & communications Technology Promotion

Contents


Spark SQL
Query Planning
RDD
Transformation
RDD[RedisRow]
Call Redis Command
Response
redis
Request
Request
...
EventHandler (Epoll)
Request
Garbage Collect
Evict Queue
Free Queue
Command Execute
Request
Command
Params
fpWrite
Col
1 Row Data
MainDict
KEY
VALUE
HashTable
KEY
VALUE
1:1
2018
1:2
dykim
HashTable
fpScan
RowGroups
Iterate
Collect
Result
20181108
gildong
...
20181106
Yeonghee
metakeys
Or
NOT
EQ
And
Parse to Statement Tree
Evaluate Each Metakeys
19

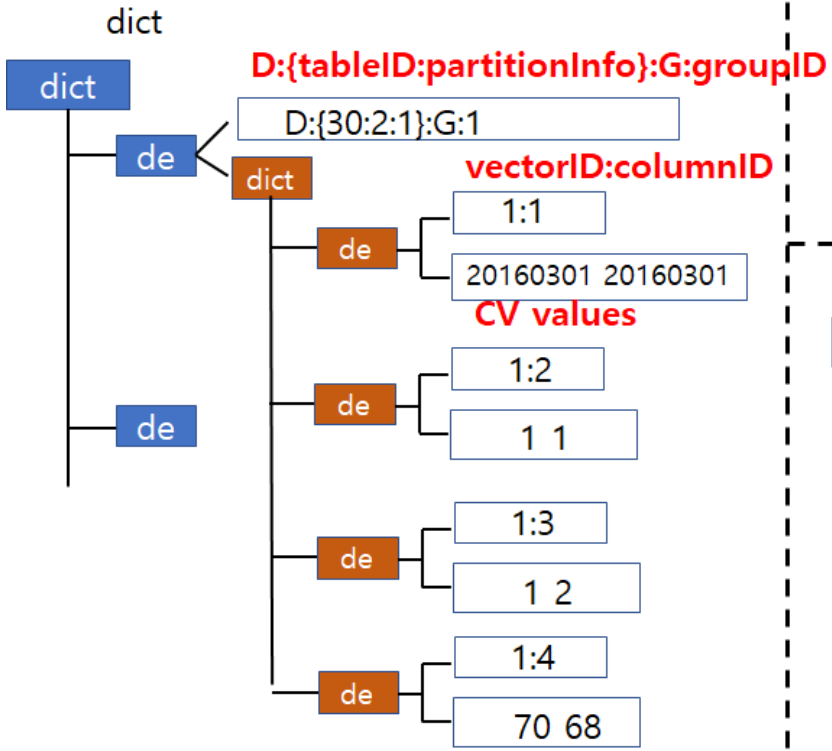
Remind...

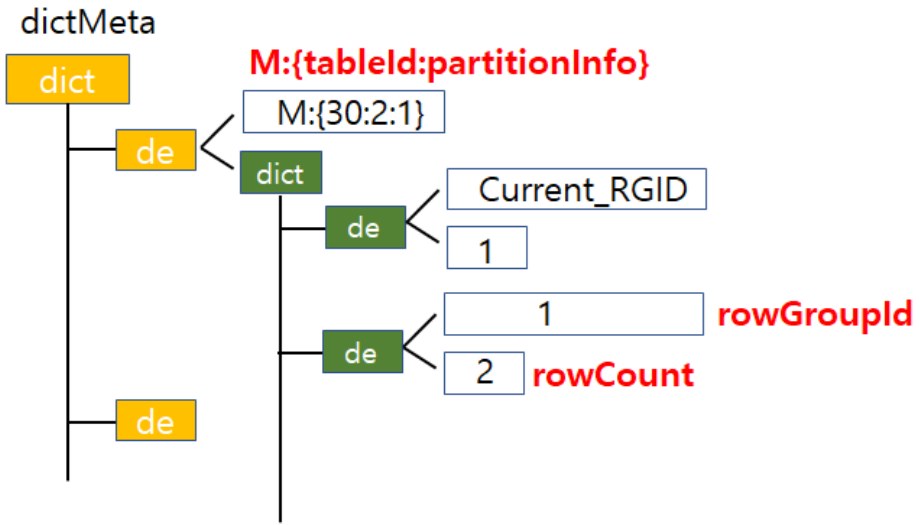
vectorId
→ rowGroupSize와 vectorSize에 의해 결정됨

*vectorSize = 2
*rowGroupsize=4

TableId = 30 partitionInfo : 2:1

	Date	Enb_id	Cell_id	cei
vectorID = 1	20160301	1	1	70
	20160301	1	2	68
vectorID = 2	20160301	1	3	42
	20160301	1	4	12
	20160301	1	5	99
	20160301	1	6	22





fpScan

- Redis 및 RocksDB에서 다음 조건인 data들을 조회함
  - Table ID
  - Partition
  - Columns

- Command Parameters
  - Key
    - ex) D:{tableId:partitionInfo}
  - Column IDs
    - 확인하고 싶은 column ID들
    - ex) 1,4,6

- Return (To Client)
  - ex) redis-cli> fpScan D:{32:1:0} 1,4,6

20190130, Yonsei, 최원기
20190130, Yonsei, 성한승
20190130, Yonsei, 이지환
20190130, Yonsei, 김도영
20190130, Yonsei, 이지은
...

fpScan - Code Structure

fpScanCommand()

createScanParameter()

- Client의 Parameter들을 Parsing
- ScanParameter Object 생성

Parse

populateScanParameter()

- ScanParameter에 metadict data를 populate

Populate

scanDataFromADDB()

- ScanParameter로 Redis와 RocksDB에서 Data-Collect
- Return results to Client

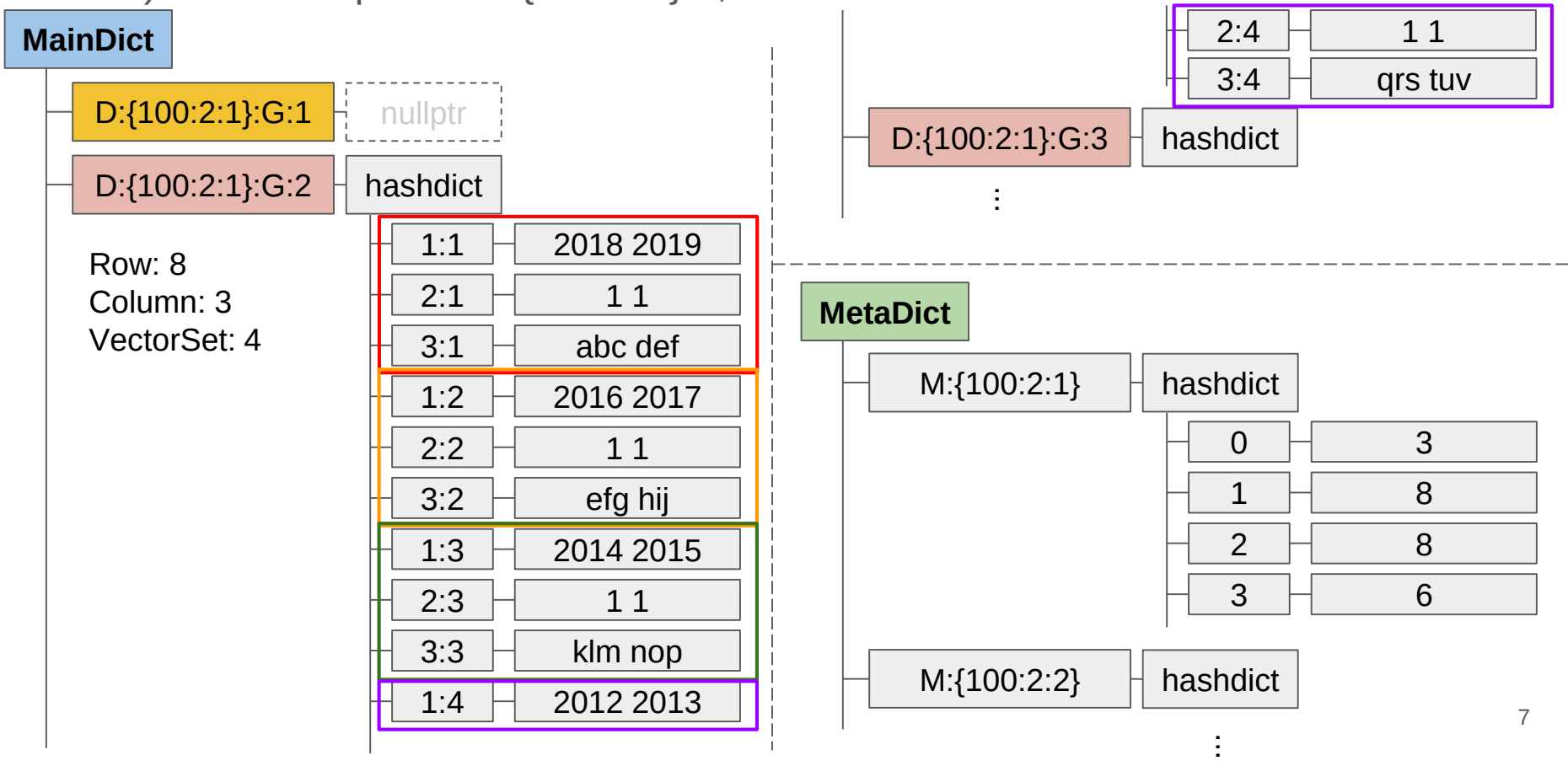
Iterate & Collect

```
/*
 * fpScanCommand
 * Scan data from the database(Redis & RocksDB)
 * --- Parameters ---
 *  arg1: Key(Table ID & PartitionInfo ID)
 *  arg2: Column IDs to find
 *
 * --- Usage Examples ---
 *  Parameters:
 *      key: "D:{3:1:2}"
 *          tableId: "3"
 *          partitionInfoId: "1:2"
 *      columnIds: ["2", "3", "4"]
 *  Command:
 *      redis-cli> FPSCAN D:{3:2:1} 2,3,4
 *  Results:
 *      redis-cli> "20180509"
 *      redis-cli> "Do young Kim"
 *      redis-cli> "Yonsei Univ"
 *      ...
 */
void fpScanCommand(client *c) {
    serverLog(LL_DEBUG, "FPSCAN COMMAND START");

    /*Creates scan parameters*/
    ScanParameter *scanParam = createScanParameter(c);

    /*Populates row group information to scan parameters*/
    int totalDataCount = populateScanParameter(c->db, scanParam);
    serverLog(LL_DEBUG, "total data count: %d", totalDataCount);

    /*Non-Vector response version*/
    void *replylen = addDeferredMultiBulkLength(c);
    size_t numreplies = scanDataFromADDB_non_vector(c, c->db, scanParam);
    freeScanParameter(scanParam);
    setDeferredMultiBulkLength(c, replylen, numreplies);
}
```

fpScan - Example

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3



fpScan - Code Structure

fpScanCommand()

createScanParameter()

- Client의 Parameter들을 Parsing
- ScanParameter Object 생성

Parse

populateScanParameter()

- ScanParameter에 metadict data를 populate

Populate

scanDataFromADDB()

- ScanParameter로 Redis와 RocksDB에서 Data-Collect
- Return results to Client

Iterate & Collect

```
/*
 * fpScanCommand
 * Scan data from the database(Redis & RocksDB)
 * --- Parameters ---
 *  arg1: Key(Table ID & PartitionInfo ID)
 *  arg2: Column IDs to find
 *
 * --- Usage Examples ---
 *  Parameters:
 *      key: "D:{3:1:2}"
 *          tableId: "3"
 *          partitionInfoId: "1:2"
 *      columnIds: ["2", "3", "4"]
 *  Command:
 *      redis-cli> FPSCAN D:{3:2:1} 2,3,4
 *  Results:
 *      redis-cli> "20180509"
 *      redis-cli> "Do young Kim"
 *      redis-cli> "Yonsei Univ"
 *      ...
 */
void fpScanCommand(client *c) {
    serverLog(LL_DEBUG, "FPSCAN COMMAND START");

    /*Creates scan parameters*/
    ScanParameter *scanParam = createScanParameter(c);

    /*Populates row group information to scan parameters*/
    int totalDataCount = populateScanParameter(c->db, scanParam);
    serverLog(LL_DEBUG, "total data count: %d", totalDataCount);

    /*Non-Vector response version*/
    void *replylen = addDeferredMultiBulkLength(c);
    size_t numreplies = scanDataFromADDB_non_vector(c, c->db, scanParam);
    freeScanParameter(scanParam);
    setDeferredMultiBulkLength(c, replylen, numreplies);
}
```

fpScan - createScanParameter()

- Parameter로부터 Scan에 필요한 Parameter들을 Parse
  - Key = “D:{tableID:partitionInfo}”
  - Columns = “1,3”
- Return → ScanParameter
  - int startRowGroupId
  - int totalRowGroupCount
  - NewDataKeyInfo *dataKeyInfo
  - RowGroupParameter *rowGroupParams (array)
  - ColumnParameter *columnParam

```
/*Scan Parameters*/
typedef struct _RowGroupParameter {
    robj *dictObj;      // RowGroup dict table object
    uint64_t isInRocksDb:1, rowCount:63;
} RowGroupParameter;

typedef struct _ColumnParameter {
    sds original;
    int columnCount;
    Vector columnIdList;        // int* vector
    Vector columnIdStrList;     // string vector
} ColumnParameter;

typedef struct _ScanParameter {
    int startRowGroupId;
    int totalRowGroupCount;
    NewDataKeyInfo *dataKeyInfo;
    RowGroupParameter *rowGroupParams;
    ColumnParameter *columnParam;
} ScanParameter;

/*Partition Filter Parameters*/
typedef union _ConditionValue {
    void *cond;
    long l;
    sds s;
} ConditionValue;
```

```
ScanParameter *createScanParameter(const client *c) {
    ScanParameter *param = (ScanParameter *) zmalloc(sizeof(ScanParameter));
    param->startRowGroupId = 0;
    param->dataKeyInfo = parsingDataKeyInfo((sds) c->argv[1]->ptr);
    param->totalRowGroupCount = getRowGroupInfoAndSetRowGroupInfo(
            c->db, param->dataKeyInfo);
    param->rowGroupParams = (RowGroupParameter *) zmalloc(
            sizeof(RowGroupParameter) * param->totalRowGroupCount);
    param->columnParam = parseColumnParameter((sds) c->argv[2]->ptr);
    return param;
}
```

fpScan - createScanParameter()

- Parameter로부터 Scan에 필요한 Parameter들을 Parse
  - Key = “D:{tableID:partitionInfo}”
  - Columns = “1,3”
- Return → ScanParameter
  - **int startRowGroupId**
    - 해당 Key의 RowGroup ID의 시작 지점
      - D:{tableID:partitionInfo}의 RowGroup 시작 지점
    - RowGroup은 1부터 시작임
    - 아직 RowGroup들을 가져오는 단계가 아니므로 0으로 초기화

```c
/*Scan Parameters*/
typedef struct _RowGroupParameter {
    robj *dictObj;      // RowGroup dict table object
    uint64_t isInRocksDb:1, rowCount:63;
} RowGroupParameter;

typedef struct _ColumnParameter {
    sds original;
    int columnCount;
    Vector columnIdList;        // int* vector
    Vector columnIdStrList;     // string vector
} ColumnParameter;

typedef struct _ScanParameter {
    int startRowGroupId;
    int totalRowGroupCount;
    NewDataKeyInfo *dataKeyInfo;
    RowGroupParameter *rowGroupParams;
    ColumnParameter *columnParam;
} ScanParameter;

/*Partition Filter Parameters*/
typedef union _ConditionValue {
    void *cond;
    long l;
    sds s;
} ConditionValue;
```

```c
ScanParameter *createScanParameter(const client *c) {
    ScanParameter *param = (ScanParameter *) zmalloc(sizeof(ScanParameter));
    param->startRowGroupId = 0;
    param->dataKeyInfo = parsingDataKeyInfo((sds) c->argv[1]->ptr);
    param->totalRowGroupCount = getRowGroupInfoAndSetRowGroupInfo(
            c->db, param->dataKeyInfo);
    param->rowGroupParams = (RowGroupParameter *) zmalloc(
            sizeof(RowGroupParameter) * param->totalRowGroupCount);
    param->columnParam = parseColumnParameter((sds) c->argv[2]->ptr);
    return param;
}
```

fpScan - createScanParameter()

- Parameter로부터 Scan에 필요한 Parameter들을 Parse
  - Key = “D:{tableID:partitionInfo}”
  - Columns = “1,3”
- Return → ScanParameter
  - **NewDataKeyInfo *dataKeyInfo**
    - Key에서 tableId, partitionInfo를 parse
    - tableId
    - partitionInfo
    - partitionCnt
    - rowGroupId = 0 (default)

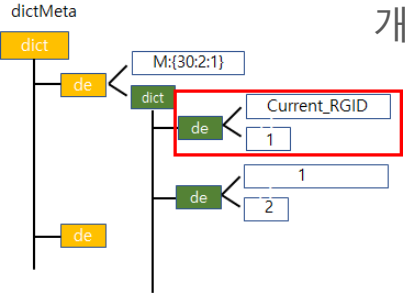
```
typedef struct NewDataKeyInfo {
  int tableId;
  int rowGroupId;
  int row_number;
  uint32_t isPartitionString:1, partitionCnt:31;
  Partition partitionInfo;
  uint32_t timeStamp;
} NewDataKeyInfo;
```

```
    int columnCount;
    Vector columnIdList;        // int* vector
    Vector columnIdStrList;     // string vector
} ColumnParameter;

typedef struct _ScanParameter {
    int startRowGroupId;
    int totalRowGroupCount;
    NewDataKeyInfo *dataKeyInfo;
    RowGroupParameter *rowGroupParams;
    ColumnParameter *columnParam;
} ScanParameter;

/*Partition Filter Parameters*/
typedef union _ConditionValue {
    void *cond;
    long l;
    sds s;
} ConditionValue;
```

```
ScanParameter *createScanParameter(const client *c) {
    ScanParameter *param = (ScanParameter *) zmalloc(sizeof(ScanParameter));
    param->startRowGroupId = 0;
    param->dataKeyInfo = parsingDataKeyInfo((sds) c->argv[1]->ptr);
    param->totalRowGroupCount = getRowGroupInfoAndSetRowGroupInfo(
            c->db, param->dataKeyInfo);
    param->rowGroupParams = (RowGroupParameter *) zmalloc(
            sizeof(RowGroupParameter) * param->totalRowGroupCount);
    param->columnParam = parseColumnParameter((sds) c->argv[2]->ptr);
    return param;
}
```

fpScan - createScanParameter()

- Parameter로부터 Scan에 필요한 Parameter들을 Parse
  - Key = “D:{tableID:partitionInfo}”
  - Columns = “1,3”
- Return → ScanParameter
  - **int totalRowGroupCount**
    - Metadict에서 dataKeyInfo에 해당하는 meta를 조회하여 dataKeyInfo에 해당하는 total row group 개수를 return
    - RowGroupID는 순차로 증가하므로, 현재 RowGroupID가 total row group 개수

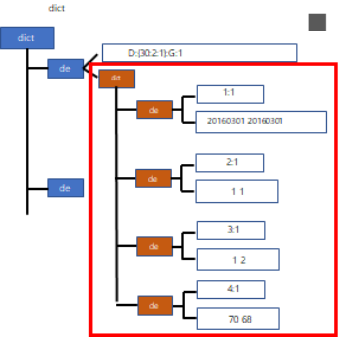



```
/*Scan Parameters*/
typedef struct _RowGroupParameter {
    robj *dictObj;      // RowGroup dict table object
    uint64_t isInRocksDb:1, rowCount:63;
} RowGroupParameter;

typedef struct _ColumnParameter {
    sds original;
    int columnCount;
    Vector columnIdList;        // int* vector
    Vector columnIdStrList;     // string vector
} ColumnParameter;

typedef struct _ScanParameter {
    int startRowGroupId;
    int totalRowGroupCount;
    NewDataKeyInfo *dataKeyInfo;
    RowGroupParameter *rowGroupParams;
    ColumnParameter *columnParam;
} ScanParameter;

/*Partition Filter Parameters*/
typedef union _ConditionValue {
    void *cond;
    long l;
    sds s;
} ConditionValue;
```

```
ScanParameter *createScanParameter(const client *c) {
    ScanParameter *param = (ScanParameter *) zmalloc(sizeof(ScanParameter));
    param->startRowGroupId = 0;
    param->dataKeyInfo = parsingDataKeyInfo((sds) c->argv[1]->ptr);
    param->totalRowGroupCount = getRowGroupInfoAndSetRowGroupInfo(
            c->db, param->dataKeyInfo);
    param->rowGroupParams = (RowGroupParameter *) zmalloc(
            sizeof(RowGroupParameter) * param->totalRowGroupCount);
    param->columnParam = parseColumnParameter((sds) c->argv[2]->ptr);
    return param;
}
```

fpScan - createScanParameter()

- Parameter로부터 Scan에 필요한 Parameter들을 Parse
  - Key = “D:{tableID:partitionInfo}”
  - Columns = “1,3”
- Return → ScanParameter
  - **RowGroupParameters *rowGroupParams**
    - Maindict에서 Key에 해당하는 RowGroup들의 hashdict를 저장함
    - RowGroup이 RocksDB에 있으면 **isInRocksDB**를 trigger
    - Parse 단계에서는 allocation만 진행하고 Populate 단계에서 Maindict로부터 가져옴




```
/*Scan Parameters*/
typedef struct _RowGroupParameter {
    robj *dictObj;      // RowGroup dict table object
    uint64_t isInRocksDb:1, rowCount:63;
} RowGroupParameter;

typedef struct _ColumnParameter {
    sds original;
    int columnCount;
    Vector columnIdList;        // int* vector
    Vector columnIdStrList;     // string vector
} ColumnParameter;

typedef struct _ScanParameter {
    int startRowGroupId;
    int totalRowGroupCount;
    NewDataKeyInfo *dataKeyInfo;
    RowGroupParameter *rowGroupParams;
    ColumnParameter *columnParam;
} ScanParameter;

/*Partition Filter Parameters*/
typedef union _ConditionValue {
    void *cond;
    long l;
    sds s;
} ConditionValue;
```

```
ScanParameter *createScanParameter(const client *c) {
    ScanParameter *param = (ScanParameter *) zmalloc(sizeof(ScanParameter));
    param->startRowGroupId = 0;
    param->dataKeyInfo = parsingDataKeyInfo((sds) c->argv[1]->ptr);
    param->totalRowGroupCount = getRowGroupInfoAndSetRowGroupInfo(
            c->db, param->dataKeyInfo);
    param->rowGroupParams = (RowGroupParameter *) zmalloc(
            sizeof(RowGroupParameter) * param->totalRowGroupCount);
    param->columnParam = parseColumnParameter((sds) c->argv[2]->ptr);
    return param;
}
```

fpScan - createScanParameter()

- Parameter로부터 Scan에 필요한 Paramete
  - Key = “D:{tableID:partitionInfo}”
  - Columns = “1,3”
- Return → ScanParameter
  - **ColumnParameter *columnParam**
    - Parameter의 ColumnIDs를 parse하여 저장
    - “1, 3”
      - columnParam->columnIdList = [1, 3] (int)
      - columnParam->columnIdStrList = [“1”, “3”] (sds)

```c
/*Scan Parameters*/
typedef struct _RowGroupParameter {
    robj *dictObj;      // RowGroup dict table object
    uint64_t isInRocksDb:1, rowCount:63;
} RowGroupParameter;
```

```c
/* ADDB Create Scan parameter*/
ColumnParameter *parseColumnParameter(const sds rawColumnIdsString) {
    ColumnParameter *param = (ColumnParameter *) zmalloc(
            sizeof(ColumnParameter));
    param->original = sdsnew(rawColumnIdsString);

    vectorTypeInit(&param->columnIdStrList, STL_TYPE_SDS);
    vectorTypeInit(&param->columnIdList, STL_TYPE_LONG);

    int tokenCounts;
    sds *tokens = sdssplit(param->original, RELMODEL_COLUMN_DELIMITER,
                           &tokenCounts);

    if (tokens == NULL || tokenCounts == 0) {
        serverLog(LL_WARNING, "Fatal: column parameter is broken: no data");
        serverPanic("column parameter is broken");
    }

    for (int i = 0; i < tokenCounts; ++i) {
        // TODO(totoro): Checks that column ID is valid.
        vectorAdd(&param->columnIdStrList, sdsdup(tokens[i]));
        vectorAdd(&param->columnIdList, (void *) (long) atoi(tokens[i]));
    }

    param->columnCount = vectorCount(&param->columnIdList);
    sdsfreesplitres(tokens, tokenCounts);
    return param;
}
```

```c
    param->rowGroupParams = (RowGroupParameter *) zmalloc(
            sizeof(RowGroupParameter) * param->totalRowGroupCount);
    param->columnParam = parseColumnParameter((sds) c->argv[2]->ptr);
    return param;
}
```

fpScan - createScanParameter()

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ScanParameter		
startRowGroupId		0
totalRowGroupCount		3
NewDataKeyInfo		
tableId		100
partitionInfo		2:1
rowGroupId		0
rowGroupParams		
0	dict	nullptr
	isInRocksDB	false
	rowCount	0
1	dict	nullptr
	isInRocksDB	false
	rowCount	0
2	dict	nullptr
	isInRocksDB	false
	rowCount	0
columnParam		
original		“1,3”
columnCount		2
columnIdList		[1, 3]
columnIdStrList		[“1”, “3”]

fpScan - Code Structure

fpScanCommand()

createScanParameter()

- Client의 Parameter들을 Parsing
- ScanParameter Object 생성

Parse

populateScanParameter()

- ScanParameter에 metadict data를 populate

Populate

scanDataFromADDB()

- ScanParameter로 Redis와 RocksDB에서 Data-Collect
- Return results to Client

Iterate & Collect

```
/*
 * fpScanCommand
 * Scan data from the database(Redis & RocksDB)
 * --- Parameters ---
 *  arg1: Key(Table ID & PartitionInfo ID)
 *  arg2: Column IDs to find
 *
 * --- Usage Examples ---
 *  Parameters:
 *      key: "D:{3:1:2}"
 *          tableId: "3"
 *          partitionInfoId: "1:2"
 *      columnIds: ["2", "3", "4"]
 *  Command:
 *      redis-cli> FPSCAN D:{3:2:1} 2,3,4
 *  Results:
 *      redis-cli> "20180509"
 *      redis-cli> "Do young Kim"
 *      redis-cli> "Yonsei Univ"
 *      ...
 */
void fpScanCommand(client *c) {
    serverLog(LL_DEBUG, "FPSCAN COMMAND START");

    /*Creates scan parameters*/
    ScanParameter *scanParam = createScanParameter(c);

    /*Populates row group information to scan parameters*/
    int totalDataCount = populateScanParameter(c->db, scanParam);
    serverLog(LL_DEBUG, "total data count: %d", totalDataCount);

    /*Non-Vector response version*/
    void *replylen = addDeferredMultiBulkLength(c);
    size_t numreplies = scanDataFromADDB_non_vector(c, c->db, scanParam);
    freeScanParameter(scanParam);
    setDeferredMultiBulkLength(c, replylen, numreplies);
}
```

fpScan - populateScanParameter()

- ScanParameter의 rowGroupParams를 populate 하는 함수
- RowGroupParams
  - RowGroup마다 저장되어있는 hashdict를 저장
  - RowGroup이 RocksDB로 Tierring된 경우
    - hashdict가 null
    - **isInRocksDB**를 Trigger
  - Return 값으로 Scan 해야할 총 Data Count를 return함.

```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
       ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - populateScanParameter()

- Key에 해당하는 모든 RowGroup들을 조회함

```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
       ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - populateScanParameter()

- Key에 해당하는 모든 RowGroup들을 조회함
  - rowGroupID는 1부터 시작함
  - Key를 parsing한 dataKeyInfo에 rowGroupID를 등록해 줌

```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
       ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - populateScanParameter()

- Key에 해당하는 모든 RowGroup들을 조회함
  - RowGroup이 채워진 dataKeyInfo를 string 형태의 dataKey로 변환함.
  - ex)

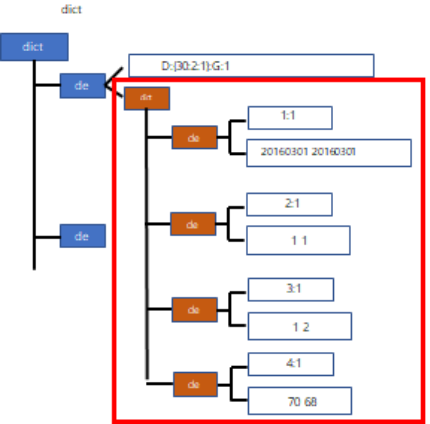
NewDataKeyInfo	
tableID	30
partitionInfo	1:2
RowGroupID	1

→ D:{30:1:2}:G:1

```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
        ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - populateScanParameter()

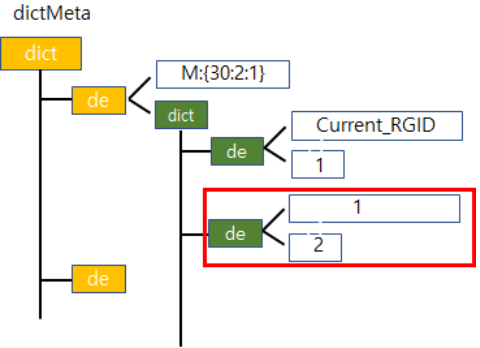
- Key에 해당하는 모든 RowGroup들을 조회함
  - 변환한 dataKey로 MainDict를 조회하여, dataKey에 해당하는 hashdict를 찾음
  - dataKey에 해당하는 hashdict가 MainDict에 존재하면 populate
  - 없으면 RocksDB에 Tierring 됐으므로, **isInRocksDB**를 Trigger함




```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
        ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - populateScanParameter()

- Key에 해당하는 모든 RowGroup들을 조회함
  - MetaDict에서 dataKey를 조회하여, RowGroup에 해당하는 rowCount를 가져옴
  - ScanParam->rowGroupParams에 rowCount를 업데이트
  - totalDataCount도 업데이트
  - 변환한 dataKey는 더이상 사용하지 않으므로 decrRefCount!




```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
       ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - populateScanParameter()

- ~~Key에 해당하는 모든 RowGroup들을 조회함~~
- 전체 row count를 return하며 종료됨

```
/*
 * populateScanParameter
 * - Description
 *      Populates row group data to scan parameter by looking up MetaDict
 * - Return
 *      Returns total data count scaned by scan parameter.
 */
int populateScanParameter(redisDb *db, ScanParameter *scanParam) {
    int totalDataCount = 0;

    for (size_t i = 0; i < scanParam->totalRowGroupCount; ++i) {
        int rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;
        robj *dataKey = generateDataKey(scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i] = createRowGroupParameter(db, dataKey);

        int rowCount = getRowNumberInfoAndSetRowNumberInfo(
                db, scanParam->dataKeyInfo);
        scanParam->rowGroupParams[i].rowCount = rowCount;
        totalDataCount += scanParam->columnParam->columnCount * rowCount;
        decrRefCount(dataKey);
    }
    return totalDataCount;
}

RowGroupParameter createRowGroupParameter(redisDb *db, robj *dataKey) {
    RowGroupParameter param;
    expireIfNeeded(db, dataKey);
    param.dictObj = lookupKey(db, dataKey, LOOKUP_NONE);

    if (
            param.dictObj == NULL ||
            param.dictObj->location == LOCATION_PERSISTED
       ) {
        param.isInRocksDb = true;
    } else {
        param.isInRocksDb = false;
    }

    return param;
}
```

fpScan - createScanParameter()

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ScanParameter		
startRowGroupId		0
totalRowGroupCount		3
NewDataKeyInfo		
tableId		100
partitionInfo		2:1
rowGroupId		0
rowGroupParams		
0	dict	nullptr
	isInRocksDB	true
	rowCount	8
1	dict	0xf93c7a13
	isInRocksDB	false
	rowCount	8
2	dict	0xfba1b98
	isInRocksDB	false
	rowCount	6
columnParam		
original		“1,3”
columnCount		2
columnIdList		[1, 3]
columnIdStrList		[“1”, “3”]

fpScan - Code Structure

fpScanCommand()

createScanParameter()

- Client의 Parameter들을 Parsing
- ScanParameter Object 생성

Parse

populateScanParameter()

- ScanParameter에 metadict data를 populate

Populate

scanDataFromADDB()

- ScanParameter로 Redis와 RocksDB에서 Data-Collect
- Return results to Client

Iterate & Collect

```
/*
 * fpScanCommand
 * Scan data from the database(Redis & RocksDB)
 * --- Parameters ---
 *  arg1: Key(Table ID & PartitionInfo ID)
 *  arg2: Column IDs to find
 *
 * --- Usage Examples ---
 *  Parameters:
 *      key: "D:{3:1:2}"
 *          tableId: "3"
 *          partitionInfoId: "1:2"
 *      columnIds: ["2", "3", "4"]
 *  Command:
 *      redis-cli> FPSCAN D:{3:2:1} 2,3,4
 *  Results:
 *      redis-cli> "20180509"
 *      redis-cli> "Do young Kim"
 *      redis-cli> "Yonsei Univ"
 *      ...
 */
void fpScanCommand(client *c) {
    serverLog(LL_DEBUG, "FPSCAN COMMAND START");

    /*Creates scan parameters*/
    ScanParameter *scanParam = createScanParameter(c);

    /*Populates row group information to scan parameters*/
    int totalDataCount = populateScanParameter(c->db, scanParam);
    serverLog(LL_DEBUG, "total data count: %d", totalDataCount);

    /*Non-Vector response version*/
    void *replylen = addDeferredMultiBulkLength(c);
    size_t numreplies = scanDataFromADDB_non_vector(c, c->db, scanParam);
    freeScanParameter(scanParam);
    setDeferredMultiBulkLength(c, replylen, numreplies);
}
```

Direct Reply to client

fpScan - scanDataFromADDB()

- Key “D:{tableID:partitionInfo}”에 해당하는 모든 Data들을 조회하고 Client로 출력
- D:{tableID:partitionInfo}의 모든 RowGroup을 순차적으로 조회함
- RowGroup이 저장된 위치에 따라 다른 함수가 호출됨
  - **RocksDB** (Tierred)
    - _cachedScanOnRocksDB
  - **Redis**
    - _cachedScan

```
size_t scanDataFromADDB_non_vector(client *c, redisDb *db,
                                   ScanParameter *scanParam) {
    size_t startRowGroupIdx = scanParam->startRowGroupId;
    ColumnParameter *columnParam = scanParam->columnParam;

    size_t numReplies = 0;
    for (size_t i = startRowGroupIdx; i < scanParam->totalRowGroupCount; ++i) {
        size_t rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;

        // Performs ColumnVector cached scan.
        if (scanParam->rowGroupParams[rowGroupId - 1].isInRocksDb) {
            numReplies += _cachedScanOnRocksDB_non_vector(
                c, db, rowGroupId, scanParam);
            continue;
        }

        numReplies += _cachedScan_non_vector(c, db, rowGroupId, scanParam);
    }
    return numReplies;
}
```

fpScan - scanDataFromADDB()

- D:{tableID:partitionInfo}의 모든 RowGroup들을 Iteration
  - dataKeyInfo에 현재 iterating중인 rowGroupID를 등록함

```
size_t scanDataFromADDB_non_vector(client *c, redisDb *db,
                                   ScanParameter *scanParam) {
    size_t startRowGroupIdx = scanParam->startRowGroupId;
    ColumnParameter *columnParam = scanParam->columnParam;

    size_t numReplies = 0;
    for (size_t i = startRowGroupIdx; i < scanParam->totalRowGroupCount; ++i) {
        size_t rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;

        // Performs ColumnVector cached scan.
        if (scanParam->rowGroupParams[rowGroupId - 1].isInRocksDb) {
            numReplies += _cachedScanOnRocksDB_non_vector(
                c, db, rowGroupId, scanParam);
            continue;
        }

        numReplies += _cachedScan_non_vector(c, db, rowGroupId, scanParam);
    }
    return numReplies;
}
```

fpScan - scanDataFromADDB()

- D:{tableID:partitionInfo}의 모든 RowGroup들을 Iteration
  - RowGroup이 **RocksDB**로 Tierring된 경우
  - Populate 단계에서 **isInRocksDB**를 Trigger했으므로, 감지 가능
  - _cachedScanOnRocksDB

rowGroupParams		
0	dict	nullptr
	isInRocksDB	**true**
	rowCount	8

```
size_t scanDataFromADDB_non_vector(client *c, redisDb *db,
                                   ScanParameter *scanParam) {
    size_t startRowGroupIdx = scanParam->startRowGroupId;
    ColumnParameter *columnParam = scanParam->columnParam;

    size_t numReplies = 0;
    for (size_t i = startRowGroupIdx; i < scanParam->totalRowGroupCount; ++i) {
        size_t rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;

        // Performs ColumnVector cached scan.
        if (scanParam->rowGroupParams[rowGroupId - 1].isInRocksDb) {
            numReplies += _cachedScanOnRocksDB_non_vector(
                c, db, rowGroupId, scanParam);
            continue;
        }

        numReplies += _cachedScan_non_vector(c, db, rowGroupId, scanParam);
    }
    return numReplies;
}
```

fpScan - scanDataFromADDB()

- D:{tableID:partitionInfo}의 모든 RowGroup들을 Iteration
  - RowGroup이 **Redis**에 남아 있는 경우
  - _cachedScan

rowGroupParams		
1	dict	0xf93c7a13
	isInRocksDB	**false**
	rowCount	8
2	dict	0xfba1b98
	isInRocksDB	**false**
	rowCount	6

```
size_t scanDataFromADDB_non_vector(client *c, redisDb *db,
                                   ScanParameter *scanParam) {
    size_t startRowGroupIdx = scanParam->startRowGroupId;
    ColumnParameter *columnParam = scanParam->columnParam;

    size_t numReplies = 0;
    for (size_t i = startRowGroupIdx; i < scanParam->totalRowGroupCount; ++i) {
        size_t rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;

        // Performs ColumnVector cached scan.
        if (scanParam->rowGroupParams[rowGroupId - 1].isInRocksDb) {
            numReplies += _cachedScanOnRocksDB_non_vector(
                c, db, rowGroupId, scanParam);
            continue;
        }

        numReplies += _cachedScan_non_vector(c, db, rowGroupId, scanParam);
    }
    return numReplies;
}
```

fpScan - _cachedScan()

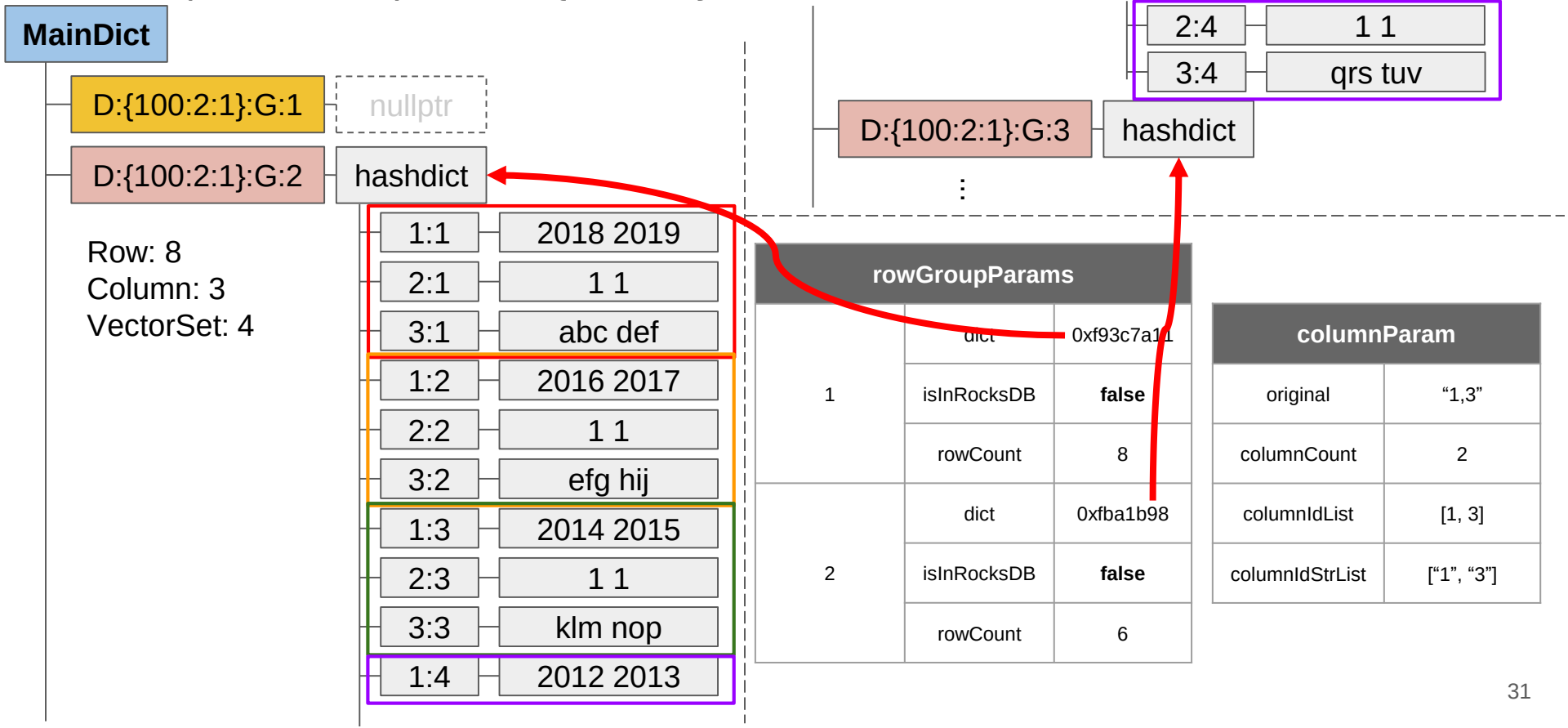
- Scan on RowGroup(**Redis**)
- RowGroup의 hashdict를 순회하며 데이터를 가져옴

```
size_t _cachedScan_non_vector(client *c, redisDb *db, size_t rowGroupId,
                              ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (robj **) zmalloc(
        sizeof(robj *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (int i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    serverLog(LL_VERBOSE, "     ");
    serverLog(LL_VERBOSE, "[SCAN] Scan On Redis");
    serverLog(
        LL_VERBOSE,
        "RowGroupId[%d], RowGroup->rowCount[%d]",
        rowGroupId, rowGroupParam->rowCount);

    size_t numReplies = 0;

    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);
    return numReplies;
}
```

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3



fpScan - _cachedScan()

- RowGroup의 Parameter
  - hashdict
  - ~~isInRocksDB~~
  - rowCount
- ColumnParameter
  - columnIds (int array)

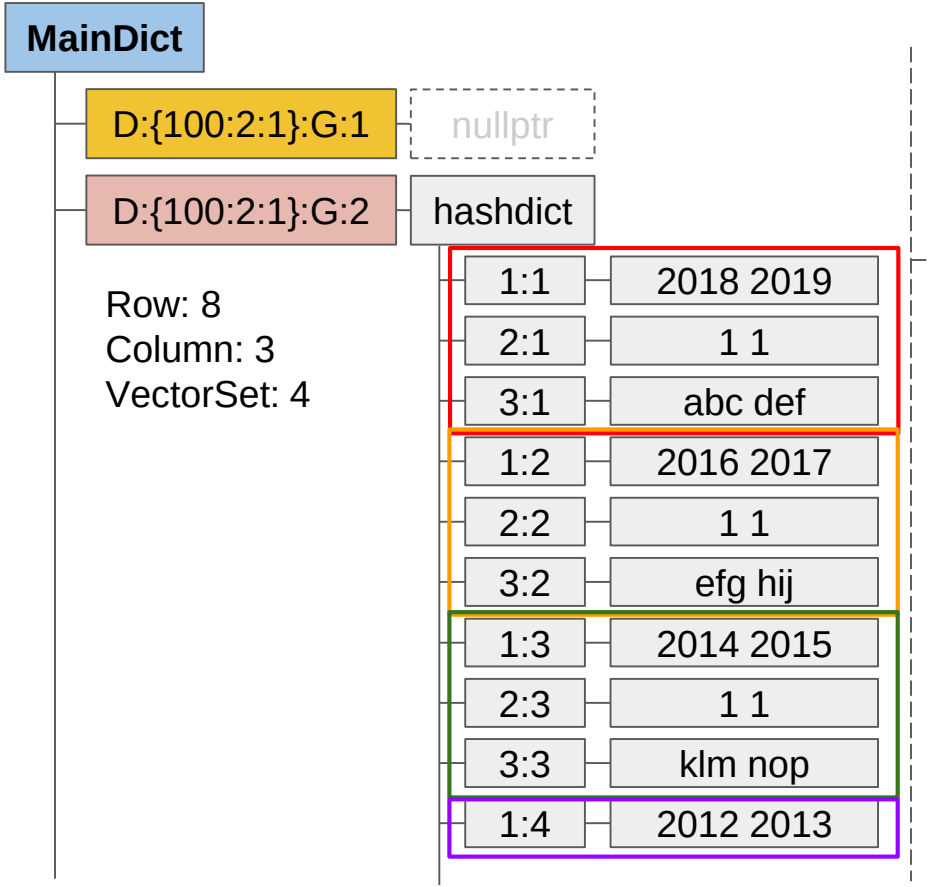
- column에 묶여있는 Vector들을 memory caching하여 접근속도를 높임
  - Column에 대하여 Memory Cache
  - cachedColumnVectorObjs
    - ColumnVector의 Objects (robj)
    - ex) “2018 2019”
  - cachedColumnVectorIds
    - ColumnVector의 IDs (int)

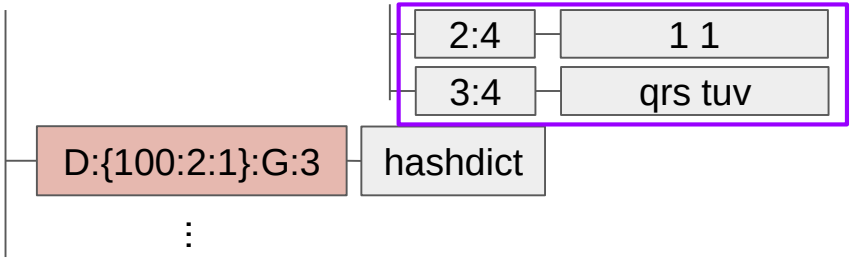
```
size_t _cachedScan_non_vector(client *c, redisDb *db, size_t rowGroupId,
                              ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (robj **) zmalloc(
        sizeof(robj *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (int i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    serverLog(LL_VERBOSE, "     ");
    serverLog(LL_VERBOSE, "[SCAN] Scan On Redis");
    serverLog(
        LL_VERBOSE,
        "RowGroupId[%d], RowGroup->rowCount[%d]",
        rowGroupId, rowGroupParam->rowCount);

    size_t numReplies = 0;

    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);
    return numReplies;
}
```

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





columnParam	
original	“1,3”
columnCount	2
columnIdList	[1, 3]
columnIdStrList	[“1”, “3”]

Memory Cache	
ColumnVectorObj	
0 → **1**	*nullptr*
1 → **3**	*nullptr*
ColumnVectorIds	
0	-1
1	-1

fpScan - _cachedScan()

- MemoryCache Build Up이 끝나면, 한개의 RowGroup를 순회

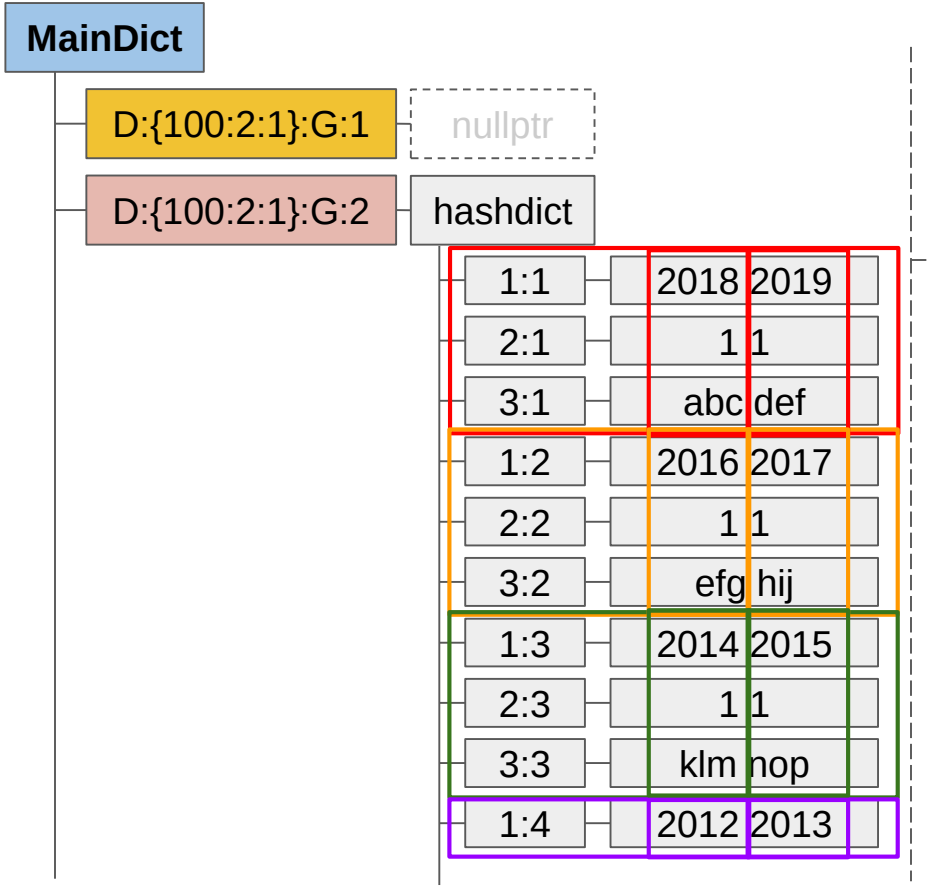
- for Rows
  for Columns
  순서로 순회

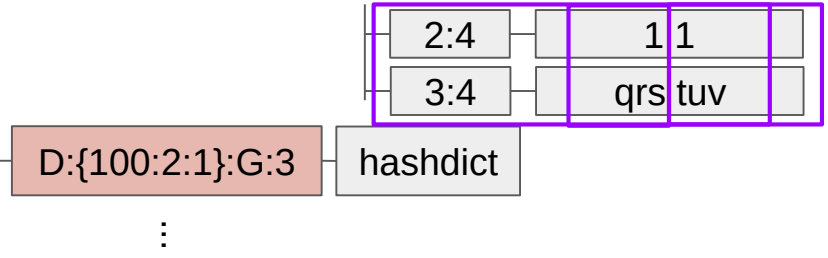
```
size_t _cachedScan_non_vector(client *c, redisDb *db, size_t rowGroupId,
                              ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (robj **) zmalloc(
        sizeof(robj *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (int i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    serverLog(LL_VERBOSE, "     ");
    serverLog(LL_VERBOSE, "[SCAN] Scan On Redis");
    serverLog(
        LL_VERBOSE,
        "RowGroupId[%d], RowGroup->rowCount[%d]",
        rowGroupId, rowGroupParam->rowCount);

    size_t numReplies = 0;

    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);
    return numReplies;
}
```

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → **1**	*nullptr*
1 → **3**	*nullptr*
ColumnVectorIds	
0	-1
1	-1

Rows
[

Cols
[2018, ~~1~~, abc],
[2019, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

fpScan - _cachedScan()

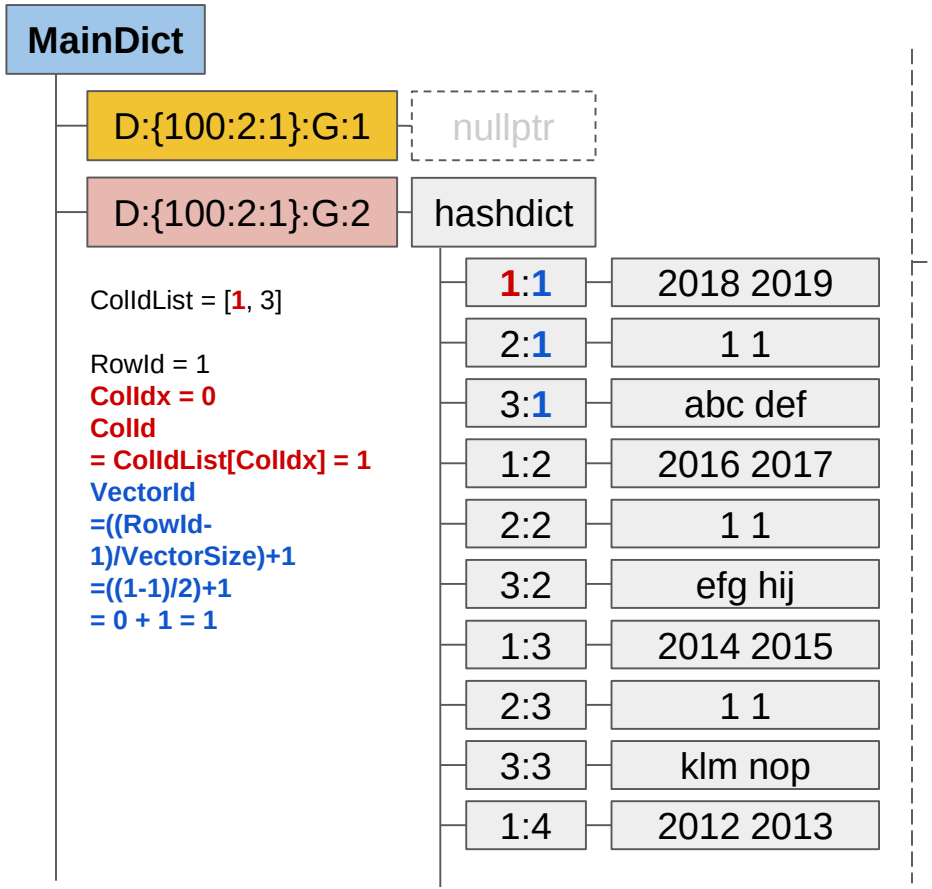
- columnParam의 IDList에서, columnID를 가져옴
  - ex) columnParam->columnIdList
    = [1, 3]
- rowID를 columnVectorID로 변환함
  - columnVectorID
    = ((rowID - 1) / kColumnVectorSize) + 1

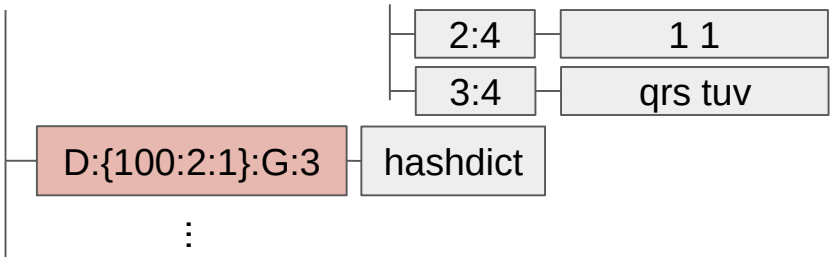
```
for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
    size_t rowId = j + 1;
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        size_t columnId = (long) vectorGet(&columnParam->columnIdList, k);
        int columnVectorId = getColumnVectorId(rowId);

        // Caching column vector.
        if (columnVectorId != cachedColumnVectorIds[k]) {
            // Redis Column Vector
            // Redis Key, which is DataField key (Row & Column pair)
            // Ex) "1:2"
            sds dataKey = getDataFieldSds(columnId, columnVectorId);
            robj *hashDictObj = rowGroupParam->dictObj;
            dict *hashDict = (dict *) hashDictObj->ptr;
            dictEntry *entry = dictFind(hashDict, dataKey);
            cachedColumnVectorIds[k] = columnVectorId;
            cachedColumnVectorObjs[k] = (robj *) dictGetVal(entry);
            sdsfree(dataKey);
        }

        Vector *columnVector = (Vector *) cachedColumnVectorObjs[k]->ptr;
        sds value = vectorGet(columnVector, getColumnVectorIndex(rowId));
        addReplyBulkSds(c, sdsdup(value));
        numReplies++;
    }
}
```

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → 1	*nullptr*
1 → 3	*nullptr*
ColumnVectorIds	
0	-1
1	-1

Rows
[

Cols
[**2018**, ~~1~~, abc],
[2019, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

fpScan - _cachedScan()

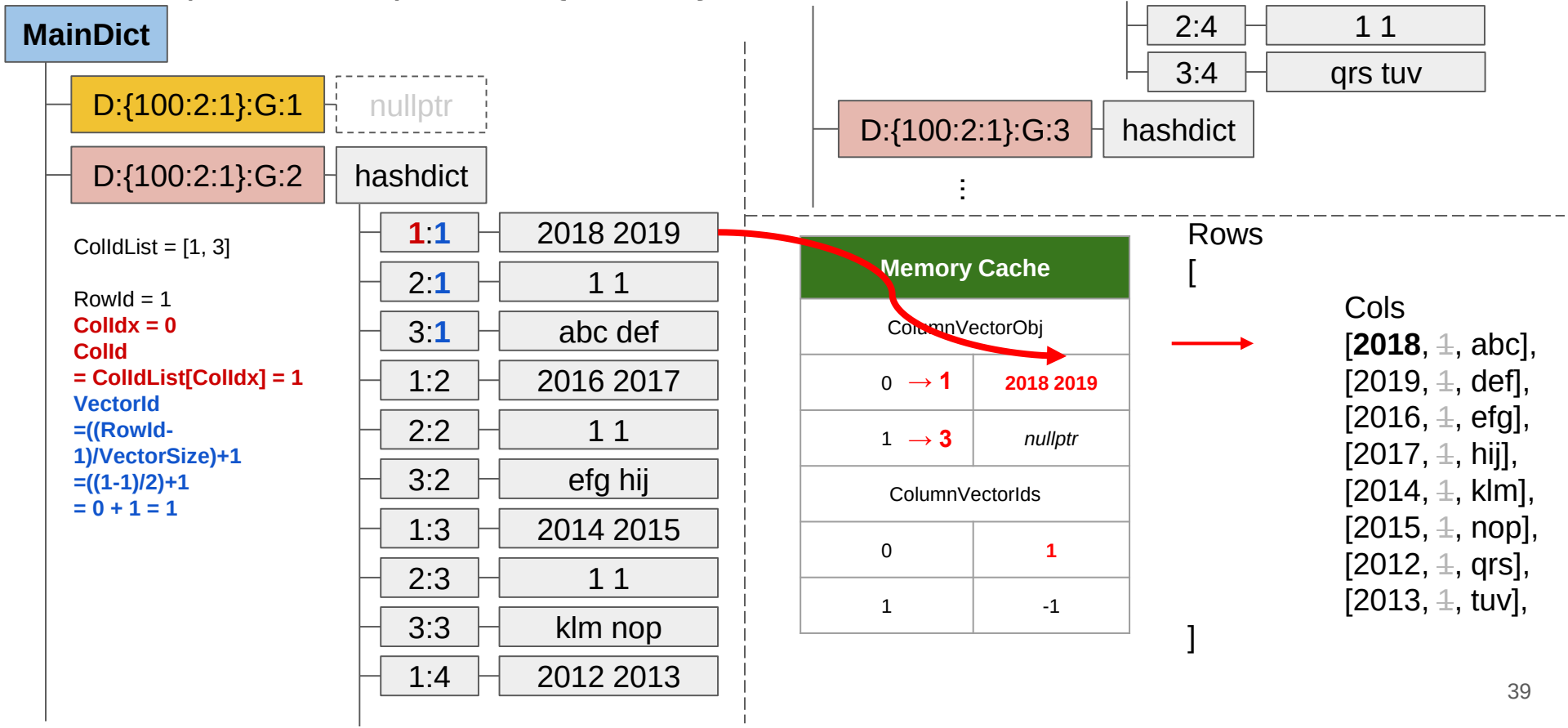
- 현재 columnVector가 Memory cache되어있지 않은 경우
  - columnID, columnVectorID로 DataField를 생성함
    - ex) columnID = 1, columnVectorID = 2 → DataField = “2:1”
  - RowGroup의 hashdict에서 DataField에 해당하는 ColumnVector를 가져옴
  - Memory Cache에 ColumnVector와 columnVectorID를 저장함

```
for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
    size_t rowId = j + 1;
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        size_t columnId = (long) vectorGet(&columnParam->columnIdList, k);
        int columnVectorId = getColumnVectorId(rowId);

        // Caching column vector.
        if (columnVectorId != cachedColumnVectorIds[k]) {
            // Redis Column Vector
            // Redis Key, which is DataField key (Row & Column pair)
            // Ex) "1:2"
            sds dataKey = getDataFieldSds(columnId, columnVectorId);
            robj *hashDictObj = rowGroupParam->dictObj;
            dict *hashDict = (dict *) hashDictObj->ptr;
            dictEntry *entry = dictFind(hashDict, dataKey);
            cachedColumnVectorIds[k] = columnVectorId;
            cachedColumnVectorObjs[k] = (robj *) dictGetVal(entry);
            sdsfree(dataKey);
        }

        Vector *columnVector = (Vector *) cachedColumnVectorObjs[k]->ptr;
        sds value = vectorGet(columnVector, getColumnVectorIndex(rowId));
        addReplyBulkSds(c, sdsdup(value));
        numReplies++;
    }
}
```

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3



fpScan - _cachedScan()

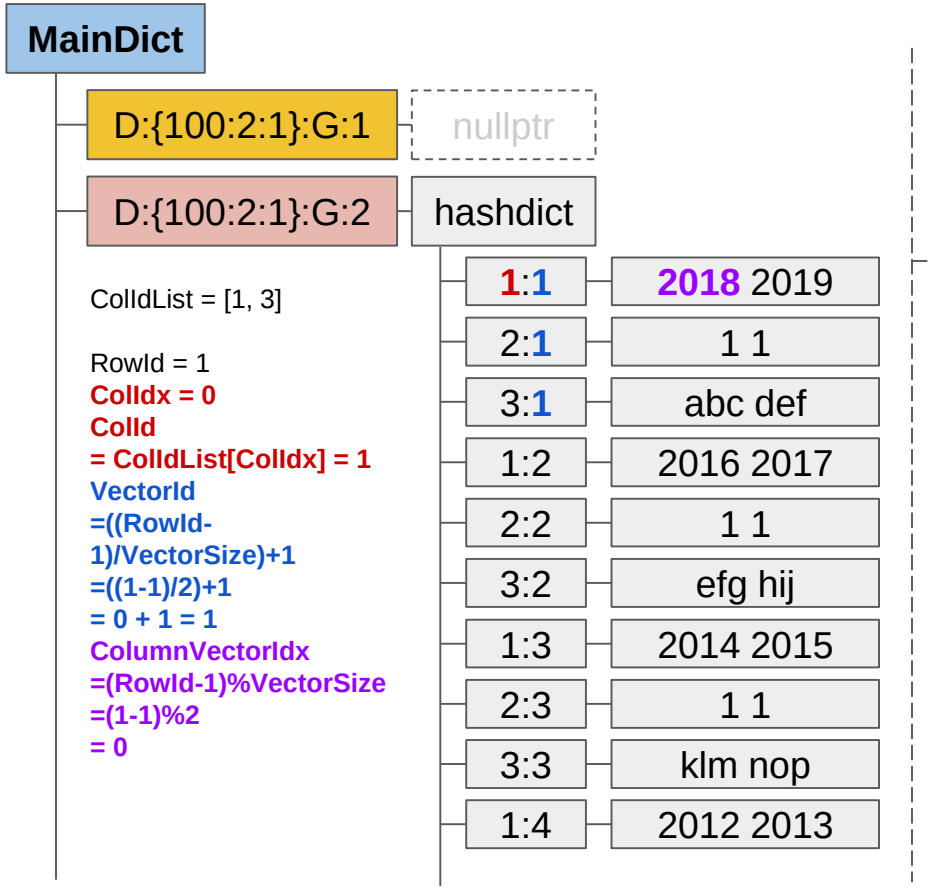
- 현재 columnVector가 Memory cache된 경우
  - Memory Cache에서 조회하는 column에 해당하는 ColumnVector를 가져옴
  - ColumnVector에서 rowID에 해당하는 데이터를 가져옴
    - ColumnVectorIndex
      = (rowID - 1) % kColumnVectorSize
  - Client로 Reply...

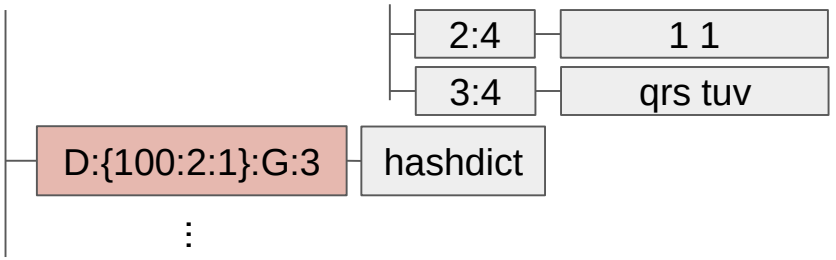
```
for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
    size_t rowId = j + 1;
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        size_t columnId = (long) vectorGet(&columnParam->columnIdList, k);
        int columnVectorId = getColumnVectorId(rowId);

        // Caching column vector.
        if (columnVectorId != cachedColumnVectorIds[k]) {
            // Redis Column Vector
            // Redis Key, which is DataField key (Row & Column pair)
            // Ex) "1:2"
            sds dataKey = getDataFieldSds(columnId, columnVectorId);
            robj *hashDictObj = rowGroupParam->dictObj;
            dict *hashDict = (dict *) hashDictObj->ptr;
            dictEntry *entry = dictFind(hashDict, dataKey);
            cachedColumnVectorIds[k] = columnVectorId;
            cachedColumnVectorObjs[k] = (robj *) dictGetVal(entry);
            sdsfree(dataKey);
        }

        Vector *columnVector = (Vector *) cachedColumnVectorObjs[k]->ptr;
        sds value = vectorGet(columnVector, getColumnVectorIndex(rowId));
        addReplyBulkSds(c, sdsdup(value));
        numReplies++;
    }
}
```

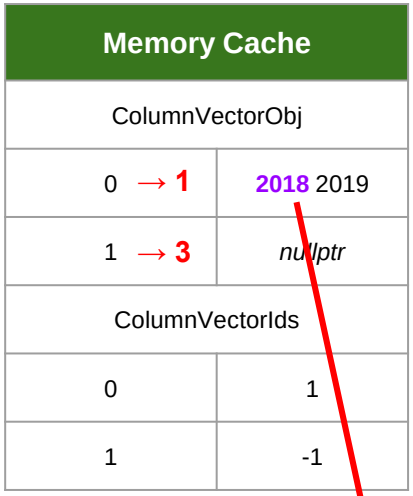
fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3







Rows
[

Cols
[**2018**, ~~1~~, abc],
[2019, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

Reply “2018” To Client

Next Col...

Rows
[

Cols
[~~2018~~, ~~1~~, **abc**],
[2019, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

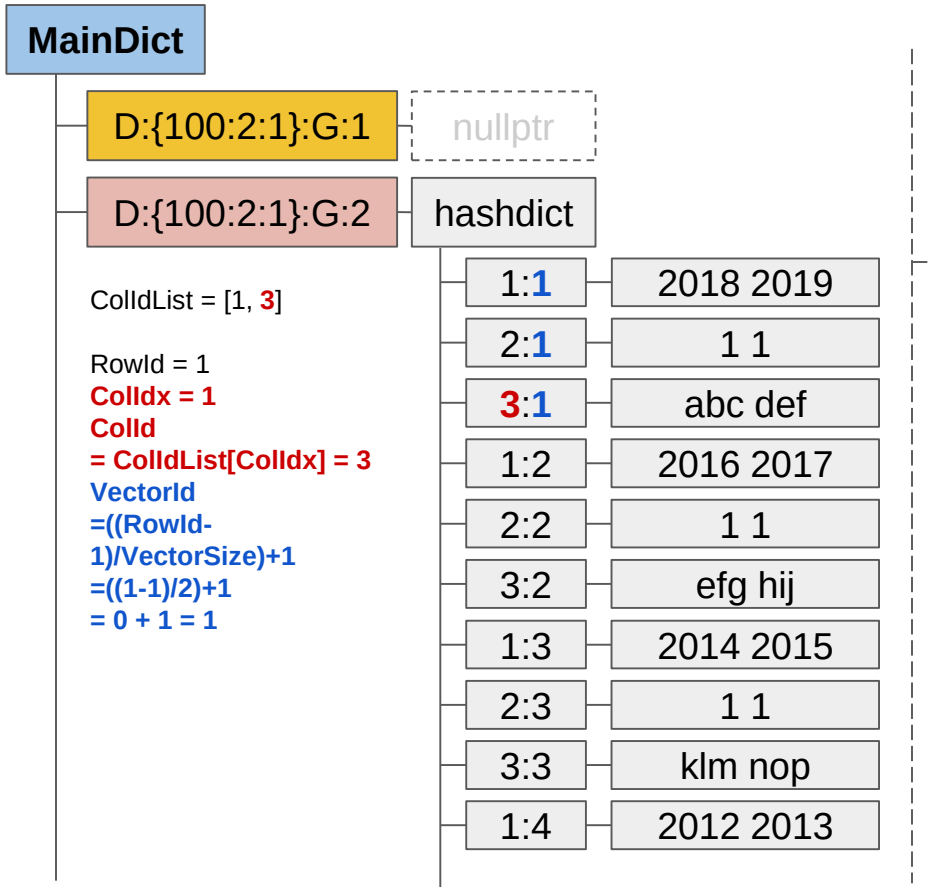
]

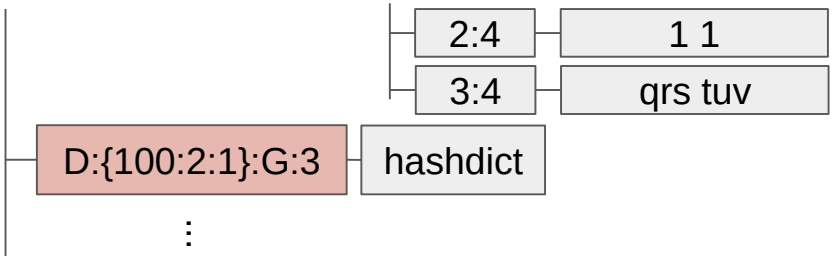
Memory Cache	
ColumnVectorObj	
0	2018 2019
1	*nullptr*
ColumnVectorIds	
0	1
1	-1

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → 1	2018 2019
1 → 3	*nullptr*
ColumnVectorIds	
0	1
1	-1

Rows
[

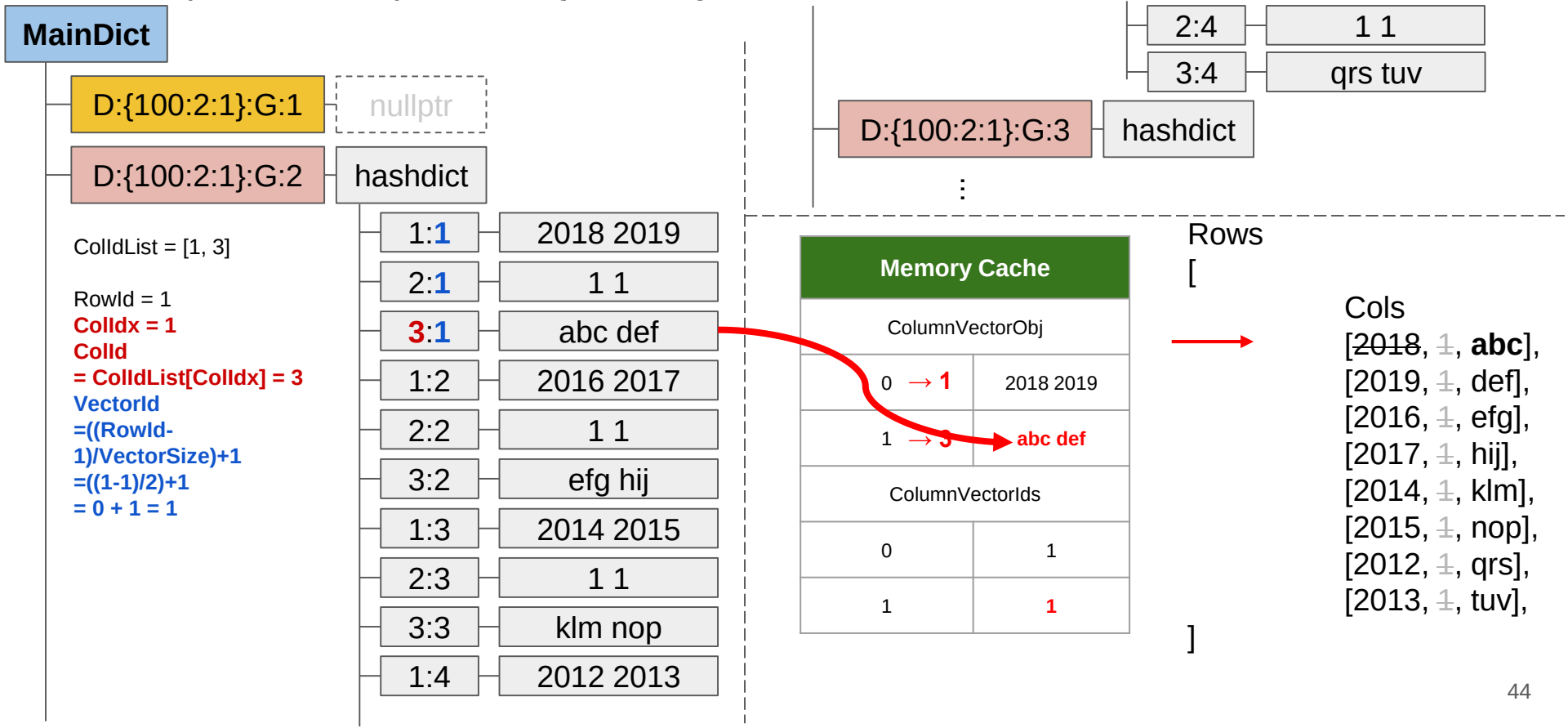
Cols
[~~2018~~, ~~1~~, **abc**],
[2019, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

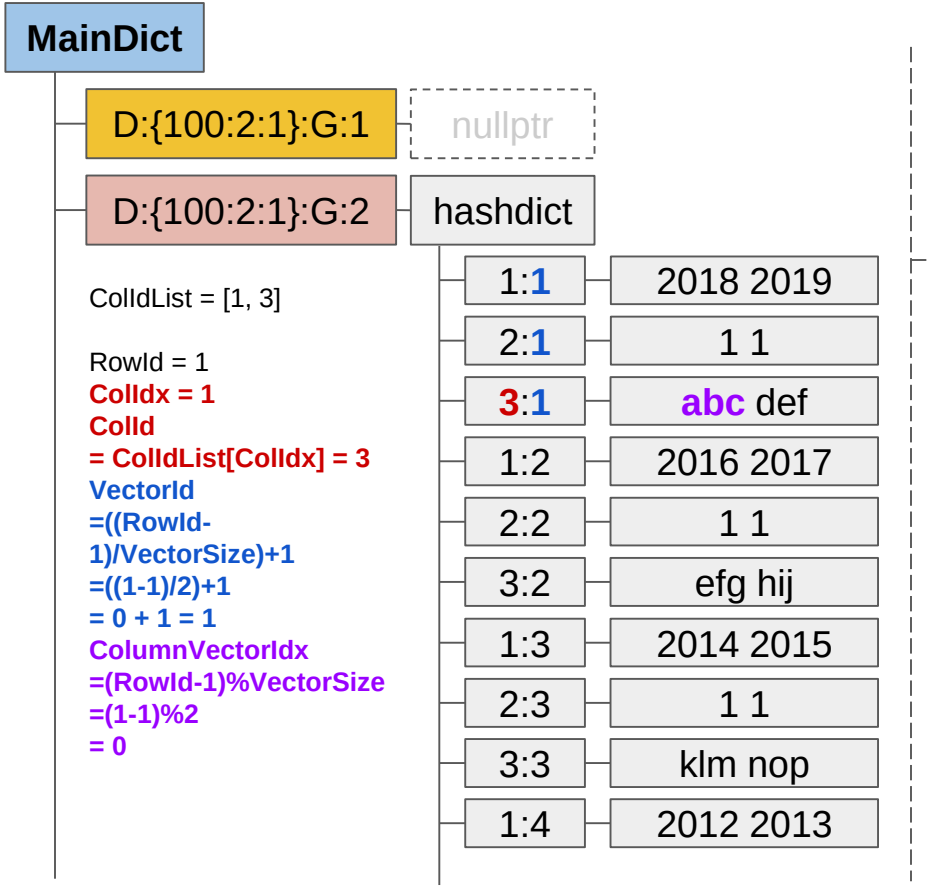
- ex) redis-cli> fpScan D:{100:2:1} 1,3

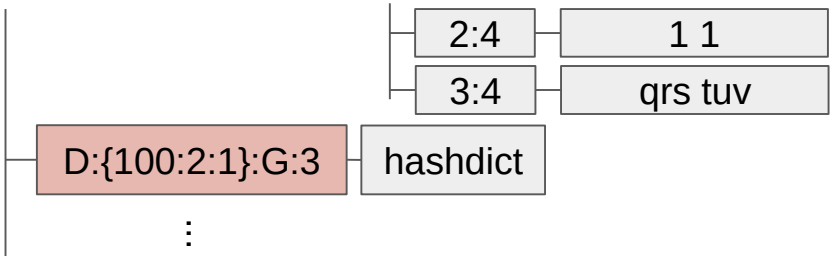


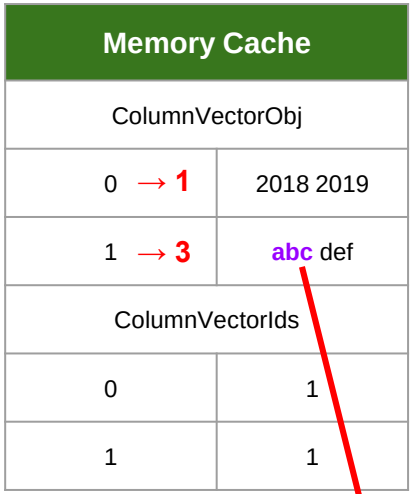
fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3







Rows
[

Cols
[~~2018~~, ~~1~~, **abc**],
[2019, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

Reply “abc” To Client

Next Row...

Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[**2019**, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

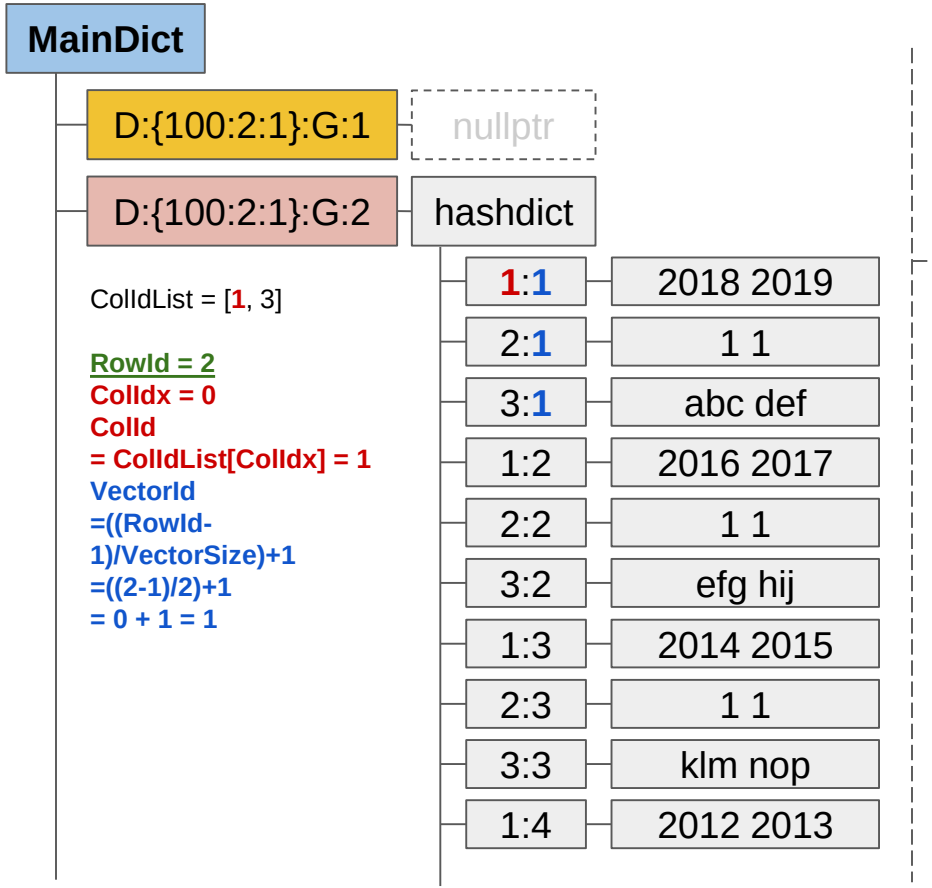
]

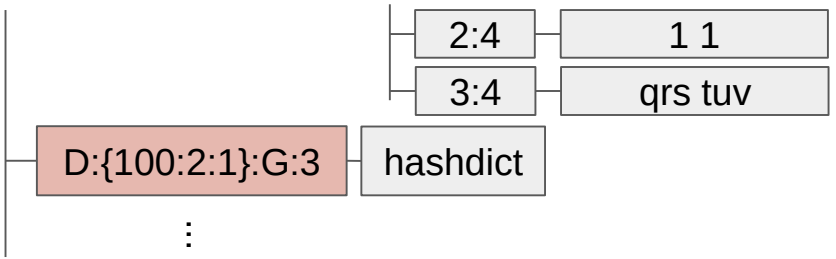
Memory Cache	
ColumnVectorObj	
0	2018 2019
1	abc def
ColumnVectorIds	
0	1
1	1

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → 1	2018 2019
1 → 3	abc def
ColumnVectorIds	
0	1
1	1

Rows
[

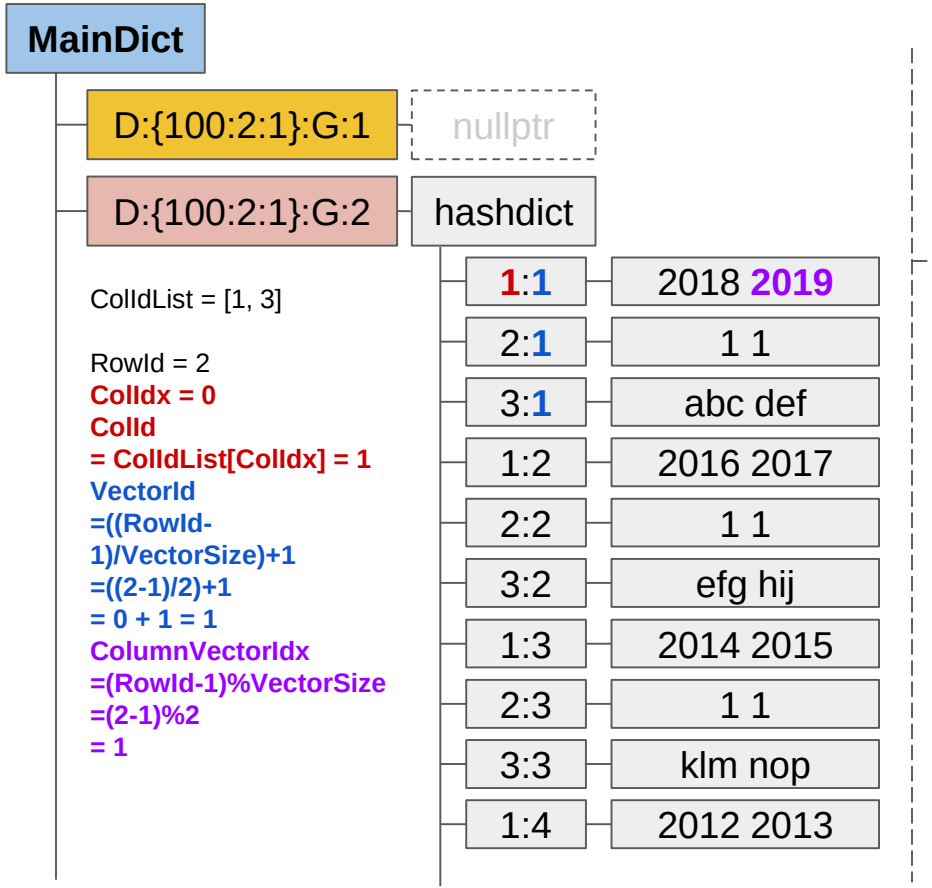
Cols
[~~2018~~, ~~1~~, ~~abc~~],
[**2019**, ~~1~~, def],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

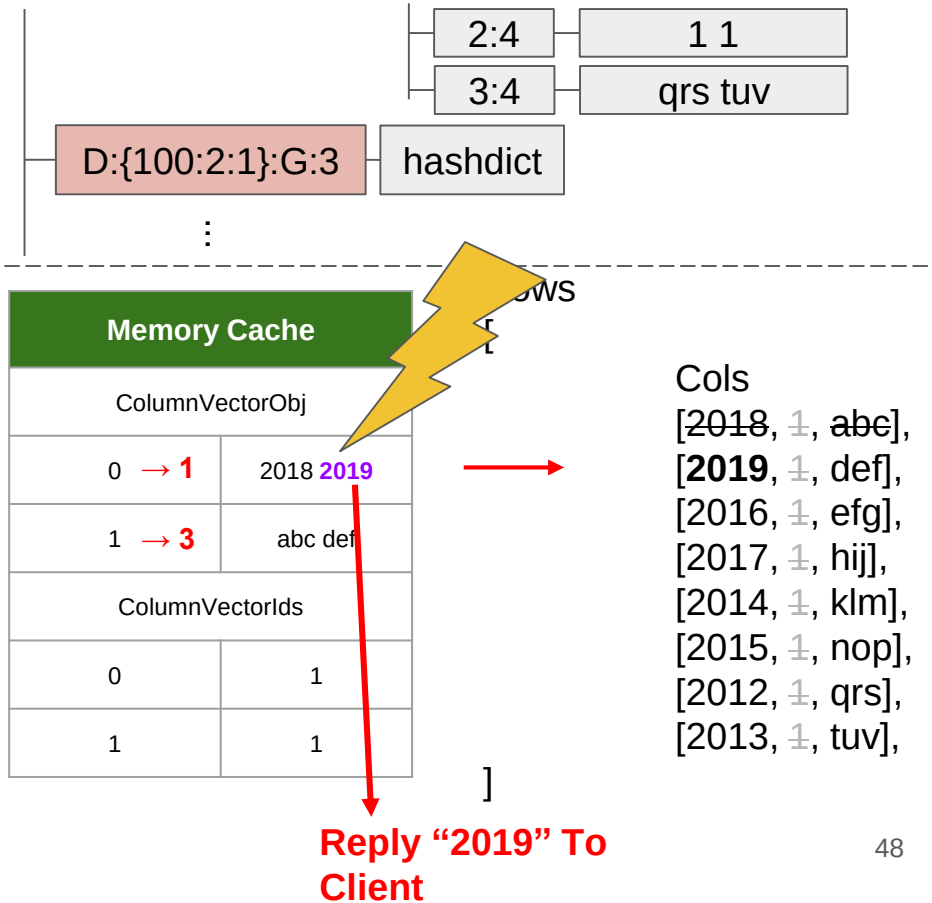
]

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Next Col...

Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, **def**],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

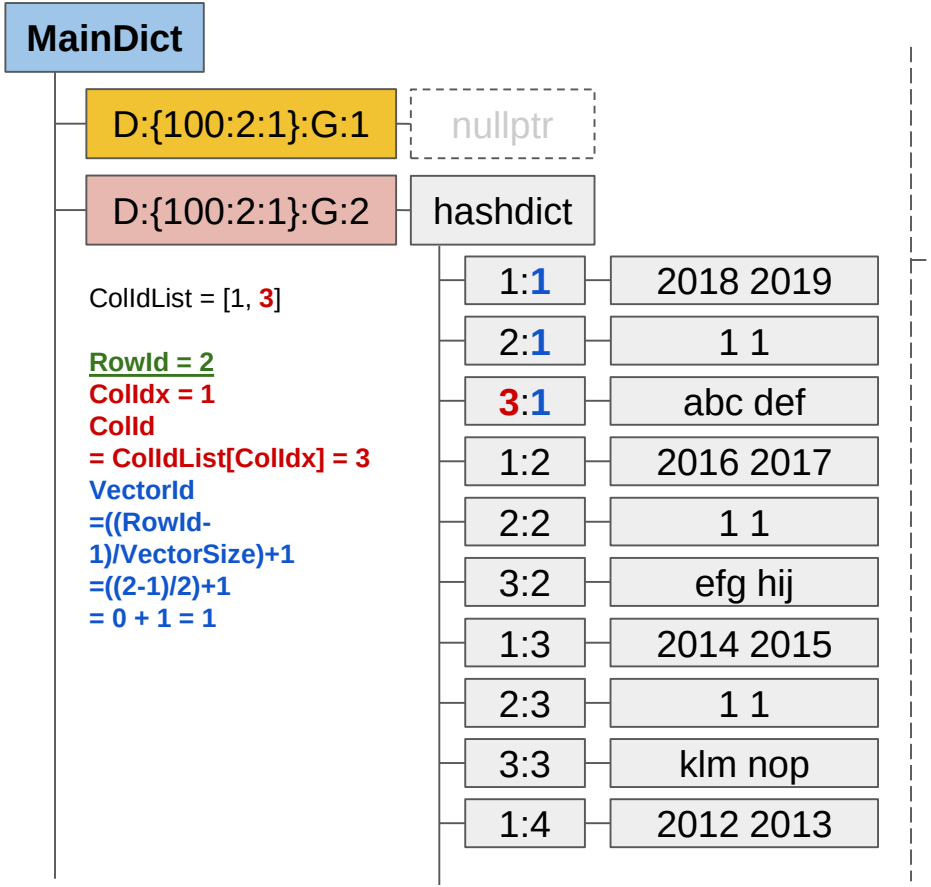
]

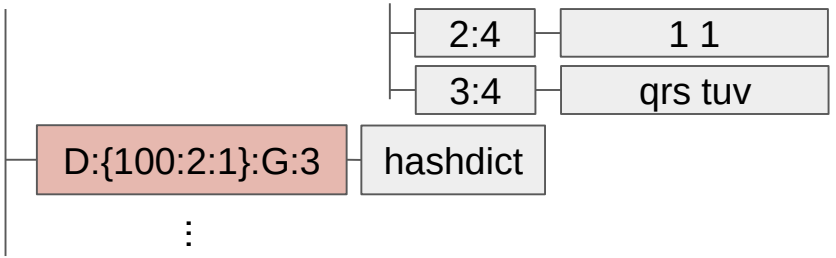
Memory Cache	
ColumnVectorObj	
0	2018 2019
1	abc def
ColumnVectorIds	
0	1
1	1

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → 1	2018 2019
1 → 3	abc def
ColumnVectorIds	
0	1
1	1

Rows
[

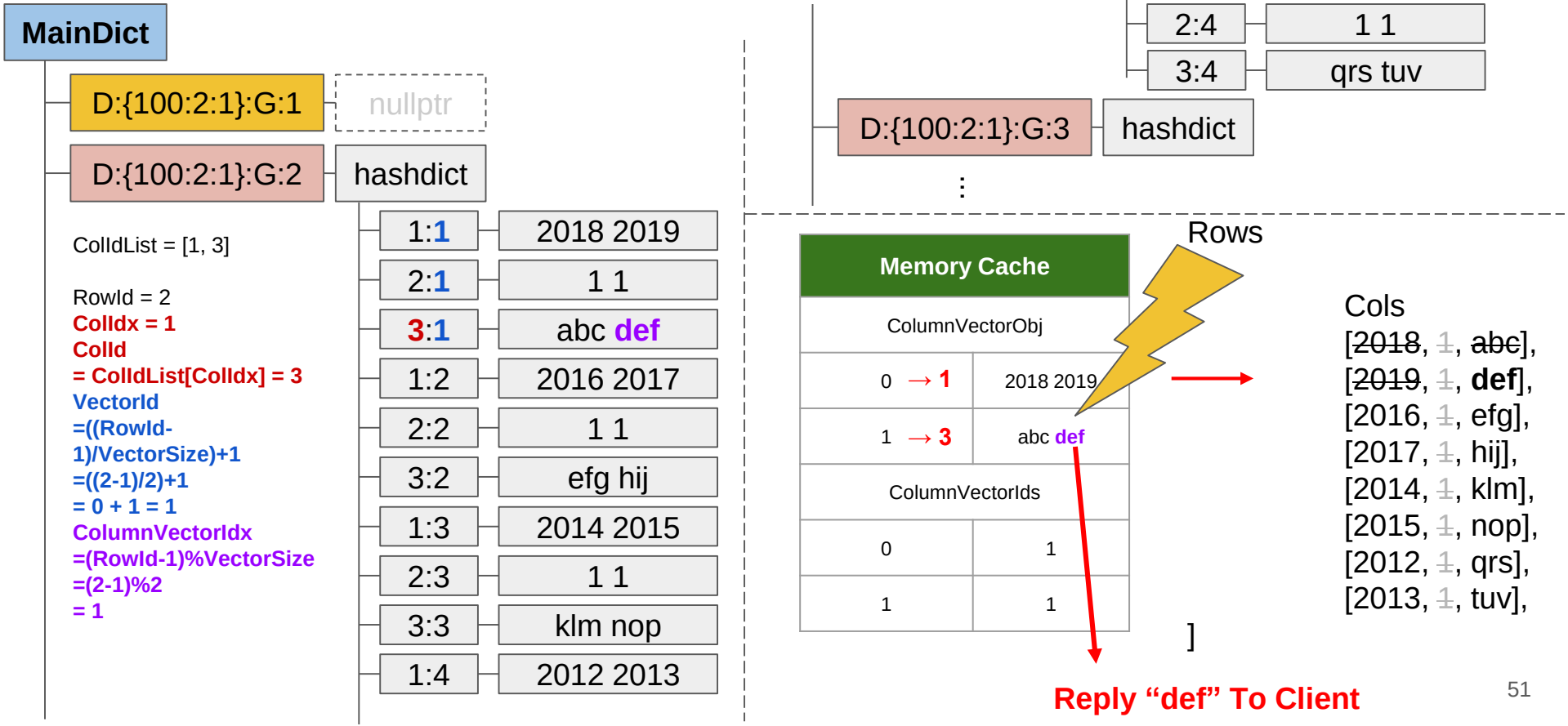
Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, **def**],
[2016, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3



Next Vector...

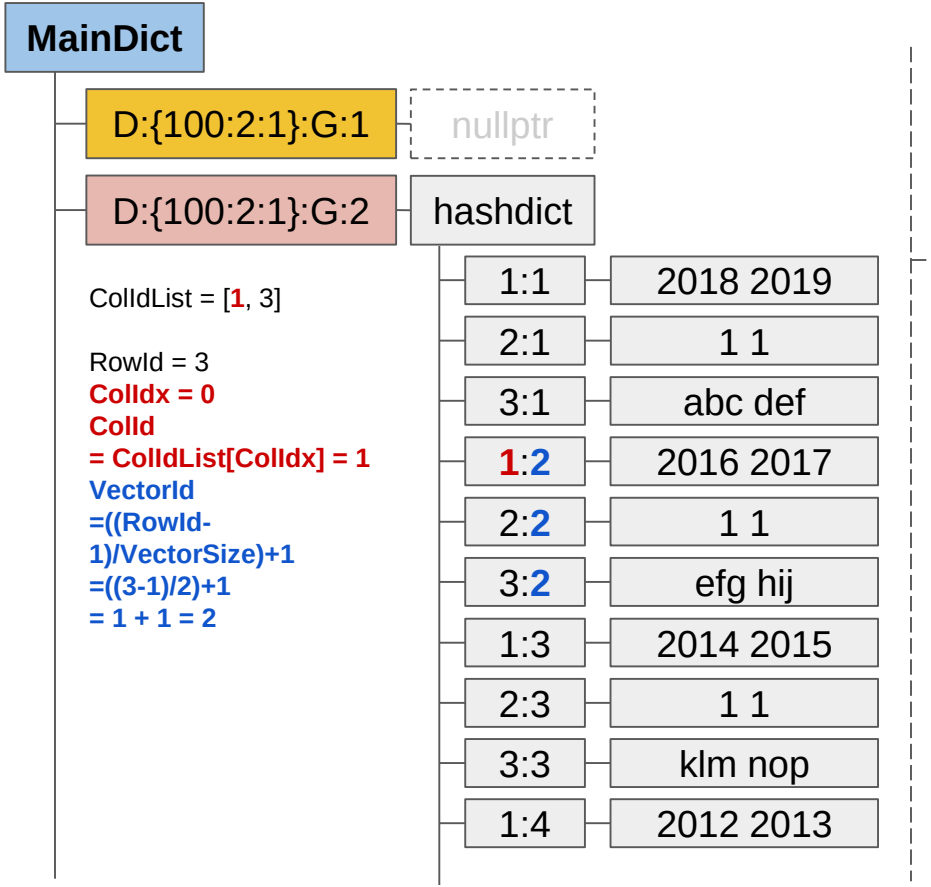
Rows
[
Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[**2016**, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],
]

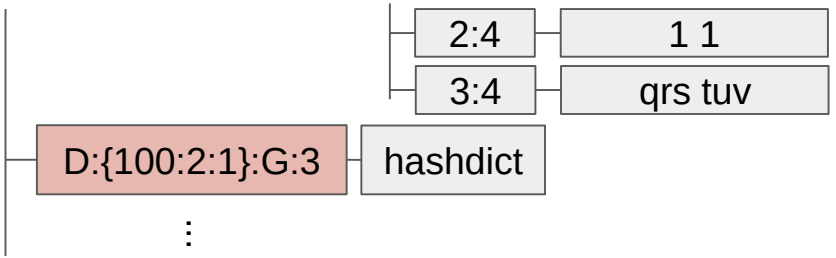
Memory Cache	
ColumnVectorObj	
0	2018 2019
1	abc def
ColumnVectorIds	
0	1
1	1

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → 1	2018 2019
1 → 3	abc def
ColumnVectorIds	
0	1
1	1

Rows
[

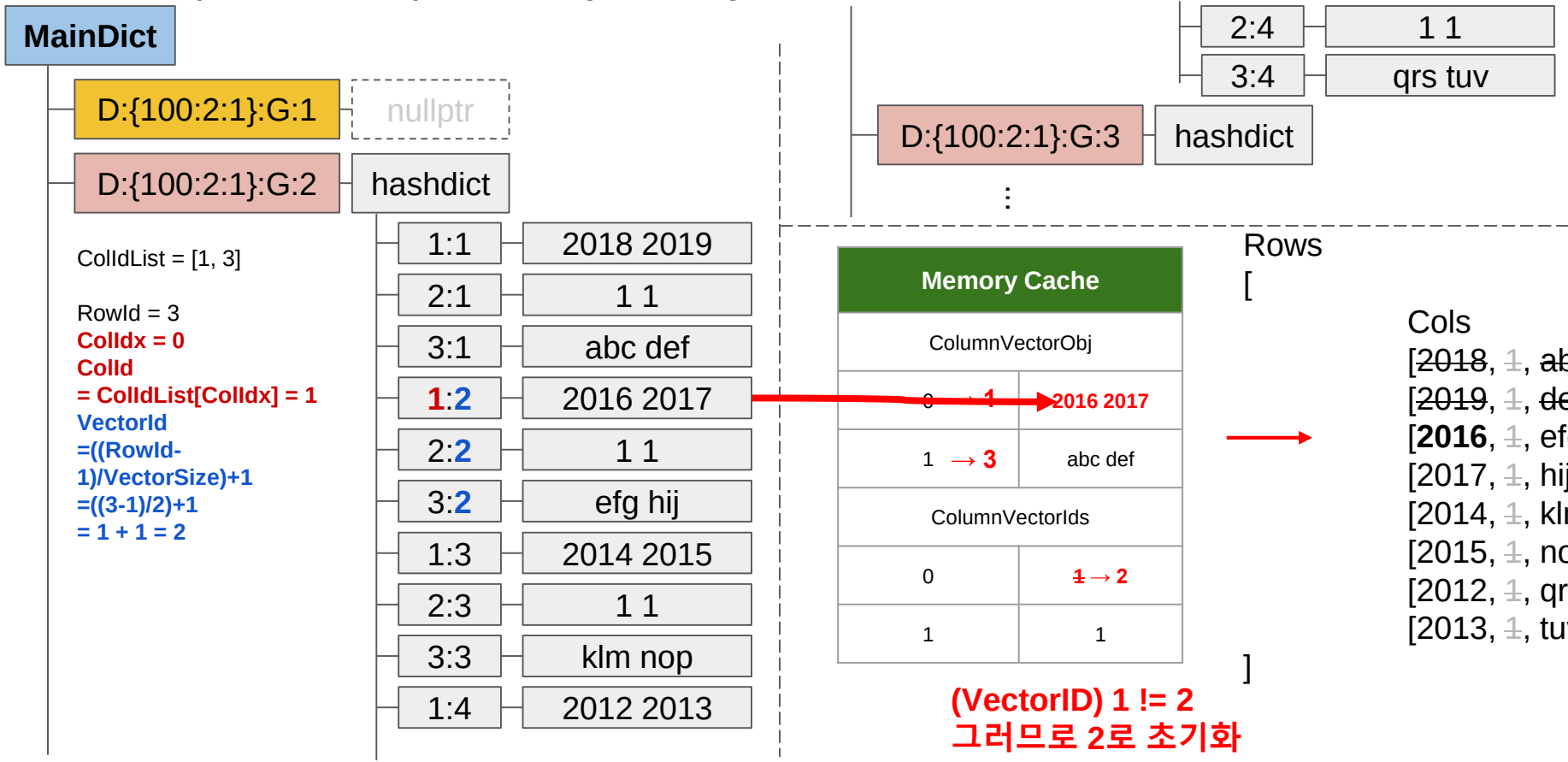
Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[**2016**, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

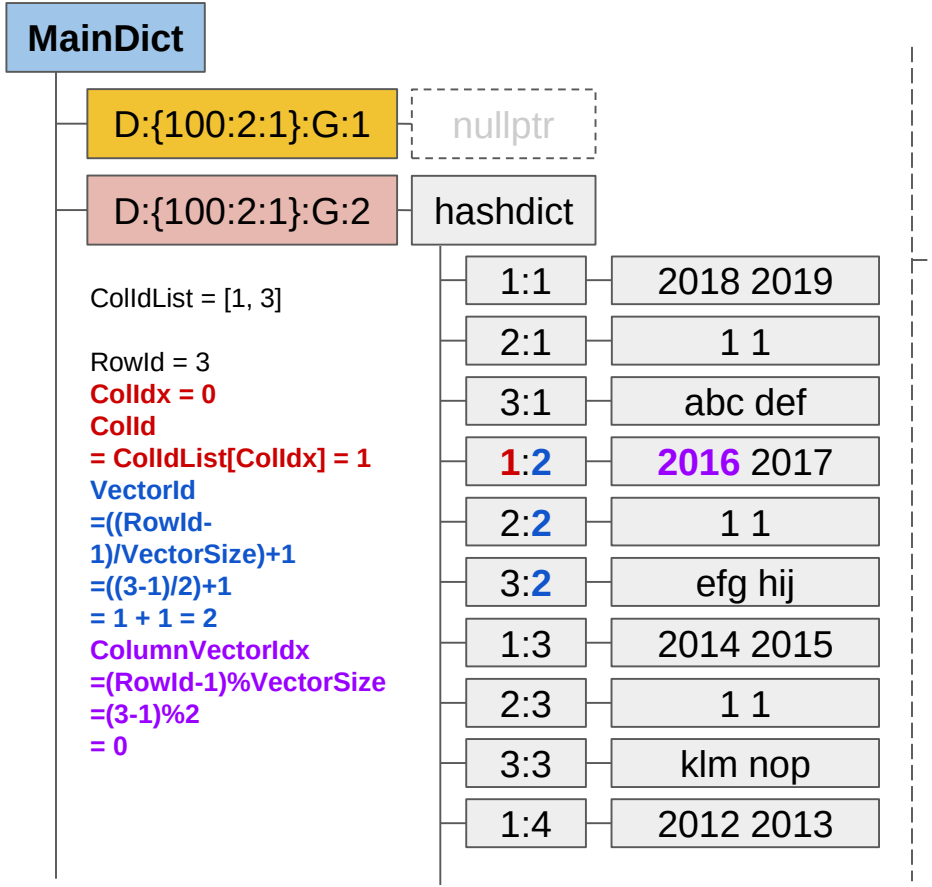
- ex) redis-cli> fpScan D:{100:2:1} 1,3

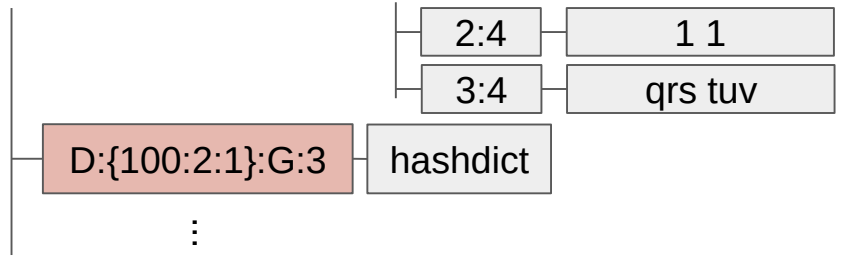


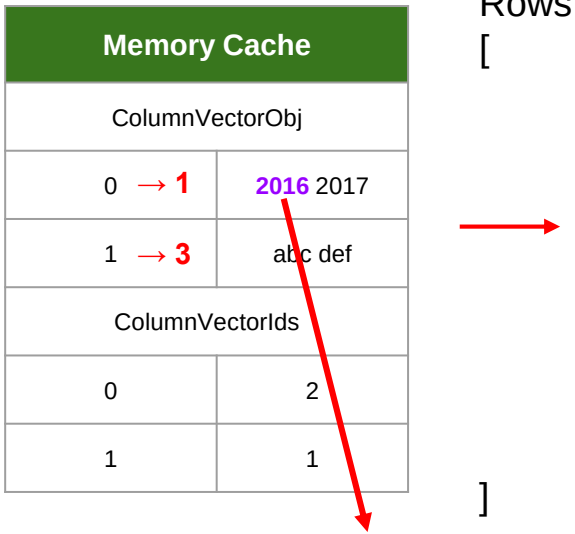
fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3







Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[**2016**, ~~1~~, efg],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

Reply “2016” To Client

Next Col...

Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[~~2016~~, ~~1~~, **efg**],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

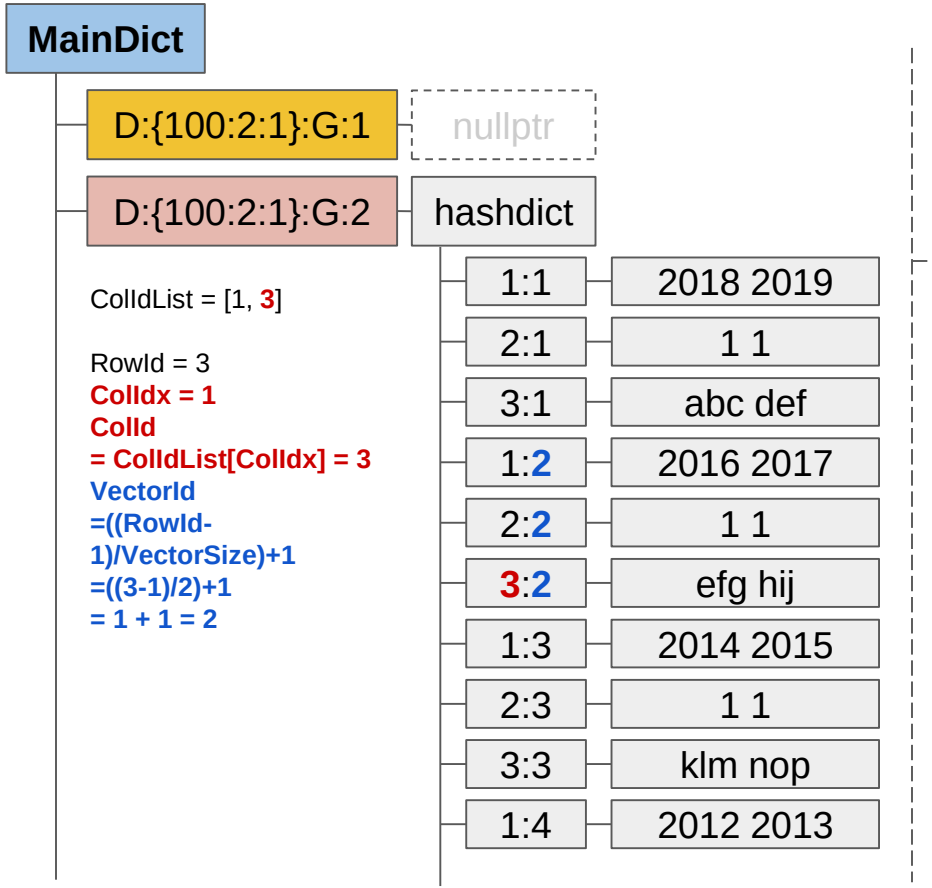
]

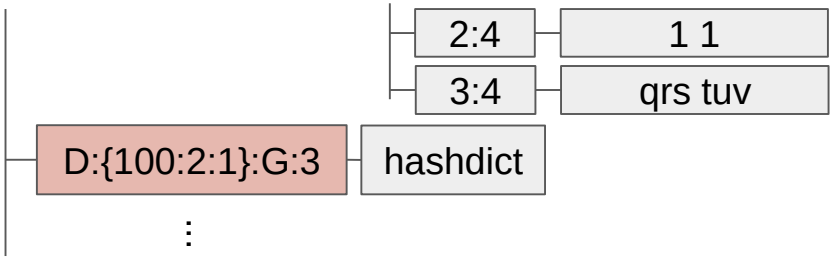
Memory Cache	
ColumnVectorObj	
0	2016 2017
1	abc def
ColumnVectorIds	
0	2
1	1

fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





Memory Cache	
ColumnVectorObj	
0 → 1	2016 2017
1 → 3	abc def
ColumnVectorIds	
0	2
1	1

Rows
[

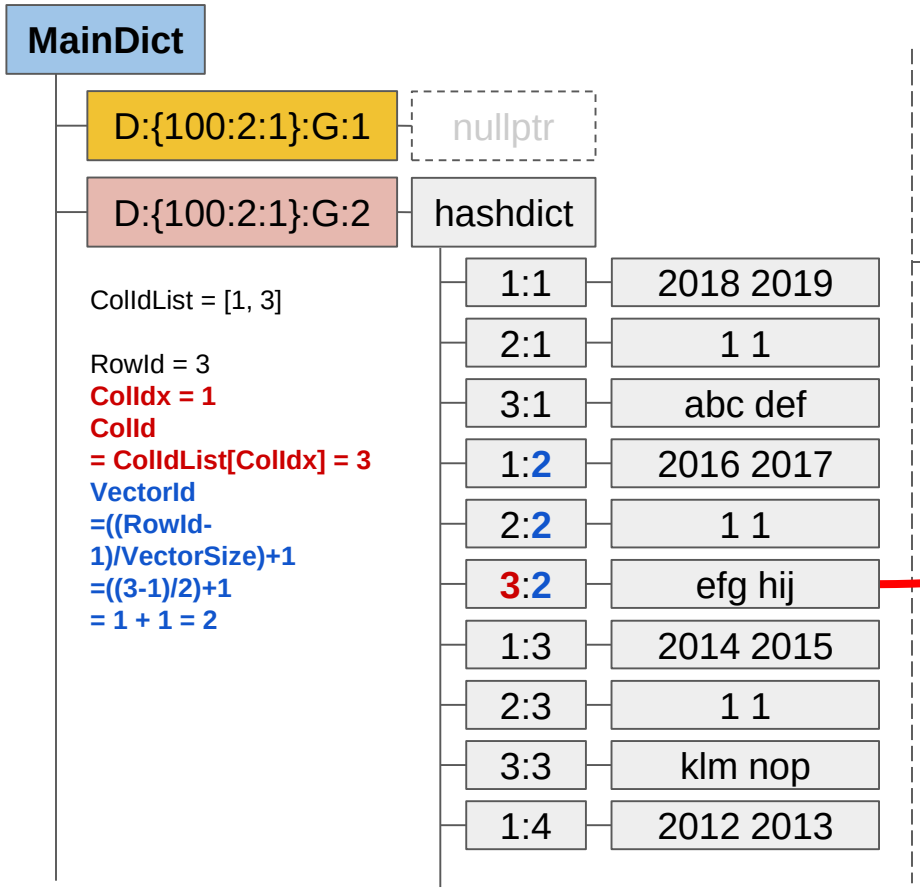
Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[~~2016~~, ~~1~~, **efg**],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

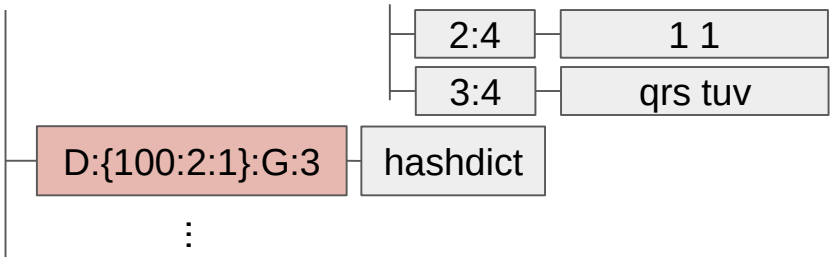
]

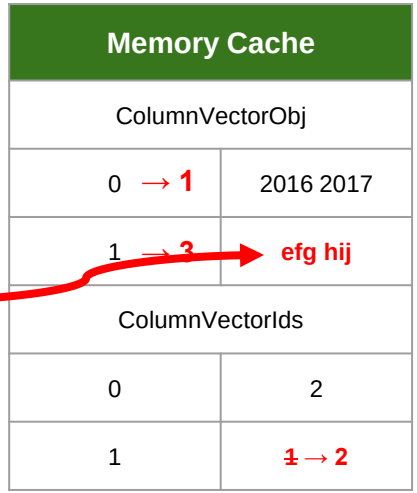
fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3







Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[~~2016~~, ~~1~~, **efg**],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

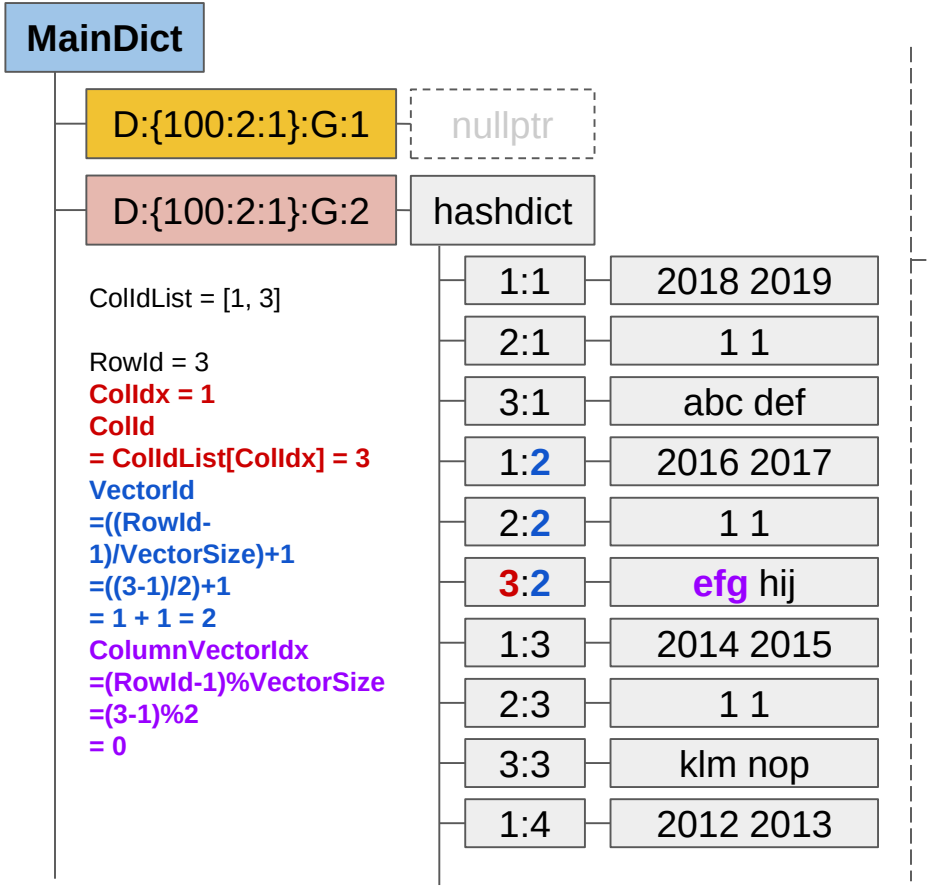
]

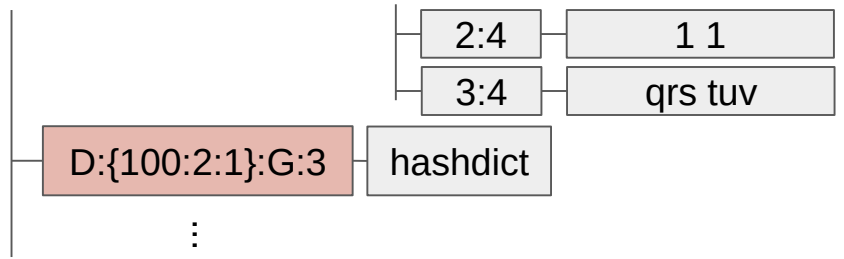
(VectorID) 1 != 2
그러므로 2로 초기화

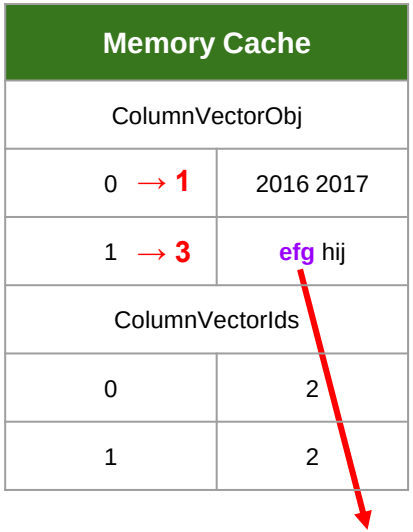
fpScan - _cachedScan()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3







Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[~~2016~~, ~~1~~, **efg**],
[2017, ~~1~~, hij],
[2014, ~~1~~, klm],
[2015, ~~1~~, nop],
[2012, ~~1~~, qrs],
[2013, ~~1~~, tuv],

]

Reply “efg” To Client

A few moments later…….

Finished reply final row of RowGroup 1… → *D:{100:2:1}:G:2*

Rows
[

Cols
[~~2018~~, ~~1~~, ~~abc~~],
[~~2019~~, ~~1~~, ~~def~~],
[~~2016~~, ~~1~~, ~~efg~~],
[~~2017~~, ~~1~~, ~~hij~~],
[~~2014~~, ~~1~~, ~~klm~~],
[~~2015~~, ~~1~~, ~~nop~~],
[~~2012~~, ~~1~~, ~~qrs~~],
[~~2013~~, ~~1~~, ~~tuv~~],

]

Memory Cache	
ColumnVectorObj	
0	2012 2013
1	qrs tuv
ColumnVectorIds	
0	4
1	4

fpScan - _cachedScan()

- Memory Cached ColumnVectors
  - 한개의 RowGroup에 대한 Iteration이 끝나면 Memory Cache를 삭제함

Memory Cache	
~~ColumnVectorObj~~	
~~0~~	~~2012 2013~~
~~1~~	~~qrs tuv~~
~~ColumnVectorIds~~	
~~0~~	~~4~~
~~1~~	~~4~~

```
size_t _cachedScan_non_vector(client *c, redisDb *db, size_t rowGroupId,
                              ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (robj **) zmalloc(
        sizeof(robj *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (int i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    serverLog(LL_VERBOSE, "     ");
    serverLog(LL_VERBOSE, "[SCAN] Scan On Redis");
    serverLog(
        LL_VERBOSE,
        "RowGroupId[%d], RowGroup->rowCount[%d]",
        rowGroupId, rowGroupParam->rowCount);

    size_t numReplies = 0;

    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);
    return numReplies;
}
```

Next RowGroup… → *D:{100:2:1}:G:3*

Rows
[
Cols
[2010, ~~1~~, wxy],
[2011, ~~1~~, zAB],
[2008, ~~1~~, CDE],
[2009, ~~1~~, FGH],
[2006, ~~1~~, IJK],
[2007, ~~1~~, LMO],
]

Memory Cache	
ColumnVectorObj	
0	*nullptr*
1	*nullptr*
ColumnVectorIds	
0	-1
1	-1

rowGroupParams		
~~1~~	~~dict~~	~~0xf93c7a11~~
	~~isInRocksDB~~	~~**false**~~
	~~rowCount~~	~~8~~
2	dict	0xfba1b98
	isInRocksDB	**false**
	rowCount	6

***Do It Yourself**
D:{100:2:1}:G:3에 대해서 cachedScan

fpScan - scanDataFromADDB()

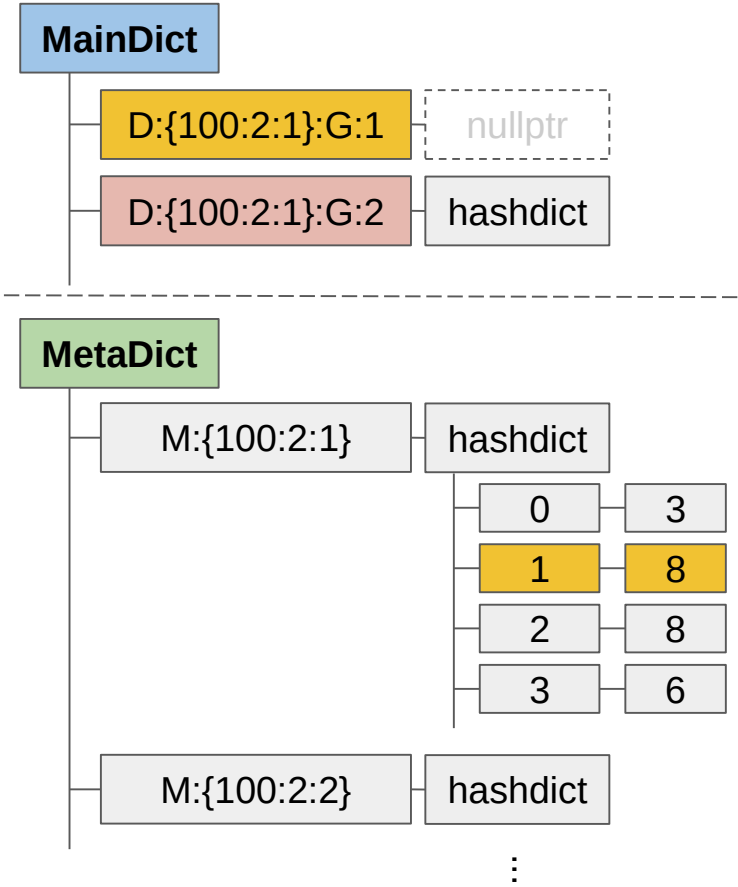
- D:{tableID:partitionInfo}의 모든 RowGroup들을 Iteration
  - RowGroup이 **RocksDB**로 Tierring된 경우
  - Populate 단계에서 **isInRocksDB**를 Trigger했으므로, 감지 가능
  - _cachedScanOnRocksDB

rowGroupParams		
0	dict	*nullptr*
	isInRocksDB	**true**
	rowCount	8

```
size_t scanDataFromADDB_non_vector(client *c, redisDb *db,
                                   ScanParameter *scanParam) {
    size_t startRowGroupIdx = scanParam->startRowGroupId;
    ColumnParameter *columnParam = scanParam->columnParam;

    size_t numReplies = 0;
    for (size_t i = startRowGroupIdx; i < scanParam->totalRowGroupCount; ++i) {
        size_t rowGroupId = i + 1;
        scanParam->dataKeyInfo->rowGroupId = rowGroupId;

        // Performs ColumnVector cached scan.
        if (scanParam->rowGroupParams[rowGroupId - 1].isInRocksDb) {
            numReplies += _cachedScanOnRocksDB_non_vector(
                c, db, rowGroupId, scanParam);
            continue;
        }

        numReplies += _cachedScan_non_vector(c, db, rowGroupId, scanParam);
    }
    return numReplies;
}
```

fpScan - _cachedScanOnRocksDB()

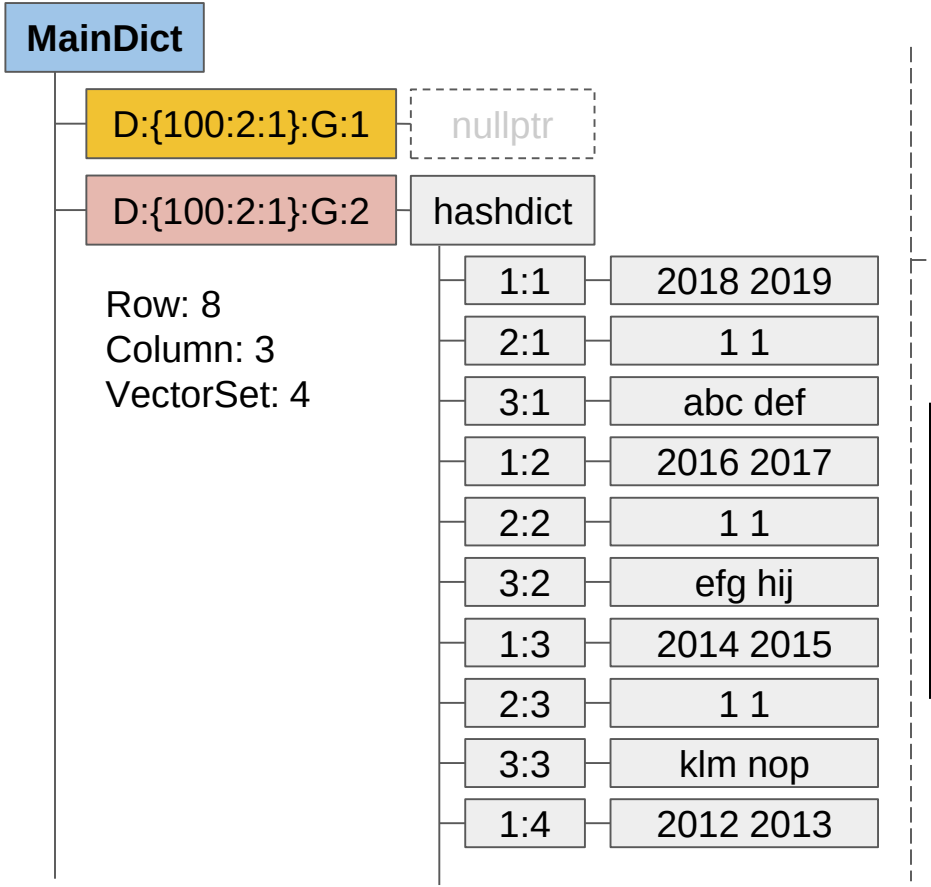
- Scan on RowGroup(**RocksDB**)
- RowGroup이 **RocksDB**로 Tierring되었으므로, hashdict가 존재하지 않음
- **RocksDB**에서 직접 RowGroup의 데이터들을 가져와야 함

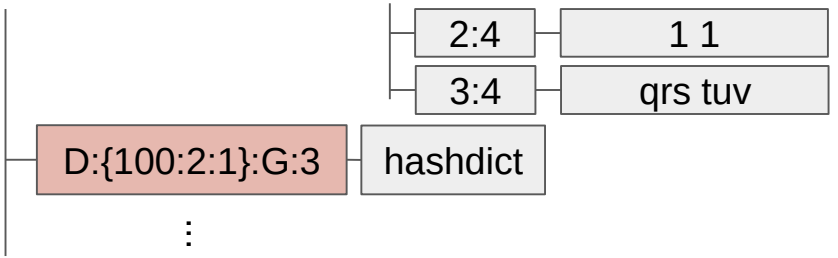


fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3





rowGroupParams		
0	dict	*nullptr*
	isInRocksDB	**true**
	rowCount	8

columnParam	
original	″1,3″
columnCount	2
columnIdList	[1, 3]
columnIdStrList	[″1″, ″3″]

fpScan - Example

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

MainDict

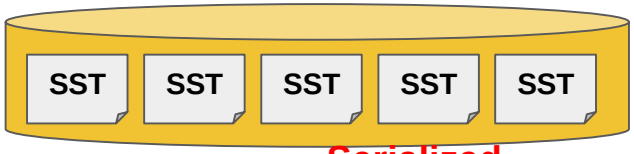
D:{100:2:1}:G:1	nullptr
D:{100:2:1}:G:2	hashdict

MetaDict

M:{100:2:1}	hashdict
M:{100:2:2}	hashdict

0	3
1	8
2	8
3	6

⋮



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

fpScan - _cachedScanOnRocksDB()

- RowGroup의 Parameter
  - ~~hashdict~~
  - isInRocksDB
  - rowCount
- ColumnParameter
  - columnIds (int array)

- column에 묶여있는 Vector들을 memory caching하여 접근속도를 높임
  - Column에 대하여 Memory Cache
  - cachedColumnVectorObjs
    - ColumnVector의 Objects (robj)
    - ex) “2018 2019”
  - cachedColumnVectorIds
    - ColumnVector의 IDs (int)

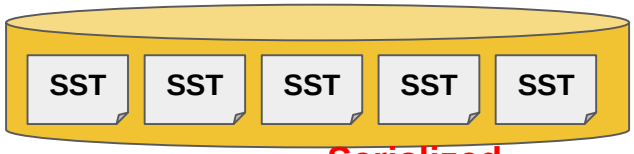
- Redis와 동일함

```
size_t _cachedScanOnRocksDB_non_vector(client *c, redisDb *db,
                                       size_t rowGroupId,
                                       ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (Vector **) zmalloc(
        sizeof(Vector *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (size_t i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    size_t numReplies = 0;
    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    // Removes created ColumnVector robjs.
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        vectorFreeDeep(cachedColumnVectorObjs[k]);
        zfree(cachedColumnVectorObjs[k]);
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);

    return numReplies;
}
```

fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

columnParam	
original	″1,3″
columnCount	2
columnIdList	[1, 3]
columnIdStrList	[″1″, ″3″]

Memory Cache	
ColumnVectorObj	
0 → 1	*nullptr*
1 → 3	*nullptr*
ColumnVectorIds	
0	-1
1	-1

fpScan - _cachedScanOnRocksDB()

- MemoryCache Build Up이 끝나면, 한개의 RowGroup를 순회

- for Rows
  for Columns
  순서로 순회

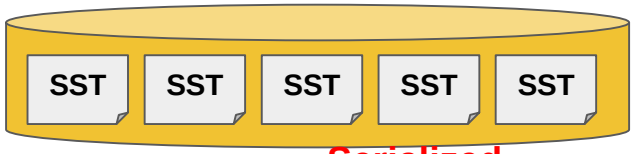
- Redis와 동일함

```
size_t _cachedScanOnRocksDB_non_vector(client *c, redisDb *db,
                                       size_t rowGroupId,
                                       ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (Vector **) zmalloc(
        sizeof(Vector *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (size_t i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    size_t numReplies = 0;
    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    // Removes created ColumnVector robjs.
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        vectorFreeDeep(cachedColumnVectorObjs[k]);
        zfree(cachedColumnVectorObjs[k]);
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);

    return numReplies;
}
```

fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	*nullptr*
1 → **3**	*nullptr*
ColumnVectorIds	
0	-1
1	-1

Rows
[

Cols
[2030, ~~1~~, zyx],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

- columnParam의 IDList에서, columnID를 가져옴
  - ex) columnParam->columnIdList
    = [1, 3]
- rowID를 columnVectorID로 변환함
  - columnVectorID
    = ((rowID - 1) / kColumnVectorSize) + 1

```
size_t numReplies = 0;
for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
    size_t rowId = j + 1;
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        size_t columnId = (long) vectorGet(&columnParam->columnIdList, k);
        size_t columnVectorId = getColumnVectorId(rowId);

        // Caching column vector.
        if (columnVectorId != cachedColumnVectorIds[k]) {
            // RocksDB Column Vector
            // RocksDB Key, which is Rocks key
            // Ex) ""
            sds dataKey = generateDataRocksKeySds(
                scanParam->dataKeyInfo, columnId, columnVectorId);
            Vector *vector = getColumnVectorFromRocksDB(db, dataKey);
            cachedColumnVectorIds[k] = columnVectorId;
            cachedColumnVectorObjs[k] = vector;
            sdsfree(dataKey);
        }

        Vector *columnVector = (Vector *) cachedColumnVectorObjs[k];
        sds value = vectorGet(columnVector, getColumnVectorIndex(rowId));
        addReplyBulkSds(c, sdsdup(value));
        numReplies++;
    }
}
```

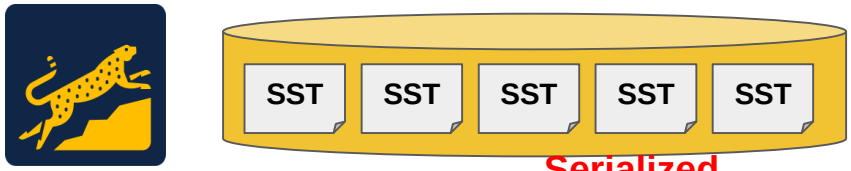
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [**1**, 3]

RowId = 1
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((1-1)/2)+1
= 0 + 1 = 1



Serialized

D:{100:2:1}:G:1:F:**1**:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:3:**1**	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	*nullptr*
1 → **3**	*nullptr*
ColumnVectorIds	
0	-1
1	-1

Rows
[

→

Cols
[**2030**, ~~1~~, zyx],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

- 현재 columnVector가 Memory cache되어있지 않은 경우
  - columnID, columnVectorID로 RocksKey를 생성함
    - ex)
      dataKey = “D:{100:2:1}:G:1”
      columnID = 1, columnVectorID = 2
      → RocksKey = “D:{100:2:1}:G:1:F:1:2”
  - RocksDB에서 RocksKey에 해당하는 ColumnVector를 가져옴
    - Deserialize
      ex) V:{T:type:N:2}:D:[2030:2031]
      → 2030 2031
  - Memory Cache에 ColumnVector와 columnVectorID를 저장함

```
size_t numReplies = 0;
for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
    size_t rowId = j + 1;
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        size_t columnId = (long) vectorGet(&columnParam->columnIdList, k);
        size_t columnVectorId = getColumnVectorId(rowId);

        // Caching column vector.
        if (columnVectorId != cachedColumnVectorIds[k]) {
            // RocksDB Column Vector
            // RocksDB Key, which is Rocks key
            // Ex) ""
            sds dataKey = generateDataRocksKeySds(
                scanParam->dataKeyInfo, columnId, columnVectorId);
            Vector *vector = getColumnVectorFromRocksDB(db, dataKey);
            cachedColumnVectorIds[k] = columnVectorId;
            cachedColumnVectorObjs[k] = vector;
            sdsfree(dataKey);
        }

        Vector *columnVector = (Vector *) cachedColumnVectorObjs[k];
        sds value = vectorGet(columnVector, getColumnVectorIndex(rowId));
        addReplyBulkSds(c, sdsdup(value));
        numReplies++;
    }
}
```

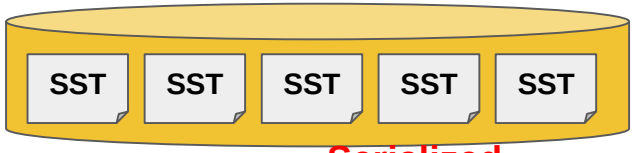
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 1
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((1-1)/2)+1
= 0 + 1 = 1



Serialized

D:{100:2:1}:G:1:F:**1**:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:3:**1**	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache

ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	*nullptr*
ColumnVectorIds	
0	**1**
1	-1

Rows
[

Cols
[**2030**, ~~1~~, zyx],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

- 현재 columnVector가 Memory cache된 경우
  - Memory Cache에서 조회하는 column에 해당하는 ColumnVector를 가져옴
  - ColumnVector에서 rowID에 해당하는 데이터를 가져옴
    - ColumnVectorIndex
      = (rowID - 1) % kColumnVectorSize
  - Client로 Reply...

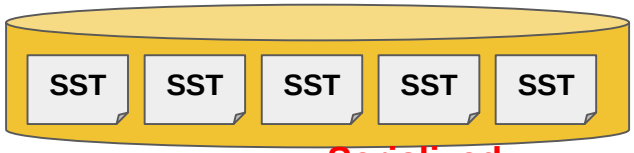
```
size_t numReplies = 0;
for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
    size_t rowId = j + 1;
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        size_t columnId = (long) vectorGet(&columnParam->columnIdList, k);
        size_t columnVectorId = getColumnVectorId(rowId);

        // Caching column vector.
        if (columnVectorId != cachedColumnVectorIds[k]) {
            // RocksDB Column Vector
            // RocksDB Key, which is Rocks key
            // Ex) ""
            sds dataKey = generateDataRocksKeySds(
                scanParam->dataKeyInfo, columnId, columnVectorId);
            Vector *vector = getColumnVectorFromRocksDB(db, dataKey);
            cachedColumnVectorIds[k] = columnVectorId;
            cachedColumnVectorObjs[k] = vector;
            sdsfree(dataKey);
        }

        Vector *columnVector = (Vector *) cachedColumnVectorObjs[k];
        sds value = vectorGet(columnVector, getColumnVectorIndex(rowId));
        addReplyBulkSds(c, sdsdup(value));
        numReplies++;
    }
}
```

fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

RowId = 1
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((1-1)/2)+1
= 0 + 1 = 1

ColumnVectorIdx
= (RowId-1)%VectorSize
= (1-1)%2
= 0

Memory Cache	
ColumnVectorObj	
0 → **1**	**2030** 2031
1 → **3**	*nullptr*
ColumnVectorIds	
0	1
1	-1

Reply “2030” To Client

Rows
[
Cols
[**2030**, ~~1~~, zyx],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],
]

Next Col...

Rows
[

Cols
[~~2030~~, ~~1~~, **zyx**],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

Memory Cache	
ColumnVectorObj	
0	2030 2031
1	*nullptr*
ColumnVectorIds	
0	1
1	-1

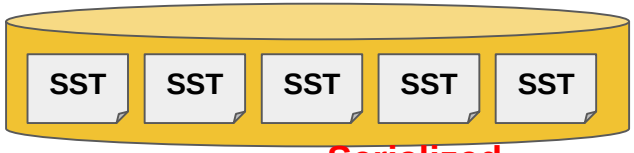
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, **3**]

RowId = 1
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((1-1)/2)+1
= 0 + 1 = 1



Serialized

D:{100:2:1}:G:1:F:1:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:**3**:**1**	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	*nullptr*
ColumnVectorIds	
0	1
1	-1

Rows
[

→

Cols
[~~2030~~, ~~1~~, **zyx**],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

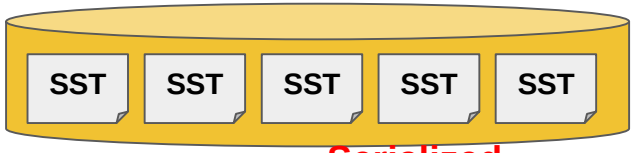
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 1
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((1-1)/2)+1
= 0 + 1 = 1



Serialized

D:{100:2:1}:G:1:F:1:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:**3**:**1**	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache

ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	**zyx wvu**
ColumnVectorIds	
0	1
1	**1**

Rows
[

Cols
[~~2030~~, ~~1~~, **zyx**],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

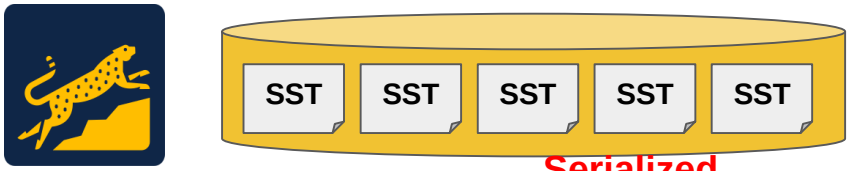
kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 1
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((1-1)/2)+1
= 0 + 1 = 1

ColumnVectorIdx
= (RowId-1)%VectorSize
= (1-1)%2
= 0



Serialized

D:{100:2:1}:G:1:F:1:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:**3**:**1**	**zyx** wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	**zyx** wvu
ColumnVectorIds	
0	1
1	1

Rows
[
→
Cols
[~~2030~~, ~~1~~, **zyx**],
[2031, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],
]

Reply “zyx” To Client

Next Row...

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[**2031**, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

Memory Cache	
ColumnVectorObj	
0	2030 2031
1	zyx wvu
ColumnVectorIds	
0	1
1	1

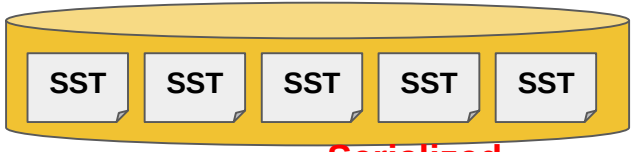
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [**1**, 3]

RowId = 2
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((2-1)/2)+1
= 0 + 1 = 1



Serialized

D:{100:2:1}:G:1:F:**1**:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:3:**1**	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	zyx wvu
ColumnVectorIds	
0	1
1	1

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[**2031**, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

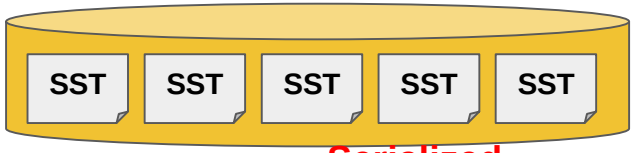
kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

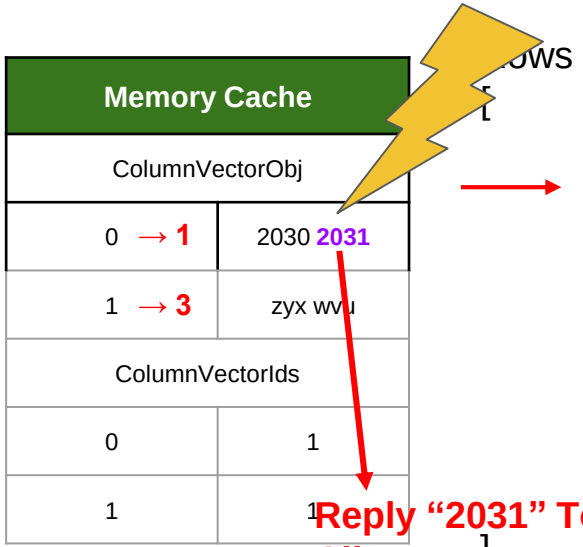
RowId = 2
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((2-1)/2)+1
= 0 + 1 = 1

ColumnVectorIdx
= (RowId-1)%VectorSize
= (2-1)%2
= 1



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc



Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[**2031**, ~~1~~, wvu],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

Next Col...

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, **wvu**],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

Memory Cache	
ColumnVectorObj	
0	2030 2031
1	zyx wvu
ColumnVectorIds	
0	1
1	1

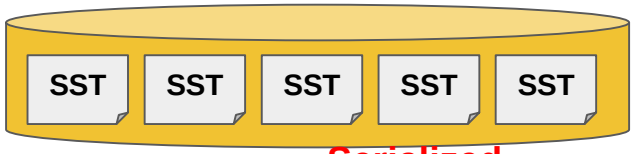
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, **3**]

RowId = 2
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((2-1)/2)+1
= 0 + 1 = 1



Serialized

D:{100:2:1}:G:1:F:1:**1**	2030 2031
D:{100:2:1}:G:1:F:2:**1**	1 1
D:{100:2:1}:G:1:F:**3**:**1**	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	zyx wvu
ColumnVectorIds	
0	1
1	1

Rows
[

→

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, **wvu**],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

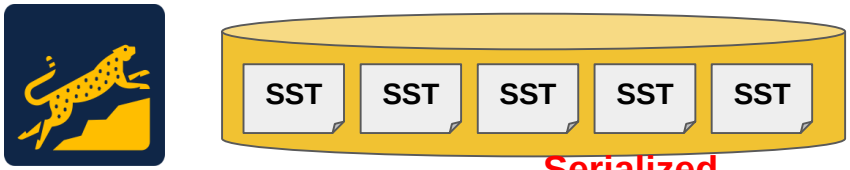
kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

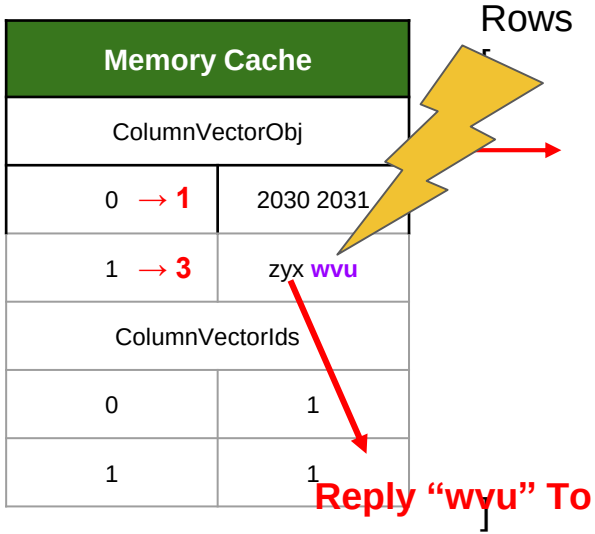
RowId = 2
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((2-1)/2)+1
= 0 + 1 = 1

ColumnVectorIdx
= (RowId-1)%VectorSize
= (2-1)%2
= 1



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc



Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, **wvu**],
[2028, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],
]

Reply “wvu” To Client

Next Vector...

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[**2028**, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

Memory Cache	
ColumnVectorObj	
0	2030 2031
1	zyx wvu
ColumnVectorIds	
0	1
1	1

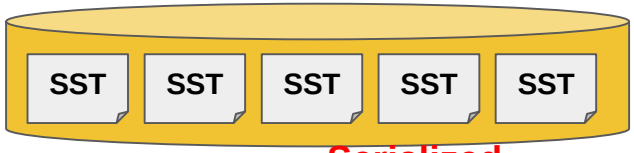
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 3
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((3-1)/2)+1
= 1 + 1 = 2



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:**1**:**2**	2028 2029
D:{100:2:1}:G:1:F:2:**2**	1 1
D:{100:2:1}:G:1:F:3:**2**	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2030 2031
1 → **3**	zyx wvu
ColumnVectorIds	
0	1
1	1

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[**2028**, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

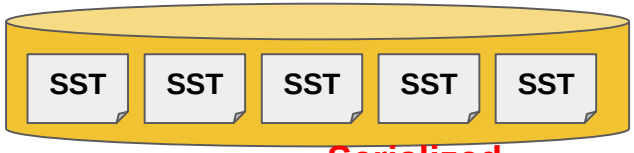
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 3
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((3-1)/2)+1
= 1 + 1 = 2



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:**1**:**2**	2028 2029
D:{100:2:1}:G:1:F:2:**2**	1 1
D:{100:2:1}:G:1:F:3:**2**	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache

ColumnVectorObj	
0 → ~~1~~	2028 2029
1 → **3**	zyx wvu
ColumnVectorIds	
0	~~1~~ → **2**
1	1

(VectorID) 1 != 2
그러므로 2로 초기화

Rows
[
Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[**2028**, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],
]

fpScan - _cachedScanOnRocksDB()

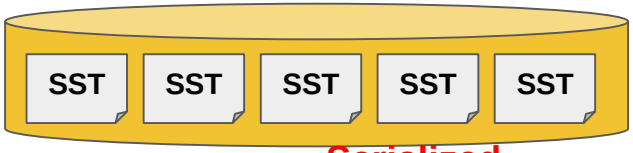
kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 3
ColIdx = 0
ColId
= ColIdList[ColIdx] = 1
VectorId
=((RowId-1)/VectorSize)+1
=((3-1)/2)+1
= 1 + 1 = 2

ColumnVectorIdx
= (RowId-1)%VectorSize
= (3-1)%2
= 0



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:**1**:**2**	**2028** 2029
D:{100:2:1}:G:1:F:2:**2**	1 1
D:{100:2:1}:G:1:F:3:**2**	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	**2028** 2029
1 → **3**	zyx wvu
ColumnVectorIds	
0	2
1	1

Rows
[
Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[**2028**, ~~1~~, tsr],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],
]

Reply “2028” To Client

Next Col...

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[~~2028~~, ~~1~~, **tsr**],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

Memory Cache	
ColumnVectorObj	
0	2028 2029
1	zyx wvu
ColumnVectorIds	
0	2
1	1

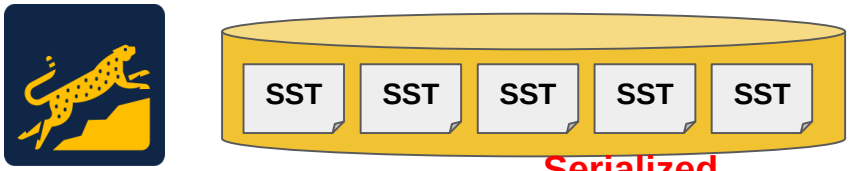
fpScan - _cachedScanOnRocksDB()

kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, **3**]

RowId = 3
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((3-1)/2)+1
= 1 + 1 = 2



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:**2**	2028 2029
D:{100:2:1}:G:1:F:2:**2**	1 1
D:{100:2:1}:G:1:F:**3**:**2**	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2028 2029
1 → **3**	zyx wvu
ColumnVectorIds	
0	2
1	1

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[~~2028~~, ~~1~~, **tsr**],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

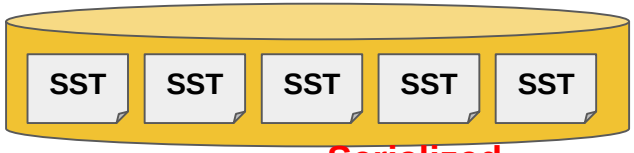
kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 3
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((3-1)/2)+1
= 1 + 1 = 2

ColumnVectorIdx
= (RowId-1)%VectorSize
= (3-1)%2
= 0



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:2	2028 2029
D:{100:2:1}:G:1:F:2:2	1 1
D:{100:2:1}:G:1:F:3:2	tsr qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

(VectorID) 1 != 2
그러므로 2로 초기화

Memory Cache	
ColumnVectorObj	
0 → 1	2028 2029
1 → 3	tsr qpo
ColumnVectorIds	
0	2
1	~~1~~ → 2

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[~~2028~~, ~~1~~, **tsr**],
[2029, ~~1~~, qpo],
[2026, ~~1~~, nml],
[2027, ~~1~~, kji],
[2024, ~~1~~, hgf],
[2025, ~~1~~, edc],

]

fpScan - _cachedScanOnRocksDB()

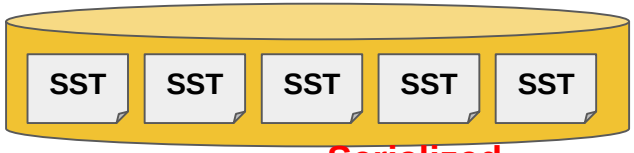
kColumnVectorSize = 2
kRowGroupSize = 8

- ex) redis-cli> fpScan D:{100:2:1} 1,3

ColIdList = [1, 3]

RowId = 3
ColIdx = 1
ColId
= ColIdList[ColIdx] = 3
VectorId
=((RowId-1)/VectorSize)+1
=((3-1)/2)+1
= 1 + 1 = 2

ColumnVectorIdx
= (RowId-1)%VectorSize
= (3-1)%2
= 0



Serialized

D:{100:2:1}:G:1:F:1:1	2030 2031
D:{100:2:1}:G:1:F:2:1	1 1
D:{100:2:1}:G:1:F:3:1	zyx wvu
D:{100:2:1}:G:1:F:1:**2**	2028 2029
D:{100:2:1}:G:1:F:2:**2**	1 1
D:{100:2:1}:G:1:F:**3**:**2**	**tsr** qpo
D:{100:2:1}:G:1:F:1:3	2026 2027
D:{100:2:1}:G:1:F:2:3	1 1
D:{100:2:1}:G:1:F:3:3	nml kji
D:{100:2:1}:G:1:F:1:4	2024 2025
D:{100:2:1}:G:1:F:2:4	1 1
D:{100:2:1}:G:1:F:3:4	hgf edc

Memory Cache	
ColumnVectorObj	
0 → **1**	2028 2029
1 → **3**	**tsr** qpo
ColumnVectorIds	
0	2
1	2

Rows
[
Cols
[~~2030~~, 1, ~~zyx~~],
[~~2031~~, 1, ~~wvu~~],
[~~2028~~, 1, **tsr**],
[2029, 1, qpo],
[2026, 1, nml],
[2027, 1, kji],
[2024, 1, hgf],
[2025, 1, edc],
]

Reply “tsr” To Client

A few moments later…….

Finished reply final row of RowGroup 1… → *D:{100:2:1}:G:1*

Rows
[

Cols
[~~2030~~, ~~1~~, ~~zyx~~],
[~~2031~~, ~~1~~, ~~wvu~~],
[~~2028~~, ~~1~~, ~~tsr~~],
[~~2029~~, ~~1~~, ~~qpo~~],
[~~2026~~, ~~1~~, ~~nml~~],
[~~2027~~, ~~1~~, ~~kji~~],
[~~2024~~, ~~1~~, ~~hgf~~],
[~~2025~~, ~~1~~, ~~edc~~],

]

Memory Cache	
ColumnVectorObj	
0	2024 2025
1	hgf edc
ColumnVectorIds	
0	4
1	4

fpScan - _cachedScanOnRocksDB()

- Memory Cached ColumnVectors
  - 한개의 RowGroup에 대한 Iteration이 끝나면 Memory Cache를 삭제함
  - Redis와는 다르게, Deserialized Vector이므로, Redis Level에 새로 생성된 Vector임
    → Deep Free

Memory Cache	
~~ColumnVectorObj~~	
~~0~~	~~2024 2025~~
~~1~~	~~hgf edc~~
~~ColumnVectorIds~~	
~~0~~	~~4~~
~~1~~	~~4~~

```
size_t _cachedScanOnRocksDB_non_vector(client *c, redisDb *db,
                                       size_t rowGroupId,
                                       ScanParameter *scanParam) {
    // RowGroup index(i) = rowGroupId - 1
    RowGroupParameter *rowGroupParam =
        &scanParam->rowGroupParams[rowGroupId - 1];
    ColumnParameter *columnParam = scanParam->columnParam;

    robj **cachedColumnVectorObjs = (Vector **) zmalloc(
        sizeof(Vector *) * columnParam->columnCount);
    int *cachedColumnVectorIds = (int *) zmalloc(
        sizeof(int) * columnParam->columnCount);
    for (size_t i = 0; i < columnParam->columnCount; ++i) {
        cachedColumnVectorObjs[i] = NULL;
        cachedColumnVectorIds[i] = -1;
    }

    size_t numReplies = 0;
    for (size_t j = 0; j < rowGroupParam->rowCount; ++j) {
        size_t rowId = j + 1;
        for (size_t k = 0; k < columnParam->columnCount; ++k) {
            // ...
        }
    }

    // Removes created ColumnVector robjs.
    for (size_t k = 0; k < columnParam->columnCount; ++k) {
        vectorFreeDeep(cachedColumnVectorObjs[k]);
        zfree(cachedColumnVectorObjs[k]);
    }

    zfree(cachedColumnVectorObjs);
    zfree(cachedColumnVectorIds);

    return numReplies;
}
```

Next RowGroup in RocksDB… → *D:{100:2:1}:G:4*

Rows
[
Cols
[2048, ~~1~~, ZYX],
[2049, ~~1~~, WVU],
[2046, ~~1~~, TSR],
[2047, ~~1~~, QPO],
[2044, ~~1~~, NML],
]

Memory Cache	
ColumnVectorObj	
0	2006 2007
1	IJK LMO
ColumnVectorIds	
0	3
1	3

rowGroupParams		
~~0~~	~~dict~~	~~nullptr~~
	~~isInRocksDB~~	~~**true**~~
	~~rowCount~~	~~8~~
2	dict	nullptr
	isInRocksDB	**true**
	rowCount	5

***Do It Yourself**
D:{100:2:1}:G:4에 대해서 cachedScanOnRocksDB

fpScan - Code Structure

fpScanCommand()

createScanParameter()

- Client의 Parameter들을 Parsing
- ScanParameter Object 생성

Parse

populateScanParameter()

- ScanParameter에 metadict data를 populate

Populate

scanDataFromADDB()

- ScanParameter로 Redis와 RocksDB에서 Data-Collect
- Return results to Client

Iterate & Collect

```
/*
 * fpScanCommand
 * Scan data from the database(Redis & RocksDB)
 * --- Parameters ---
 *  arg1: Key(Table ID & PartitionInfo ID)
 *  arg2: Column IDs to find
 *
 * --- Usage Examples ---
 *  Parameters:
 *      key: "D:{3:1:2}"
 *          tableId: "3"
 *          partitionInfoId: "1:2"
 *      columnIds: ["2", "3", "4"]
 *  Command:
 *      redis-cli> FPSCAN D:{3:2:1} 2,3,4
 *  Results:
 *      redis-cli> "20180509"
 *      redis-cli> "Do young Kim"
 *      redis-cli> "Yonsei Univ"
 *      ...
 */
void fpScanCommand(client *c) {
    serverLog(LL_DEBUG, "FPSCAN COMMAND START");

    /*Creates scan parameters*/
    ScanParameter *scanParam = createScanParameter(c);

    /*Populates row group information to scan parameters*/
    int totalDataCount = populateScanParameter(c->db, scanParam);
    serverLog(LL_DEBUG, "total data count: %d", totalDataCount);

    /*Non-Vector response version*/
    void *replylen = addDeferredMultiBulkLength(c);
    size_t numreplies = scanDataFromADDB_non_vector(c, c->db, scanParam);
    freeScanParameter(scanParam);
    setDeferredMultiBulkLength(c, replylen, numreplies);
}
```

Q & A