DIFS flow check

with nfd

2020년 04월 13일

정아름

arm@gurum.cc

목차

1. DIFS **4**

1.1. Flow check 4

1.1.1 NFD routing 5

1.1.2. FS tree 6

1.1.2.1. 처음 실행 6

1.1.2.2. 파일 put 이후 6

1.1.3. Manifest 7

1.2. InterestFilter 8

1.2.1. read 8

1.2.2. write 8

1.2.3. manifest 8

1.3. Code walkthrough 8

1.3.1. ndn part 8

1.3.1.1. registerPrefix 8

1.3.1.2. setInterestFilter 9

1.3.1.3. interest handler 9

1.3.1.4. expressInterest 9

1.3.1.5. reply handler 10

1.3.2. storage part 10

1.3.2.1. insertData 10

1.3.2.2. insertManifest 11

1.3.2.3. deleteData/erase 11

1.3.2.4. readData 12

1.3.2.5. readManifest 12

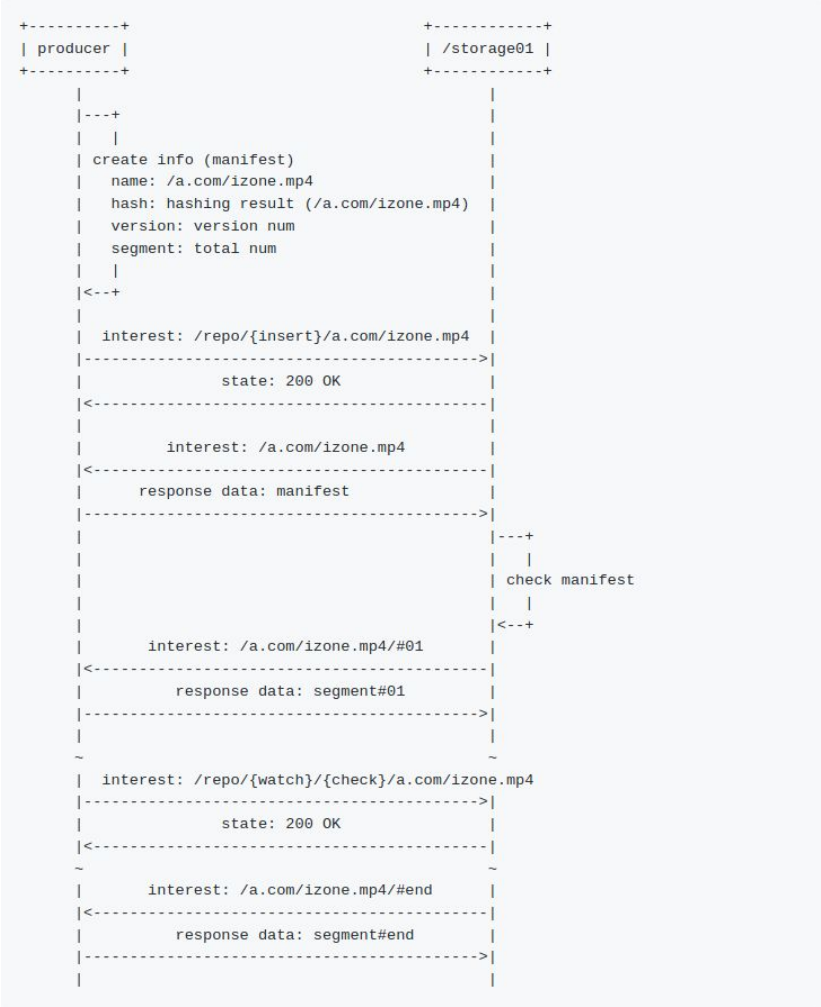
2. repo-ng **14**

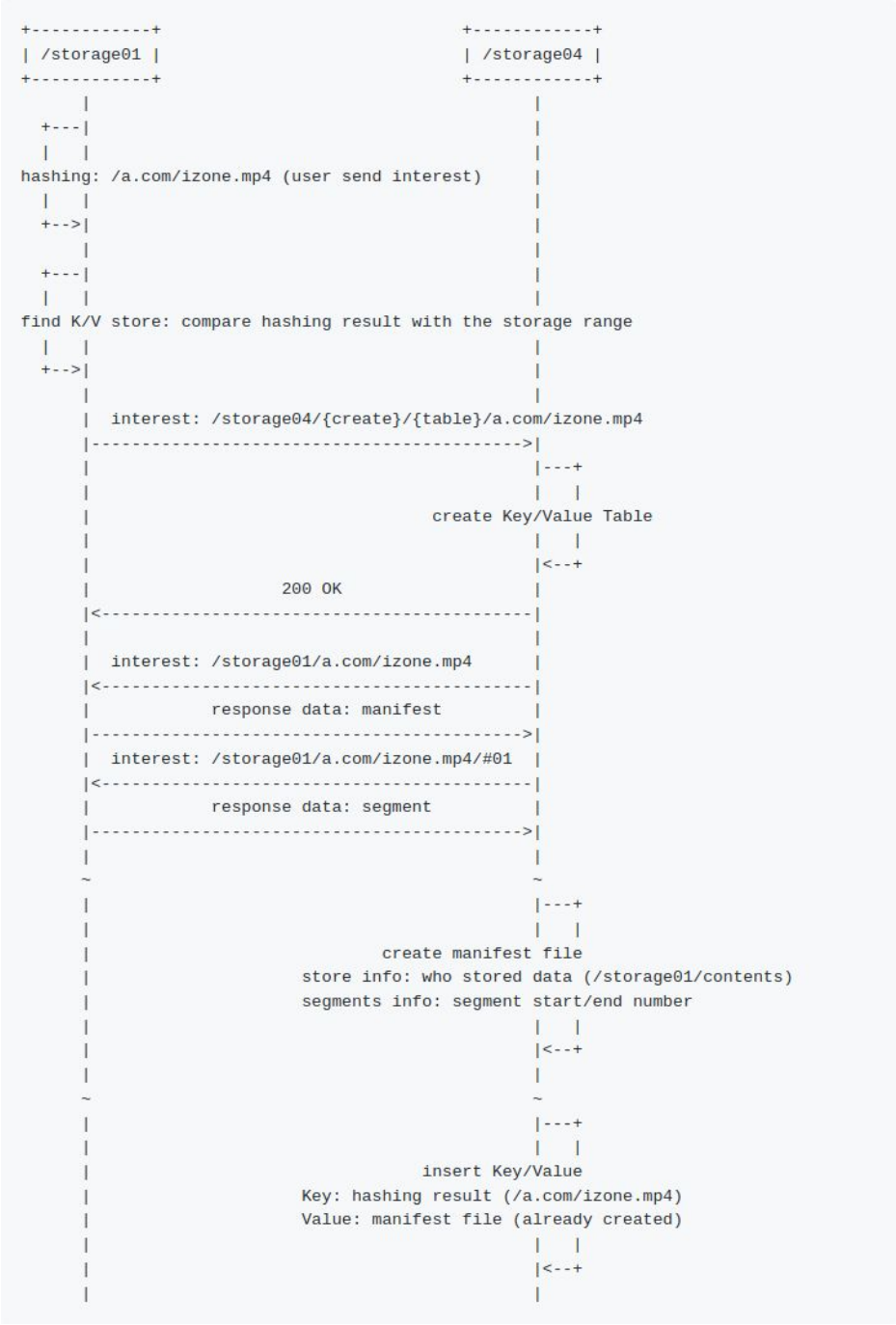
2.1. nfd routing 14

2.2. Code walkthrough 14
2.2.1. storage part 14
2.2.1.1. insertData 14
2.2.1.2. deleteData 14
2.2.1.3. readData 15
2.2.2. sqlite part 16
2.2.2.1. fullEnumerate 16
2.2.2.2. insert 17
2.2.2.3. erase 19
2.2.2.4. read 20
2.2.2.5. size 21

1. DIFS

1.1. Flow check





1.1.1 NFD routing

```
kjwon15@malformed:~/workspace/tmp-repo <(b'd606ecf')+?> % nfdc route list
prefix=/get nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/0 nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/0/data nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/localhost/nfd nexthop=267 origin=app cost=0 flags=child-inherit expires=never
```

(repo-0만 가동하였을 때)

```
kjwon15@malformed:~/workspace/tmp-repo <(b'd606ecf')+?> % nfdc route list
prefix=/get nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/get nexthop=286 origin=app cost=0 flags=child-inherit expires=never
prefix=/get nexthop=287 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo nexthop=286 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo nexthop=287 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/0 nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/0/data nexthop=284 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/1 nexthop=286 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/1/data nexthop=286 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/2 nexthop=287 origin=app cost=0 flags=child-inherit expires=never
prefix=/example/repo/2/data nexthop=287 origin=app cost=0 flags=child-inherit expires=never
prefix=/localhost/nfd nexthop=267 origin=app cost=0 flags=child-inherit expires=never
```

(repo-0..2를 모두 가동하였을 때)

prefix	
/get	GET 요청(cli to repo)
/example/repo	PUT 요청
/example/repo/0	GET 요청(manifest, data)
/example/repo/0/data	GET(data)

1.1.2. FS tree

1.1.2.1. 처음 실행

```
.
├── 0
│   ├── data
│   └── manifest
├── 1
│   ├── data
│   └── manifest
└── 2
    ├── data
    └── manifest

9 directories, 0 files
```

1.1.2.2. 파일 put 이후

```
.
```

```
├── 0
│   ├── data
│   └── manifest
├── 1
│   ├── data
│   │   └── 64
│   │       └── ae7feac04e895299c352b42b2e98219ecc4456
│   └── manifest
└── 2
    ├── data
    └── manifest
        └── bf60b7671423e439c9858fd6b417a935bb82871f

10 directories, 2 files
```

1.1.3. Manifest

```
{
  "info": {
    "name": "/data/1/%FD%00%00%01q%9A%CA%F5%0C",
    "hash": "f1cf19a772d9d4790c7a7d7a0b46509ccb96593c",
    "startBlockId": "0",
    "endBlockId": "0"
  },
  "storages": [
    {
      "storage_name": "/example/repo/0",
      "segment": {
        "start": "0",
        "end": "0"
      }
    }
  ]
}
```

1.2. InterestFilter

1.2.1. read

prefix	
/example/repo/0/data	실제 데이터
/get	매니페스트 요청 (cli to repo)

1.2.2. write

prefix	
/example/repo/insert	데이터 삽입
/example/repo/insert check	
/example/repo/0/info	매니페스트 전달(insert)

1.2.3. manifest

prefix	
/example/repo/0/create	매니페스트 생성
/example/repo/0/find	매니페스트 조회

1.3. Code walkthrough

1.3.1. ndn part

1.3.1.1. registerPrefix

```
m_face.registerPrefix(clusterPrefix, nullptr,
  [] (const Name& clusterPrefix, const std::string& reason) {
    std::cerr << "Cluster prefix " << clusterPrefix << "
registration error: " << reason << std::endl;
```

특정 prefix를 가진 인터레스트를 받을 수 있도록 NFD의 라우트 목록에 추가한다. 이를 통해 추가적으로 NFD 라우트를 추가하지 않고도 하나의 route를 이용해 모든 데이터를 처리할 수 있다.

1.3.1.2. setInterestFilter

```
getFace().setInterestFilter(
  Name(prefix).append("create"),
  bind(&ManifestHandle::onCreateInterest, this, _1, _2));

getFace().setInterestFilter(
  Name(prefix).append("find"),
  bind(&ManifestHandle::onFindInterest, this, _1, _2));
```

특정 prefix를 가진 인터레스트를 받을 수 있도록 핸들러를 등록한다. 해당 prefix가 registerPrefix를 통해 등록 된 prefix를 가지지 않는다면 스스로 route를 등록하게 된다.

1.3.1.3. interest handler

```
void
ReadHandle::onInterest(const Name& prefix, const Interest&
interest);
```

필터로 지정한 인터레스트를 받아서 처리 할 핸들러를 구현한다.

1.3.1.4. expressInterest

```
getFace().expressInterest(
  findInterest,
  bind(&ReadHandle::onFindCommandResponse, this, _1, _2,
processId),
  bind(&ReadHandle::onFindCommandTimeout, this, _1, processId), //
Nack
  bind(&ReadHandle::onFindCommandTimeout, this, _1, processId));
```

생성한 interest를 핸들러를 지정해 face를 통해 내보낸다. processId를 통해 어떤 작업인지 식별할 수 있게 되어 있다.

1.3.1.5. reply handler

```
void
WriteHandle::onSegmentData(const Interest& interest, const Data&
data, ProcessId processId);
```

1.3.2. storage part

1.3.2.1. insertData

```
int64_t
FsStorage::insert(const Data& data)
{
  return writeData(data, DIRNAME_DATA);
}

int64_t
FsStorage::writeData(const Data& data, const char* dataType)
{
  auto name = data.getName();
  auto id = hash(name.toUri());

  Index::Entry entry(data, 0);

  boost::filesystem::path fsPath = getPath(data.getName(),
dataType);
  boost::filesystem::create_directories(fsPath.parent_path());

  std::ofstream outFileData(fsPath.string(), std::ios::binary);
  outFileData.write(
      reinterpret_cast<const char*>(data.wireEncode().wire()),
      data.wireEncode().size());

  return (int64_t)id;
```

```
}
```

기존 repo-ng와 호환성을 위해 int64_t 형식의 해시를 리턴하고 fs 스토리지에 정해진 형식대로 파일 형태로 데이터를 저장한다.

1.3.2.2. insertManifest

```
std::string
FsStorage::insertManifest(const Manifest& manifest)
{
  boost::filesystem::path fsPath = m_path / DIRNAME_MANIFEST /
manifest.getHash();

  auto json = manifest.toJson();

  std::ofstream outFile(fsPath.string());
  outFile.write(
      json.c_str(),
      json.size());

  return manifest.getHash();
}
```

기존 repo-ng와의 호환성이 필요 없기 때문에 sha1 형식의 std::string을 리턴하고 fs 스토리지에 정해진 형식대로 매니페스트를 저장한다.

1.3.2.3. deleteData/erase

```
bool
FsStorage::erase(const Name& name)
{
  auto fsPath = getPath(name, DIRNAME_DATA);

  boost::filesystem::file_status fsPathStatus =
boost::filesystem::status(fsPath);
  if (!boost::filesystem::is_directory(fsPathStatus)) {
      std::cerr << name.toUri() << " is not exists" << std::endl;
```

```
    return false;
  }

  boost::filesystem::remove_all(fsPath);
  return true;
}
```

id 대신 Name을 통해 경로를 얻어와 데이터가 들어있는 디렉터리를 삭제한다. 상위 디렉터리가 비어도 삭제하지는 않는다.

1.3.2.4. readData

```
std::shared_ptr<Data>
FsStorage::read(const Name& name)
{
  auto fsPath = getPath(name.toUri(), DIRNAME_DATA);
  auto data = make_shared<Data>();

  boost::filesystem::ifstream inFileData(fsPath,
std::ifstream::binary);
  inFileData.seekg(0, inFileData.end);
  int length = inFileData.tellg();
  inFileData.seekg(0, inFileData.beg);

  char * buffer = new char [length];
  inFileData.read(buffer, length);

  data->wireDecode(Block(reinterpret_cast<const uint8_t*>(buffer),
length));

  return data;
}
```

데이터를 읽어들여 wireDecode를 통해 Data 형식으로 디코딩 해 반환한다.

1.3.2.5. readManifest

```
Manifest
FsStorage::readManifest(const std::string hash)
```

```
{
  boost::filesystem::path fsPath = m_path / DIRNAME_MANIFEST /
hash;
  boost::filesystem::ifstream inFileData(fsPath);

  std::string json(
      (std::istreambuf_iterator<char>(inFileData)),
      std::istreambuf_iterator<char>());

  return Manifest::fromJson(json);
}
```

json형식을 읽어들여 그대로 디코딩 후 반환한다.

2. repo-ng

2.1. nfd routing

prefix	
/example/data	get
/example/repo/0	insert

2.2. Code walkthrough

2.2.1. storage part

2.2.1.1. insertData

```
bool
RepoStorage::insertData(const Data& data)
{
   bool isExist = m_index.hasData(data);
   if (isExist)
      BOOST_THROW_EXCEPTION(Error("The Entry Has Already In the
Skiplist. Cannot be Inserted!"));
   int64_t id = m_storage.insert(data);
   if (id == -1)
      return false;
   bool didInsert = m_index.insert(data, id);
   if (didInsert)
      afterDataInsertion(data.getName());
   return didInsert;
}
```

인덱스에 있는 지 확인 후 데이터를 저장하고 인덱스에 추가한다.

2.2.1.2. deleteData

```
ssize_t
```

```
RepoStorage::deleteData(const Name& name)
{
  bool hasError = false;
  std::pair<int64_t,ndn::Name> idName = m_index.find(name);
  if (idName.first == 0)
      return false;
  int64_t count = 0;
  while (idName.first != 0) {
      bool resultDb = m_storage.erase(idName.first);
      bool resultIndex = m_index.erase(idName.second); //full name
      if (resultDb && resultIndex) {
      afterDataDeletion(idName.second);
      count++;
      }
      else {
      hasError = true;
      }
      idName = m_index.find(name);
  }
  if (hasError)
      return -1;
  else
      return count;
}
```

인덱스를 검사해서 해당 prefix로 시작하는 모든 데이터를 지운 뒤 지워진 갯수를 반환한다.

2.2.1.3. readData

```
shared_ptr<Data>
RepoStorage::readData(const Interest& interest) const
{
  std::pair<int64_t,ndn::Name> idName = m_index.find(interest);
  if (idName.first != 0) {
      shared_ptr<Data> data = m_storage.read(idName.first);
      if (data) {
      return data;
      }
```

```
  }
  return shared_ptr<Data>();
}
```

2.2.2. sqlite part

2.2.2.1. fullEnumerate

```
void
SqliteStorage::fullEnumerate(const std::function<void(const
Storage::ItemMeta)>& f)
{
  sqlite3_stmt* m_stmt = 0;
  int rc = SQLITE_DONE;
  string sql = string("SELECT id, name, keylocatorHash FROM
NDN_REPO;");
  rc = sqlite3_prepare_v2(m_db, sql.c_str(), -1, &m_stmt, 0);
  if (rc != SQLITE_OK)
      BOOST_THROW_EXCEPTION(Error("Initiation Read Entries from
Database Prepare error"));
  int entryNumber = 0;
  while (true) {
      rc = sqlite3_step(m_stmt);
      if (rc == SQLITE_ROW) {

      ItemMeta item;
      item.fullName.wireDecode(Block(reinterpret_cast<const
uint8_t*>(sqlite3_column_blob(m_stmt, 1)),
                                     sqlite3_column_bytes(m_stmt,
1)));
      item.id = sqlite3_column_int(m_stmt, 0);
      item.keyLocatorHash = make_shared<const
ndn::Buffer>(sqlite3_column_blob(m_stmt, 3),

sqlite3_column_bytes(m_stmt, 3));

      try {
```

```
      f(item);
      }
      catch (...) {
      sqlite3_finalize(m_stmt);
      throw;
      }
      entryNumber++;
      }
      else if (rc == SQLITE_DONE) {
      sqlite3_finalize(m_stmt);
      break;
      }
      else {
      std::cerr << "Initiation Read Entries rc:" << rc <<
std::endl;
      sqlite3_finalize(m_stmt);
      BOOST_THROW_EXCEPTION(Error("Initiation Read Entries
error"));
      }
  }
  m_size = entryNumber;
}
```

모든 아이템을 읽어와서 인덱스에 등록한다.

2.2.2.2. insert

```
int64_t
SqliteStorage::insert(const Data& data)
{
  Name name = data.getName();

  Index::Entry entry(data, 0); //the id is not used
  int64_t id = -1;
  if (name.empty()) {
      std::cerr << "name is empty" << std::endl;
      return -1;
  }
```

```
  int rc = 0;

  sqlite3_stmt* insertStmt = 0;

  string insertSql = string("INSERT INTO NDN_REPO (id, name, data,
keylocatorHash) "
                            "VALUES (?, ?, ?, ?)");

  if (sqlite3_prepare_v2(m_db, insertSql.c_str(), -1, &insertStmt,
0) != SQLITE_OK) {
      sqlite3_finalize(insertStmt);
      std::cerr << "insert sql not prepared" << std::endl;
  }
  //Insert
  auto result = sqlite3_bind_null(insertStmt, 1);
  if (result == SQLITE_OK) {
      result = sqlite3_bind_blob(insertStmt, 2,
                            entry.getName().wireEncode().wire(),
                            entry.getName().wireEncode().size(),
SQLITE_STATIC);
  }
  if (result == SQLITE_OK) {
      result = sqlite3_bind_blob(insertStmt, 3,
                            data.wireEncode().wire(),
                            data.wireEncode().size(),
SQLITE_STATIC);
  }
  if (result == SQLITE_OK) {
      BOOST_ASSERT(entry.getKeyLocatorHash()->size() ==
ndn::util::Sha256::DIGEST_SIZE);
      result = sqlite3_bind_blob(insertStmt, 4,
                            entry.getKeyLocatorHash()->data(),
                            entry.getKeyLocatorHash()->size(),
SQLITE_STATIC);
  }

  if (result == SQLITE_OK) {
      rc = sqlite3_step(insertStmt);
      if (rc == SQLITE_CONSTRAINT) {
      std::cerr << "Insert failed" << std::endl;
      sqlite3_finalize(insertStmt);
```

```
      BOOST_THROW_EXCEPTION(Error("Insert failed"));
      }
      sqlite3_reset(insertStmt);
      m_size++;
      id = sqlite3_last_insert_rowid(m_db);
  }
  else {
      BOOST_THROW_EXCEPTION(Error("Some error with insert"));
  }

  sqlite3_finalize(insertStmt);
  return id;
}
```

3개의 blob을 바인드 한 후 insert 한 후 id를 리턴한다.

2.2.2.3. erase

```
bool
SqliteStorage::erase(const int64_t id)
{
  sqlite3_stmt* deleteStmt = 0;

  string deleteSql = string("DELETE from NDN_REPO where id = ?;");

  if (sqlite3_prepare_v2(m_db, deleteSql.c_str(), -1, &deleteStmt,
0) != SQLITE_OK) {
      sqlite3_finalize(deleteStmt);
      std::cerr << "delete statement prepared failed" << std::endl;
      BOOST_THROW_EXCEPTION(Error("delete statement prepared
failed"));
  }

  if (sqlite3_bind_int64(deleteStmt, 1, id) == SQLITE_OK) {
      int rc = sqlite3_step(deleteStmt);
      if (rc != SQLITE_DONE && rc != SQLITE_ROW) {
      std::cerr << " node delete error rc:" << rc << std::endl;
      sqlite3_finalize(deleteStmt);
      BOOST_THROW_EXCEPTION(Error(" node delete error"));
```

```
      }
      if (sqlite3_changes(m_db) != 1)
      return false;
      m_size--;
  }
  else {
      std::cerr << "delete bind error" << std::endl;
      sqlite3_finalize(deleteStmt);
      BOOST_THROW_EXCEPTION(Error("delete bind error"));
  }
  sqlite3_finalize(deleteStmt);
  return true;
}
```

id를 받아서 해당 row를 삭제한다.

2.2.2.4. read

```
shared_ptr<Data>
SqliteStorage::read(const int64_t id)
{
  sqlite3_stmt* queryStmt = 0;
  string sql = string("SELECT * FROM NDN_REPO WHERE id = ? ;");
  int rc = sqlite3_prepare_v2(m_db, sql.c_str(), -1, &queryStmt,
0);
  if (rc == SQLITE_OK) {
      if (sqlite3_bind_int64(queryStmt, 1, id) == SQLITE_OK) {
      rc = sqlite3_step(queryStmt);
      if (rc == SQLITE_ROW) {
      auto data = make_shared<Data>();
      data->wireDecode(Block(reinterpret_cast<const
uint8_t*>(sqlite3_column_blob(queryStmt, 2)),
                             sqlite3_column_bytes(queryStmt, 2)));
      sqlite3_finalize(queryStmt);
      return data;
      }
      else if (rc == SQLITE_DONE) {
      return nullptr;
      }
```

```
      else {
      std::cerr << "Database query failure rc:" << rc << std::endl;
      sqlite3_finalize(queryStmt);
      BOOST_THROW_EXCEPTION(Error("Database query failure"));
      }
      }
      else {
      std::cerr << "select bind error" << std::endl;
      sqlite3_finalize(queryStmt);
      BOOST_THROW_EXCEPTION(Error("select bind error"));
      }
      sqlite3_finalize(queryStmt);
  }
  else {
      sqlite3_finalize(queryStmt);
      std::cerr << "select statement prepared failed" << std::endl;
      BOOST_THROW_EXCEPTION(Error("select statement prepared
failed"));
  }
  return nullptr;
}
```

id를 받아서 쿼리를 날린 후 data를 디코딩 해서 반환한다.

2.2.2.5. size

```
int64_t
SqliteStorage::size()
{
  sqlite3_stmt* queryStmt = 0;
  string sql("SELECT count(*) FROM NDN_REPO ");
  int rc = sqlite3_prepare_v2(m_db, sql.c_str(), -1, &queryStmt,
0);
  if (rc != SQLITE_OK)
      {
      std::cerr << "Database query failure rc:" << rc << std::endl;
      sqlite3_finalize(queryStmt);
      BOOST_THROW_EXCEPTION(Error("Database query failure"));
      }
```

```
  rc = sqlite3_step(queryStmt);
  if (rc != SQLITE_ROW)
      {
      std::cerr << "Database query failure rc:" << rc << std::endl;
      sqlite3_finalize(queryStmt);
      BOOST_THROW_EXCEPTION(Error("Database query failure"));
      }

  int64_t nDatas = sqlite3_column_int64(queryStmt, 0);
  if (m_size != nDatas) {
      std::cerr << "The size of database is not correct! " <<
std::endl;
  }
  return nDatas;
}
```

select count(*) 쿼리를 날려 행의 갯수를 반환한다.