**Sqoop 사용자 가이드 (v1.3.0-cdh3u3)**

[**http://mczo.blogspot.com/2012/04/sqoop.html**](http://mczo.blogspot.com/2012/04/sqoop.html)

**목차**

[1. 소개](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_introduction)

[2. 지원 배포판](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_supported_releases)

[3. Sqoop 배포판](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_sqoop_releases)

[4. 사전지식](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_prerequisites)

[5. 기본적인 사용](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_basic_usage)

[6. Sqoop 도구](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_sqoop_tools)

[6.1. 명령 별칭 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_command_aliases)

[6.2. Hadoop 설치본 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_the_hadoop_installation)

[6.3. Generic/Specific 인자 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_generic_and_specific_arguments)

[6.4. 인자를 넘기기 위하여 옵션 파일 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_options_files_to_pass_arguments)

[6.5. 도구 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_tools)

[7. sqoop-import](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_import_literal)

[7.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose)

[7.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax)

[7.2.1. 데이터베이스 서버 연결하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_connecting_to_a_database_server)

[7.2.2. 가져올 데이터 선택하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_selecting_the_data_to_import)

[7.2.3. 자유형 쿼리로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_free_form_query_imports)

[7.2.4. 병렬 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_parallelism)

[7.2.5. 가져오기 프로세스 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_the_import_process)

[7.2.6. 타입 매핑 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_type_mapping)

[7.2.7. 증가분 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_incremental_imports)

[7.2.8. 파일 포맷](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_file_formats)

[7.2.9. 커다란 오브젝트](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_large_objects)

[7.2.10. 데이터를 Hive로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_importing_data_into_hive)

[7.2.11. 데이터를 HBase로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_importing_data_into_hbase)

[7.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations)

[8. sqoop-import-all-tables](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_import_all_tables_literal)

[8.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_2)

[8.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_2)

[8.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_2)

[9. sqoop-export](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_export_literal)

[9.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[9.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[9.3. Insert 대 Update](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_inserts_vs_updates)

[9.4. 내보내기와 트랜잭션](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_exports_and_transactions)

[9.5. 실패한 내보내기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_failed_exports)

[9.6. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[10. 저장된 잡](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_saved_jobs)

[11. sqoop-job](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_job_literal)

[11.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_4) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[11.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_4) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[11.3. 저장된 잡과 비밀번호](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_saved_jobs_and_passwords)

[11.4. 저장된 잡과](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_saved_jobs_and_incremental_imports) [증가분](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_incremental_imports) [가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_saved_jobs_and_incremental_imports)

[12. sqoop-metastore](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_metastore_literal)

[12.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_5) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[12.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_5) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[13. sqoop-merge](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_merge_literal)

[13.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_6) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[13.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_6) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[14. sqoop-codegen](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_codegen_literal)

[14.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_7) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[14.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_7) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[14.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_4) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[15. sqoop-create-hive-table](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_create_hive_table_literal)

[15.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_8) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[15.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_8) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[15.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_5) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[16. sqoop-eval](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_eval_literal)

[16.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_9) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[16.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_9) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[16.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_6) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[17. sqoop-list-databases](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_list_databases_literal)

[17.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_10) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[17.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_10) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[17.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_7) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[18. sqoop-list-tables](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_list_tables_literal)

[18.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_11) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[18.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_11) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[18.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_8) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[19. sqoop-help](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_help_literal)

[19.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_12) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[19.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_12) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[19.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_9) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[20. sqoop-version](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_sqoop_version_literal)

[20.1.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_13) [목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[20.2.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_13) [문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[20.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_10) [예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

[21. 호환성 정보](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_compatibility_notes)

[21.1. 지원 데이터베이스](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_supported_databases)

[21.2. MySQL](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_mysql)

[21.2.1. zeroDateTimeBehavior](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_zerodatetimebehavior)

[21.2.2. UNSIGNED 컬럼](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_unsigned_literal_columns)

[21.2.3. BLOB과 CLOB 컬럼](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_blob_literal_and_literal_clob_literal_columns)

[21.2.4. Direct-mode 트랜잭션](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_direct_mode_transactions)

[21.3. Oracle](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle)

[21.3.1. 날짜와 시간](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_dates_and_times)

[21.4. Hive에서의 스키마 정의](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_schema_definition_in_hive)

[22. 지원 얻기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_getting_support)

[23. 트러블슈팅](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_troubleshooting)

[23.1. 일반적인 트러블슈팅 프로세스](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_general_troubleshooting_process)

[23.2. 특화된 트러블슈팅 팁](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_specific_troubleshooting_tips)

[23.2.1. Oracle: 연결 리셋 에러](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle_connection_reset_errors)

[23.2.2. Oracle: Case-Sensitive 카탈로그 쿼리 에러](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle_case_sensitive_catalog_query_errors)

23.2.3. MySQL: 연결 실패

**1. 소개**

Sqoop은 Hadoop과 관계형 데이터베이스 간에 데이터를 전송할 수 있도록 설계된 도구이다. 당신은 MySQL이나 Oracle과 같은 관계형 데이터베이스 관리 시스템(RDBMS)으로부터 Hadoop 분산 파일 시스템(HDFS)으로 데이터를 가져오는 데에 Sqoop을 사용할 수 있다.

Sqoop은 앞서 말한 대부분의 프로세스를 자동화하는데, 가져올 데이터의 스키마를 설명하기 위해 해당 데이터베이스에 의존한다. Sqoop은 데이터를 가져오고 내보내는 데에 병렬연산과 고장방지기능을 제공하는 맵리듀스를사용한다.

이 문서는 Sqoop을 사용하여 데이터베이스와 Hadoop간에 데이터를 이동시키는 기초 방법을 설명하고, Sqoop 명령줄 도구 스위트의 사용에 대한 참조 정보를 제공한다. 이 문서는 다음을 대상으로 한다. :

* 시스템과 어플리케이션 프로그래머
* 시스템 관리자
* 데이터베이스 관리자
* 데이터 분석가
* 데이터 엔지니어

**2. 지원 배포판**

이 문서는 Sqoop v1.3.0-cdh3u3에 적용된다.

**3. Sqoop 배포판**

Sqoop은 아파치 소프트웨어 재단의 오픈소스 소프트웨어 제품이다.

Sqoop 소프트웨어 개발은 <http://svn.apache.org/repos/asf/incubator/sqoop/trunk>에서 진행한다. 이 사이트에서 다음 정보들을 얻을 수 있다. :

* Sqoop의 새 릴리즈와 가장 최근의 소스코드
* 이슈 추적기
* Sqoop 문서를 포함한 위키

Sqoop은 아파치 Hadoop 0.21과 Cloudera’s Distribution of Hadoop 버전 3과 호환된다.

**4. 사전지식**

이 제품을 위하여 다음의 사전지식이 필요하다. :

* 기본적인 컴퓨터 기술과 용어
* bash와 같은 명령줄 인터페이스에 대한 친숙함
* 관계형 데이터베이스 관리 시스템
* Hadoop의 목적과 사용에 대한 기본적인 친숙함

Sqoop을 사용하기 전에 Hadoop 배포판이 설치되고 설정되어 있어야 한다. 리눅스 시스템 상에서 Hadoop을 쉽게 설치하기 위하여 Cloudera Software Archive ([http://archive.cloudera.com](http://archive.cloudera.com/))에서 Cloudera’s Distribution for Hadoop (CDH3)을 내려받는 것을 추천한다.

이 문서는 당신이 리눅스 또는 리눅스와 같은 환경을 사용하고 있다고 가정한다. 만약 윈도우즈를 사용하고 있다면, 뒤따르는 대부분의 작업들을 수행하기 위하여 cygwin을 사용할 수도 있다. 만약 Mac OS X를 사용하고 있다면, 일부의 호환성 에러들을 보게 될 것이다. Sqoop은 리눅스상에서 거의 사용되고 시험되었다.

**5. 기본적인 사용**

Sqoop을 이용하여 데이터를 관계형 데이터베이스로부터 HDFS로 가져올 수 있다. 가져오기 프로세스의 입력은 데이터베이스 테이블이다. Sqoop은 테이블을 행단위로 HDFS로 읽어들일 것이다. 이 가져오기 프로세스의 출력은 가져온 테이블의 복사본을 포함하는 파일집합이다. 가져오기 프로세스는 병렬로 수행된다. 이러한 이유로 출력은 복수개의 파일이 된다. 이 파일들은 구분된 (예를 들면, 각 필드를 구분하기 위하여 콤마나 탭을 사용한) 텍스트 파일 또는 바이너리 Avro 또는 직렬화된 레코드 데이터를 포함하는 SequenceFile들이 될 수 있다.

가져오기 프로세스의 부산물은 가져온 테이블의 각 행을 캡슐화하기 위해 생성된 자바 클래스이다. 이 클래스는 가져오기 프로세스 도중에 Sqoop 그 자신에 의하여 사용된다. 이 클래스의 자바 소스코드는 차후의 맵리듀스 데이터 처리를 위하여 당신에게도 제공된다. 이 클래스는 데이터를 SequenceFile 포맷으로 직렬화되고 SequenceFile 포맷으로부터 역직렬화할 수 있다. 또한 레코드의 구분된 텍스트 형태를 구문분석(parse)도 할 수 있다. 이러한 기능은 데이터처리 파이프라인에서 HDFS에 저장된 레코드를 사용하는 맵리듀스 어플리케이션을 빠르게 개발할 수 있게 한다. 구분된 레코드 데이터를 선호하는 다른 도구를 사용하여 구문분석(parse)하는 것도 물론 가능하다.

가져온 레코드를 처리한 이후 (예를 들어, 맵리듀스 또는 Hive 사용), 결과 데이터 집합을 관계형 데이터베이스로 도로 내보내는 경우가 있을 수 있다. 외부 어플리케이션이나 사용자의 사용을 위하여 Sqoop의 내보내기 프로세스는 병렬로 HDFS로부터 구분된 텍스트 집합을 읽고, 레코드의 형태로 구문분석(parse)하고, 목적 데이터베이스 테이블의 새 행으로서 그 레코드들을 삽입하게 된다.

Sqoop은 해당 데이터베이스를 검사할 수 있는 다른 명령들을 포함하고 있다. 예를 들면, (sqoop-list-databases 도구를 이용하여) 사용가능한 데이터베이스 스키마와 (sqoop-list-tables 도구를 이용하여) 스키마 내의 테이블의 목록을 얻을 수 있다. Sqoop은 원시 SQL 실행 쉘(sqoop-eval 도구)도 갖고 있다.

가져오기, 코드 생성, 내보내기 프로세스들의 대부분은 원하는 대로 바꿀 수 있다. 가져온 특정 행범위나 컬럼을 제어할 수 있다. 파일 기반의 데이터 표현과 사용되는 데이터 포맷을 위하여 특정한 구분자와 이스케이프 문자를 정할 수 있다. 생성된 코드에서 사용되는 클래스명과 패키지명을 제어할 수 있다. 이 문서의 차후 섹션에서 이와 관련된 인자들와 다른 인자들을 Sqoop에 명시하는 방법에 대하여 설명할 것이다.

**6. Sqoop 도구**

[6.1. 명령 별칭 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_command_aliases)

[6.2. Hadoop 설치 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_the_hadoop_installation)

[6.3. Generic/Specific 인자 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_generic_and_specific_arguments)

[6.4. 인자를 넘기기 위하여 옵션 파일 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_options_files_to_pass_arguments)

[6.5. 도구 사용하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_using_tools)

Sqoop은 연관된 도구의 집합이다. Sqoop을 사용하기 위해서 사용하고자 하는 도구를 정하고 그 도구를 제어하는 인자들을 정할 것이다.

만약 Sqoop이 원래의 소스로부터 컴파일되었다면, bin/sqoop 프로그램을 구동하여 형식상의 설치과정 없이 Sqoop을 실행할 수 있다. 패키징된 Sqoop 배포(Cloudera’s Distribution for Hadoop이 포함된 RPM)을 이용하는 사람들은 이 프로그램을/usr/bin/sqoop 와 같이 프로그램이 설치된 것을 확인할 수 있다. 이 문서의 나머지에서는 이 프로그램을 sqoop이라고 언급할 것이다. 예 :

$ sqoop 도구이름 [도구인자]

|  |  |
| --- | --- |
|  | Note |
|  | $ 문자로 시작되는 다음의 예시들은 해당 명령이 (bash와 같은) 터미널 프롬프트에서 입력되어야 함을 나타내는 것이다. $ 문자는 프롬프트 자체를 나타내는 것이므로 $ 문자를 시작으로 명령어를 입력해서는 안된다. 텍스트 단락 안에 인라인으로 명령을 입력할 수 있다(예 : sqoop help). 이 예시들은 $ 접두사를 보여주지 않지만, 같은 방식으로 그것들을 입력해야 한다. 환경변수 앞에 오는 $가 있는 예시에서는 $ 쉘 프롬프트와 혼동하지 말라. 예를 들면, 문자열 그대로의 $HADOOP\_HOME은 “$”을 포함한다. |

Sqoop은 도움말 도구를 가지고 있다. 사용할 수 있는 모든 도구의 목록을 보려면 다음의 명령을 입력한다. :

$ sqoop help  
usage: sqoop COMMAND [ARGS]  
  
Available commands:  
codegen Generate code to interact with database records  
create-hive-table Import a table definition into Hive  
eval Evaluate a SQL statement and display the results  
export Export an HDFS directory to a database table  
help List available commands  
import Import a table from a database to HDFS  
import-all-tables Import tables from a database to HDFS  
list-databases List available databases on a server  
list-tables List available tables in a database  
version Display version information  
  
See 'sqoop help COMMAND' for information on a specific command.

sqoop help (tool-name)을 입력하면 특정 도구에 대한 도움말을 조회할 수 있다. (예 : sqoop help import)

--help 인자를 어떠한 명령에도 추가할 수 있다. (예 : sqoop import --help)

**6.1. 명령 별칭 사용하기**

sqoop (도구명) 문법을 입력하는 것 뿐만 아니라 sqoop-(도구명) 문법을 명시하는 별칭 스크립트들을 사용할 수 있다. 예를 들면, sqoop-import, sqoop-export 등의 각 스크립트는 특정 도구를 선택한다.

**6.2. Hadoop 설치 제어하기**

Hadoop이 제공하는 프로그램 시작 기능을 통하여 Sqoop을 작동시킨다. sqoop 명령줄 프로그램은 Hadoop에 포함된 bin/hadoop 스크립트를 실행하는 래퍼이다. 여러개의 Hadoop 설치본이 존재한다면, $HADOOP\_HOME 환경변수를 설정하여 Hadoop 설치본을 선택할 수 있다.

예 :

$ HADOOP\_HOME=/path/to/some/hadoop sqoop import --arguments...

또는:

$ export HADOOP\_HOME=/some/path/to/hadoop  
$ sqoop import --arguments...

$HADOOP\_HOME 설정되어 있지 않다면, Sqoop 은 Cloudera’s Distribution for Hadoop의 기본 설치위치인 /usr/lib/hadoop을 사용할 것이다.

$HADOOP\_CONF\_DIR 환경변수가 설정되어 있지 않다면, $HADOOP\_HOME/conf/로부터 Hadoop 설정을 로딩한다.

**6.3. Generic/Specific 인자 사용하기**

각 Sqoop 도구의 작동을 제어하기 위하여 포괄적인(generic) 인자와 도구에 특정한(specific) 인자들을 사용한다. 예를 들면:

$ sqoop help import  
usage: sqoop import [GENERIC-ARGS] [TOOL-ARGS]  
  
Common arguments:  
--connect <jdbc-uri> Specify JDBC connect string  
--connect-manager <jdbc-uri> Specify connection manager class to use  
--driver <class-name> Manually specify JDBC driver class to use  
--hadoop-home <dir> Override $HADOOP\_HOME  
--help Print usage instructions  
-P Read password from console  
--password <password> Set authentication password  
--username <username> Set authentication username  
--verbose Print more information while working  
  
[...]  
  
Generic Hadoop command-line arguments:  
(must preceed any tool-specific arguments)  
Generic options supported are  
-conf <configuration file> specify an application configuration file  
-D <property=value> use value for given property  
-fs <local|namenode:port> specify a namenode  
-jt <local|jobtracker:port> specify a job tracker  
-files <comma separated list of files> specify comma separated files to be copied to the map reduce cluster  
-libjars <comma separated list of jars> specify comma separated jar files to include in the classpath.  
-archives <comma separated list of archives> specify comma separated archives to be unarchived on the compute machines.  
  
The general command line syntax is  
bin/hadoop command [genericOptions] [commandOptions]

도구명 뒤에, 그리고 도구에 (--connect와 같은) 특정한 인자를 쓰기 **전**에 -conf, -D 등의 포괄적인 인자를 사용해야 한다. 도구에 특정한 인자가 -P와 같은 단일 문자 인자가 아닌 경우에는 두 개의 대쉬(-)로 시작되는 반면에 포괄적인 Hadoop의 인자들은 단일 대쉬로 시작된다는 것에 주의하라.

-conf, -D, -fs, -jt 인자들은 환경설정과 Hadoop의 서버설정을 제어한다. 예를 들면, -D mapred.job.name=<잡명>은 Sqoop이 시작시키는 MR잡의 이름을 설정하는 데에 사용될 수 있다. 그런데 만약 이 인자를 사용하지 않으면, 사용된 테이블명으로부터 만들어진 잡 jar 파일명을 기본값으로 설정된다.

-files, -libjars, -archives 인자들은 통상적으로 Sqoop과 함께 쓰이지 않지만, Hadoop의 내부적인 인자 구문분석(parsing) 체계의 일부분으로서 포함된다.

**6.4. 인자를 넘기기 위하여 옵션 파일 사용하기**

Sqoop을 사용할 때, 매 명령실시 마다 변하지 않는 명령줄 옵션들은 편의를 위하여 옵션 파일에 넣어둘 수 있다. 옵션 파일은 옵션 파일이 별도로 명령줄에 표시되도록 하게 하기 위하여 옵션을 옵션 파일 내의 각 줄에 명시하는 텍스트 파일이다. 옵션 파일은 줄 중간에 백슬래시 문자로 끝냄으로써 복수개의 줄에 단일 옵션을 명시할 수 있게 한다. 또한 옵션 파일 내에 해시 문자로 시작되는 주석 기능을 지원한다. 주석은 새로운 줄에 명시가 되어야 하며 옵션 텍스트와 섞여서 사용될 수 없다. 모든 주석과 빈 줄은 옵션 파일들이 명령줄 상에 풀릴 때 무시된다. 옵션이 인용 문자열로 표시되지 않는다면, 앞서거나 뒤따르는 공백들은 모두 무시된다.

옵션 파일들 내에서 옵션들이 옵션 순서에 관하여 별도로 지시된 규정을 따르는 한, 명령줄의 어느 곳에서도 명시될 수 있다. 예를 들면, 옵션들이 어디에서 로딩이 되는 지에 상관 없이 포괄적인 옵션들이 제일 먼저, 도구에 특정한 옵션들이 그 다음에, 최종적으로 자식 프로그램에 넘겨지는 옵션들이 뒤따르는 순서를 지켜야 한다.

옵션 파일을 명시하기 위해서는 편리한 위치에 옵션 파일을 단순하게 만들고 명령줄 상에서 --options-file 인자를 통하여 파일위치를 넘기면 된다.

옵션 파일이 명시되는 모든 경우에서 도구가 작동되기 전에 명령줄 상에 옵션 파일의 내용이 풀린다.

예를 들면, 가져오기 작업을 위한 다음의 Sqoop 실시는 아래에 보이는 바와 같이 다른 방식으로 명시될 수 있다. :

$ sqoop import --connect jdbc:mysql://localhost/db --username foo --table TEST  
  
$ sqoop --options-file /users/homer/work/import.txt --table TEST

옵션 파일 /users/homer/work/import.txt는 다음의 내용을 포함한다. :

import  
--connect  
jdbc:mysql://localhost/db  
--username  
foo

옵션 파일은 가독성을 높이기 위하여 빈 줄과 주석들을 포함할 수 있다. 그래서 위의 예시는 옵션 파일 /users/homer/work/import.txt가 다음과 같은 내용을 포함하는 경우에도 정확히 같은 동작을 할 것이다. :

#  
# Options file for Sqoop import  
#  
  
# Specifies the tool being invoked  
import  
  
# Connect parameter and value  
--connect  
jdbc:mysql://localhost/db  
  
# Username parameter and value  
--username  
foo  
  
#  
# Remaining options should be specified in the command line.  
#

**6.5. 도구 사용하기**

다음 섹션부터는 각 도구의 사용을 설명할 것이다. 각 도구들은 당신에게 가장 유용할 수 있는 순서대로 나열되어 있다.

**7. sqoop-import**

[7.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose)

[7.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax)

[7.2.1. 데이터베이스 서버 연결하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_connecting_to_a_database_server)

[7.2.2. 가져올 데이터 선택하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_selecting_the_data_to_import)

[7.2.3. 자유형 쿼리로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_free_form_query_imports)

[7.2.4. 병렬 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_parallelism)

[7.2.5. 가져오기 프로세스 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_the_import_process)

[7.2.6. 타입 매핑 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_type_mapping)

[7.2.7. 증가분 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_incremental_imports)

[7.2.8. 파일 포맷](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_file_formats)

[7.2.9. 커다란 오브젝트](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_large_objects)

[7.2.10. 데이터를 Hive로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_importing_data_into_hive)

[7.2.11. 데이터를 HBase로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_importing_data_into_hbase)

[7.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations)

**7.1. 목적**

import 도구는 RDMBS로부터 HDFS로 개별 테이블을 가져온다. 테이블의 각 행은 HDFS의 독립 레코드로 표현된다. 레코드들은 (한 줄 당 한 레코드씩 인) 텍스트 파일 또는 Avro와 SequenceFile과 같은 2진(binary) 표현으로 저장될 수 있다.

**7.2. 문법**

[7.2.1. 데이터베이스 서버 연결하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_connecting_to_a_database_server)

[7.2.2. 가져올 데이터 선택하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_selecting_the_data_to_import)

[7.2.3. 자유형 쿼리로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_free_form_query_imports)

[7.2.4. 병렬 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_parallelism)

[7.2.5. 가져오기 프로세스 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_the_import_process)

[7.2.6. 타입 매핑 제어하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_controlling_type_mapping)

[7.2.7. 증가분 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_incremental_imports)

[7.2.8. 파일 포맷](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_file_formats)

[7.2.9. 커다란 오브젝트](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_large_objects)

[7.2.10. 데이터를 Hive로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_importing_data_into_hive)

[7.2.11. 데이터를 HBase로 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_importing_data_into_hbase)

$ sqoop import (포괄 인자들) (import 인자들)  
$ sqoop-import (포괄 인자들) (import 인자들)

Hadoop의 포괄적인 인자들은 import 인자들보다 앞에 있어야 하지만, import 인자들끼리는 순서에 상관 없이 입력할 수 있다.

|  |  |
| --- | --- |
|  | Note |
|  | 이 문서에서 인자들은 기능 단위로 조직되어 무리를 짓는다. 어떤 무리들(예 : “공통" 인자들)은 여러 도구에서 쓰인다. 그것들의 기능에 대한 상세 설명은 이 문서에서 최초 표시 시 한번만 제공된다. |

**표 1. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <dir> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

***7.2.1. 데이터베이스 서버 연결하기***

Sqoop은 데이터베이스로부터 HDFS에 테이블을 가져오도록 설계되었다. 그렇게 하려면, 데이터베이스에 연결하는 방법을 설명하는 *연결 문자열*을 명시해야만 한다. *연결 문자열*은 URL과 유사하며 --connect 인자로 Sqoop과 의사소통이 된다. 이것은 연결할 서버와 데이터베이스를 나타낼 뿐만 아니라 포트도 명시할 수 있다. 예를 들면 :

$ sqoop import --connect jdbc:mysql://database.example.com/employees

이 문자열은 호스트 database.example.com에 있으며 employees라는 이름을 가진 MySQL 데이터베이스에 연결하겠다는 것이다. 분산 Hadoop 클러스터를 사용하고자 한다면 URL에 localhost를 사용하면 **안된다**는 점은 중요하다. 연결 문자열은 맵리듀스 클러스터에 퍼져있는 TaskTracker 노드들상에서 사용된다. 그런데 localhost를 그대로 명시한다면, 각 노드는 서로 다른 데이터베이스에 접근하거나 어떤 데이터베이스에도 접근하지 않을 것이다. 그 대신, 모든 원격 노드에서 인식될 수 있는 데이터베이스 호스트의 완전한 호스트명이나 IP주소를 사용해야만 한다.

아마도 데이터베이스에 접근하기 전에 데이터베이스에 대하여 인증할 필요가 있을 것이다. 데이터베이스에 사용자명과 암호를 제공하기 위해 --username와 --password나 -P 매개변수들을 사용할 수 있다. 예를 들면 :

$ sqoop import --connect jdbc:mysql://database.example.com/employees \  
--username aaron --password 12345

|  |  |
| --- | --- |
|  | Warning |
|  | --password 매개변수는 보안에 취약하다. 다른 사용자들은 ps와 같은 명령의 결과로부터 당신의 암호를 볼 수 있기 때문이다. **-P** 인자는 콘솔 프롬프트로부터 암호를 읽을 수 있게 하므로 기밀정보를 입력하는 경우 선호되는 방식이다. 기밀정보는 보안에 취약한 수단을 사용하는 맵리듀스 클러스터의 노드들 사이에서 여전히 전송될 소지가 있다. |

Sqoop은 MySQL을 포함한 몇가지의 데이터베이스를 자동적으로 지원한다. jdbc:mysql://로 시작되는 연결 문자열은 Sqoop에서 자동적으로 처리된다. (내장 지원되는 모든 데이터베이스의 목록은 “지원 데이터베이스" 섹션에서 제공된다. 일부 데이터베이스를 위하여 직접 JDBC 드라이버를 설치할 필요가 있을 것이다.)

JDBC가 지원되는 다른 데이터베이스에 대하여 Sqoop을 사용할 수 있다. 첫 번째로, 가져오기를 원하는 데이터베이스의 형식에 맞는 JDBC 드라이버를 다운로드하고, 클라이언트 컴퓨터의 $SQOOP\_HOME/lib 디렉토리에 그 .jar 파일을 설치하라. (만약 RPM이나 Debian 패키지로부터 설치한다면 디렉토리는 /usr/lib/sqoop/lib가 된다.) 각 드라이버 .jar파일은 드라이버로의 진입점을 정의하는 특정한 드라이버 클래스를 또한 가지고 있다. 예를 들면, MySQL의 Connector/J 라이브러리는om.mysql.jdbc.Driver라는 클래스를 가진다. 주 드라이버 클래스를 결정하기 위해 당신의 데이터베이스 벤더에 특정한 문서를 참조하라. 이 클래스는 --driver 인자로 Sqoop에 제공되어야 한다.

예를 들면, SQLServer 데이터베이스에 연결하기 위하여 가장 먼저 microsoft.com에서 드라이버를 다운로드하고 Sqoop lib 경로에 그것을 설치하라.

그 다음 Sqoop을 실행하라. 예를 들면 :

$ sqoop import --driver com.microsoft.jdbc.sqlserver.SQLServerDriver \  
--connect <connect-string> ...

JDBC를 사용하여 데이베이스에 연결하고자 할 때, --connection-param-file 옵션을 사용하는 속성 파일을 통하여 추가 JDBC 매개변수들을 선택적으로 명시할 수 있다. 이 파일의 내용은 연결이 이루어지는 동안 표준 Java 속성으로 구문분석되어 드라이버로 넘겨진다.

|  |  |
| --- | --- |
|  | Note |
|  | 속성 파일에 통하여 선택적으로 명시되는 매개변수들은 JDBC 연결에만 적용이 된다. JDBC 이외의 연결을 사용하는 고속 커넥터들은 이 매개변수들을 무시할 것이다. |

**표 2. 가져오기 제어 인자:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --append | HDFS에 이미 존재하는 데이터셋에 데이터를 붙인다. |
| --as-avrodatafile | Avro 데이터 파일로 데이터를 가져온다. |
| --as-sequencefile | SequenceFile에 데이터를 가져온다. |
| --as-textfile | (기본값) 순수한 텍스트로서 데이터를 가져온다. |
| --boundary-query <문장> | 스플릿을 생성하기 위하여 사용하는 경계 쿼리 |
| --columns <컬럼,컬럼,컬럼…> | 테이블로부터 가져올 컬럼들 |
| --direct | 직접 고속 가져오기 모드를 사용한다. |
| --direct-split-size <n> | 직접 모드로 가져올 때 모든 n바이트 당 입력 스트림을 분할시킨다. |
| --inline-lob-limit <n> | 인라인 LOB를 위한 최대 사이즈를 설정한다. |
| -m,--num-mappers <n> | 병렬로 가져오기 위하여 n개의 맵 태스크를 사용한다. |
| -e,--query <쿼리문> | *쿼리문*의 결과를 가져온다. |
| --split-by <컬럼명> | 분할 단위로 쓰이는 테이블의 컬럼 |
| --table <테이블명> | 읽을 테이블 |
| --target-dir <디렉토리> | 목적지 HDFS 디렉토리 |
| --warehouse-dir <디렉토리> | 테이블 목적지를 위한 부모 HDFS 디렉토리 |
| --where <where 절> | 가져오기 동안 사용할 WHERE절 |
| -z,--compress | 압축을 사용하게 한다. |
| --compression-codec <c> | (기본은 gzip) Hadoop 코덱을 사용한다. |
| --null-string <null-string> | 문자열 컬럼에서 널 값 대신 쓰여질 문자열 |
| --null-non-string <null-string> | 문자열이 아닌 컬럼에서 널 값 대신 쓰여질 문자열 |

--null-string과 --null-non-string 인자는 선택적이다. 만약 명시되지 않으면 문자열 “null”이 사용될 것이다.

***7.2.2. 가져올 데이터 선택하기***

Sqoop은 통상 테이블 중심 방식으로 데이터를 가져온다. 가져올 테이블을 선택하기 위해 --table 인자를 사용하라. (예 : --table employees) 이 인자는 데이터베이스에서 뷰나 다른 테이블유사 엔티티를 나타낼 수 있다.

기본적으로 테이블 내의 모든 컬럼들은 가져오기를 위해 선택된다. 가져온 데이터는 “자연적인 순서”로 HDFS에 기록된다. “자연적인 순서”란 컬럼 A, B, C를 포함하는 테이블은 다음과 같은 데이터 가져오기가 된다는 것이다. :

A1,B1,C1  
A2,B2,C2  
...

--columns 인자를 사용하여 컬럼들의 부분집합을 선택하고 그것들의 순서를 제어할 수 있다. 이것은 가져올 컬럼들의 콤마로 구분된 목록을 포함해야 한다. (예 : --columns "name,employee\_id,jobtitle")

가져오기 문장에 SQL WHERE 절을 추가하여 어느 행을 가져올 것인지를 제어할 수 있다. 기본적으로 Sqoop은 SELECT <컬럼목록> FROM <테이블명>형태의 문장을 생성한다. 여기에 인자로 WHERE절을 붙일 수 있다. 예를 들면, --where "id > 400"는id 컬럼이 400보다 큰 값을 가진 행들만 가져오게 될 것이다.

기본적으로 Sqoop은 스플릿을 생성하기 위한 경계를 알아내기 위하여 select min(<split-by>), max(<split-by>) from <테이블명>쿼리를 사용할 것이다. 어떤 경우에는 이 쿼리는 최적이 아니므로 --boundary-query 인자를 사용하여 두 개의 숫자형 컬럼들을 반환하는 임의의 쿼리를 명시할 수 있다.

***7.2.3. 자유형 쿼리로 가져오기***

Sqoop은 임의의 SQL쿼리의 결과 집합을 가져올 수 있다. --table, --columns, --where 인자를 사용하는 대신에 --query 인자로 SQL문을 명시할 수 있다.

자유형 쿼리로 가져오기를 실행할 때, --target-dir 인자로 목적지 디렉토리를 명시해야만 한다.

쿼리의 결과를 병렬로 가져오고자 한다면, 각 맵 태스크는 그 쿼리의 복사본을 실행하게 될 것이며 Sqoop이 추론한 결합 조건에 의해 파티셔닝된 결과로 나타날 것이다. 쿼리는 각 Sqoop 프로세스가 고유 조건 표현으로 교체할 $CONDITIONS 토큰을 포함해야만 한다. --split-by 인자로 스플릿할 컬럼을 또한 선택해야 한다.

예를 들면:

$ sqoop import \  
--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \  
--split-by a.id --target-dir /user/foo/joinresults

다른 방법으로써 -m 1로 단일 맵을 명시함으로써 해당 쿼리는 한번 실행되어 연속적으로 가져오기가 수행되게 될 수 있다. :

$ sqoop import \  
--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \  
-m 1 --target-dir /user/foo/joinresults

|  |  |
| --- | --- |
|  | Note |
|  | 만약 쌍따옴표로 싸여진 쿼리를 적용한다면 $CONDITIONS 대신에\$CONDITIONS를 사용하여 쉘이 그것을 쉘 변수로 취급하지 않도록 해야 한다. 예를 들면, 쌍따옴표처리된 쿼리를 다음과 같이 보이게 된다.: "SELECT \* FROM x WHERE a='foo' AND \$CONDITIONS" |

|  |  |
| --- | --- |
|  | Note |
|  | 현재 버전의 Sqoop에서 자유형 쿼리를 사용하는 기능은 WHERE절 내에 애매모호한 예상과 OR조건이 없는 단순한 쿼리로 제한된다.다음과 같은 간단한 쿼리로 한정한다. 애매모호한 예상을 일으키게 하는 서브쿼리나 조인을 가진 쿠리같은 복잡한 쿼리들을 사용하면 예상하지 못한 결과를 불러 일으킬 수가 있다. |

***7.2.4. 병렬 제어하기***

Sqoop은 대부분의 데이터베이스 근원으로부터 병렬로 데이터를 가져온다. -m이나 --num-mappers 인자를 사용하여 가져오기를 수행함으로써 사용할 맵 태스크(병렬 프로세스)의 수를 명시할 수 있다. 이 인자들은 적용할 병렬의 정도에 따라 정수값을 취한다. 기본적으로 네 개의 태스크들이 사용된다. 어떤 데이터베이스들은 이 값을 8이나 16으로 증가시킴으로써 성능이 향상될 수도 있다. 맵리듀스클러스터 내에서 가용한 태스크수보다 더 큰 값으로 병렬의 정도를 증가시키지 말라. 그렇게 되면 태스크들은 연속적으로 실행되고 가져오기를 수행하는 데에 필요한 만큼의 시간을 증가시키게 될 것이다. 비슷한 원리로 데이터베이스가 합리적으로 지원할 수 있는 만큼보다 높게 병렬의 정도를 증가시키지 말라. 100 개의 동시접속 클라이언트들이 데이터베이스에 연결하게 되면 결과적으로 성능이 버티지 못하는 지점까지 데이터베이상에 부하를 증가시킬 것이다.

병렬 가져오기를 수행할 때, Sqoop은 부하를 분할할 수 있는 기준이 필요하다. Sqoop은 부하를 분할하기 위해 *분할컬럼*을 사용한다. 기본적으로 Sqoop은 (제공되는) 주 키 컬럼을 테이블에서 인식하고 그것을 분할컬럼으로서 사용한다. 분할 컬럼의 대/소 값들은 데이터베이스로부터 가져오고 맵 태스크들은 전체 범위에서 균등하게 크기가 나누어진 컴포넌트들을 작동시킨다. 예를 들면, 주 키 컬럼인 id가 최소값이 0이고 최대값이 1000인 테이블이 있을 때, Sqoop은 4개의 태스크를 정해서 4개의 프로세스가 동작되도록 한다. 각각의 프로세스는 SELECT \* FROM sometable WHERE id >= lo AND id < hi 형태의 SQL문의 (lo, hi)를 (0, 250), (250, 500), (500, 750), (750, 1001)로 다르게 설정하여 실행하게 된다.

주 키의 실제 값들이 주어진 범위안에서 일정하게 분산되어 있지 않다면 불균형한 태스크를 초래하게 된다. --split-by인자로 서로 다른 컬럼을 명시적으로 선택해주어야 한다. (예 : --split-by employee\_id) Sqoop은 현재 다중컬럼 인덱스에 대하여 분할할 수 없다. 테이블이 인덱스가 없는 컬럼을 가지고 있거나 다중컬럼 키를 가지고 있다면, 분할컬럼을 직접 선택해야만 한다.

***7.2.5. 가져오기 프로세스 제어하기***

기본적으로 가져오기 프로세스는 합리적이고 벤더중립적인 가져오기 채널을 제공하는 JDBC를 사용할 것이다. 어떤 데이터베이스들은 데이터베이스에 튺정적인 데이터 이전 도구를 사용함으로써 가져오기를 실행할 때 높은 성능 향상을 꾀할 수 있다. 예를 들면, MySQL은 mysqldump라는 도구를 제공하여 MySQL로부터 다른 시스템으로 데이터를 매우 빠르게 내보낼 수 있다. --direct 인자를 제공함으로써 Sqoop이 직접 가져오기 채널을 시도하도록 명시할 수 있다. 이 채널은 JDBC를 사용하는 것보다 더 높은 성능을 보여준다. 현재 직접 모드는 큰 오브젝트 컬럼의 가져오기를 지원하지 않는다.

직접 모드와 함게 PostgreSQL로부터 가져오기를 실행할 때, 개별 파일들이 특정 크기에 다다른 후에 분리된 파일들로 가져온 데이터를 분할할 수 있다. 이 크기 제한은 --direct-split-size 인자로 제어된다.

기본적으로 Sqoop은 당신의HDFS 홈디렉토리 안에 foo라는 디렉토리에 foo라는 테이블을 가져올 것이다. 예를 들면, 사용자명이 someuser일 때 가져오기 도구는 /user/someuser/foo/(파일들)에 쓴다. --warehouse-dir 인자로 가져오기의 부모 디렉토리를 조정할 수 있다. 예를 들면:

$ sqoop import --connnect <연결문자열> --table foo --warehouse-dir /shared \  
...

이 명령은 /shared/foo/ 디렉토리에 파일셋을 쓴다.

아래와 같이 대상 디렉토리를 명시적으로 고를 수도 있다.:

$ sqoop import --connnect <연결문자열> --table foo --target-dir /dest \  
...

이것은 /dest 디렉토리로 파일들을 가져올 것이다. --target-dir는 --warehouse-dir.와 호환된다.

직접모드를 사용할 때 놓여져 있는 도구로 넘기는 추가적인 인자들을 명시할 수 있다. 명령줄에 --인자가 주어지면 그 뒤의 인자들은 놓여져 있는 도구로 직접 보내진다. 예를 들어, 다음은 mysqldump에 의해서 사용되는 문자셋을 조정한다.:

$ sqoop import --connect jdbc:mysql://server.foo.com/db --table bar \  
--direct -- --default-character-set=latin1

기본적으로 가져온 데이터는 새로운 대상 위치로 이동한다. 만약 목적지 디렉토리가 HDFS에 이미 존재한다면 Sqoop은 가져오기를 거부하고 그 디렉토리의 내용을 덮어쓸 것이다. --append 인자를 사용하면 Sqoop은 데이터를 임시 디렉토리로 가져오고 원래 있던 파일명들과 충돌하지 않도록 하기 위해 파일명을 바꾸고 정상적인 대상 디렉토리로 이동시킨다.

|  |  |
| --- | --- |
|  | Note |
|  | 가져오기의 직접 모드를 사용할 때, 특정 데이터베이스 클라이언트 도구들은 태스크 프로세스의 쉘 경로에 있어야 한다. PostgreSQL위하여 psql 도구가 필요한 반면에 MySQL을 위하여 mysqldump와 mysqlimport 도구들이 필요하다. |

***7.2.6. 타입 매핑 제어하기***

Sqoop은 대부분의 SQL타입들을 Java 또는 Hive의 적절한 대리자에 매핑하기 위해 사전 구성된다. 그러나 기본 매핑이 모두에게 적절하지 않을 수 있으며 (Java 매핑을 변경하기 위하여) --map-column-java 또는 (Hive 매핑을 변경하기 위하여) --map-column-hive 에 의해 덮어씌워질 수도 있다.

**표 3. 매핑을 덮어쓰는 매개변수들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --map-column-java <매핑> | 설정된 컬럼들에 대하여 SQL로부터 Java 타입으로의 매핑을 덮어쓴다. |
| --map-column-hive <매핑> | 설정된 컬럼들에 대하여 SQL로부터 Hive 타입으로의 매핑을 덮어쓴다. |

<컬럼명>=<새타입>의 형태의 매핑이 콤마로 분리된 목록으로 Sqoop에 제공한다. 예를 들면 :

$ sqoop import ... --map-column-java id=String,value=Integer

Sqoop은 어떤 설정된 매핑이 사용되지 않는 경우에 예외를 일으킬 것이다.

***7.2.7. 증가분 가져오기***

Sqoop은 이전에 가져온 행 집합보다 새로운 행들을 조회해오는데 쓰이는 증가분 가져오기 모드를 제공한다.

다음 인자들은 증가분 가져오기를 제어한다.:

**표 4. 증가분 가져오기 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --check-column (컬럼명) | 어느 행들을 가져올지 결정할 때 조사되는 컬럼을 명시한다. |
| --incremental (모드) | Sqoop이 어떻게 새로운 행들을 결정하는지를 명시한다. 모드에 대한 적합한 값은 append와 lastmodified가 해당된다. |
| --last-value (값) | 이전의 가져온 데이터로부터 검사 대상 컬럼의 최대값을 명시한다. |

Sqoop은 append와 lastmodified라고 하는 두 가지 방식의 증가분 가져오기를 지원한다. 실행할 증가분 가져오기 타입을 명시하기 위해 --incremental 인자를 사용할 수 있다.

행의 아이디 값을 증가시키면서 계속적으로 추가되는 새로운 행들이 있는 테이블을 가져오려면 append 모드를 명시해야 한다. --check-column으로 그 행의 아이디를 포함하는 컬럼을 명시한다. Sqoop은 검사 컬럼이 --last-value로 명시된 값보다 큰 값이 있는 행들을 가져온다.

Sqoop이 지원하는 또 다른 테이블 갱신 전략은 lastmodified 모드라고 부른다. 근원 테이블의 행들이 갱신될 수 있으며 각 갱신 마다 마지막으로 수정된 컬럼의 값이 현재의 타임스탬프에 설정되는 경우 이 모드를 사용해야 한다. Sqoop은 --last-value로 명시된 타임스탬프보다 더 최근의 타임스탬프를 유지하고 있는 행들을 가져온다.

증가분 가져오기가 끝날 때, 차후의 가져오기를 위하여 --last-value로 명시된 값이 화면에 출력된다. 차후의 가져오기를 수행할 때 새롭거나 갱신된 데이터를 가져온다는 것을 분명히 하기 위하여 --last-value를 명시해야 할 것이다. 증가분 가져오기를 저장된 잡으로 생성하면 이러한 작업은 자동적으로 처리된다. 저장된 잡은 증가분 가져오기를 반복적으로 수행할 때 선호되는 매카니즘이다. 더 많은 정보를 위하여 이 문서에서 나중에 나오는 저장된 잡에 관한 섹션을 참고하라.

***7.2.8. 파일 포맷***

구분된 텍스트 또는 SequenceFile, 두 가지의 파일 포맷 중 하나로 데이터를 가져올 수 있다.

구분된 텍스트는 기본 가져오기 포맷이다. --as-textfile 인자를 사용하여 파일 포맷을 명시할 수도 있다. 이 인자는 개별적인 컬럼과 행들 사이에 구분자를 두어서 각 레코드의 문자열 기반 표현을 출력 파일에 기록할 것이다. 이 구분자는 콤마, 탭, 또는 다른 문자가 될 수 있다. (구분자는 선택할 수 있으며 “출력줄 포맷팅 인자들”를 참고하라) 다음은 텍스트 기반으로 가져온 예시의 결과이다.:

1,here is a message,2010-05-01  
2,happy new year!,2010-01-01  
3,another message,2009-11-12

구분된 텍스트는 대부분의 이진 형식이 아닌 데이터 타입에 적절하다. Hive와 같은 다른 도구에 의해서 처리되는 방법을 지원 중이기도 하다.

SequenceFile은 커스텀 레코드에 특정된 데이터 타입으로 개별 레코드들 저장하는 이진 포맷이다. 이러한 데이터 타입은 자바 클래스로 나타내어진다. Sqoop은 당신을 위하여 이러한 데이터를 자동적으로 생생할 것이다. 이 포맷은 이진 표현으로 모든 데이터의 정확한 저장을 지원하고 이진 데이터(예 : VARBINARY 컬럼) 또는 커스텀 맵리듀스 프로그램에 의해서 처리되는 데이터의 저장에 적합하다. (SequenceFile으로부터 읽는 것은 레코드가 구문분석될 필요가 없기 때문에 텍스트 파일로부터 읽는 것보다 더 나은 성능을 보여준다.

Avro 데이터 파일을 다른 프로그래밍 언어로 작성된 어플리케이션과의 호환성을 제공하는 작고 효율적인 이진 포맷이다. Avro는 버전관리를 제공하기 때문에 한 예로 컬럼들이 테이블에 추가되거나 제거될 때 이전에 가져온 데이터 파일들을 새 데이터와 함께 처리될 수 있다.

기본적으로 데이터는 압축되지 않는다. -z나 --compress 인자로 deflate (gzip) 알고리즘을 사용하여 데이터를 압축할 수 있다. 또는 --compression-codec 인자를 사용하여 Hadoop 압축 코덱을 명시할 수 있다. 이것은 SequenceFile, 텍스트, Avro파일에 적용된다.

***7.2.9. 커다란 오브젝트 (Large Objects)***

Sqoop은 특별한 방식으로 커다란 오브젝트(BLOB과 CLOB 컬럼)를 다룬다. 이 데이터가 정말 크다면, 대부분의 컬럼들이 그러하듯이 이 컬럼들을 처리하기 위해 메모리에 나타낼 수 없다. 그 대신에 데이터를 스트리밍 방식으로 다룬다. 커다란 오브젝트는 다음의 경우에 대하여 데이터의 나머지와 함께 인라인으로 저장될 수 있다. 커다란 오브젝트가 모든 접근에 대하여 메모리에 완전히 실현되거나 주 데이터 스토리지와 연결된 보조 스토리지 파일에 저장될 수 있는 경우이다. 기본적으로 크기가 16MB보다 작은 커다란 오브젝트는 그 데이터의 나머지와 함께 인라인으로 저장된다. 좀 더 큰 크기일 때에는 가져오기 대상 디렉토리의 \_lobs라는 서브디렉토리에 있는 파일로 저장된다. 이 파일들은 레코드 크기가 2의 63승 바이트인 레코드들을 수용할 수 있는 스토리지를 위하여 최적화된 별도의 포맷으로 저장된다. 커다란 오브젝트들이 분리된 파일들로 쓰여지는 임계 크기는 --inline-lob-limit 인자로 제어된다. 이 인자는 바이트 단위로 인라인으로 유지될 수 있는 가장 커다란 오브젝트 사이즈를 명시하는 매개변수를 취한다. 만약 인라인 LBO 한계를 0으로 설정하면, 모든 커다란 오브젝트들은 외부 스토리지에 위치하게 될 것이다.

**표 5. 출력줄 포맷팅 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --enclosed-by <문자> | 필요한 필드를 묶는(enclose) 문자를 설정한다. |
| --escaped-by <문자> | 회피(escaping) 문자를 설정한다. |
| --fields-terminated-by <문자> | 필드구분자 문자를 설정한다. |
| --lines-terminated-by <문자> | 줄종료 문자를 설정한다. |
| --mysql-delimiters | MySQL의 기본 구분자 집합을 사용한다. (필드:, 줄: \n escaped-by: \ optionally-enclosed-by: ') |
| --optionally-enclosed-by <문자> | 필드 묶음(enclosing) 문자를 설정한다. |

구분처리된 파일들을 가져올 때, 구분자의 선택은 중요하다. 문자열 기반의 필드 내에 나오는 구분자들은 차후의 분석을 통과시킴으로써 가져온 데이터에 대한 애매모호한 구문분석을 일으킬 수 있다. 예를 들면, 문자열 "Hello, pleased to meet you"은 콤마로 설정된 필드종료 구분자를 사용하여 가져오기를 수행하지 않아야 한다.

구분자들은 다음과 같이 명시될 수 있다. :

* 문자 (--fields-terminated-by X)
* 이스케이프 문자 (--fields-terminated-by \t). 지원 이스케이프 문자들:
* \b (백스페이스)
* \n (개행)
* \r (캐리지리턴)
* \t (탭)
* \" (쌍따옴표)
* \\' (홀따옴표)
* \\ (백슬래시)
* \0 (NUL) - 이것은 필드나 줄 사이에 문자를 삽입할 것이다. --enclosed-by, --optionally-enclosed-by, 또는 --escaped-by 인자 중 하나를 위하여 사용되는 경우, 묶기(enclosing)와 회피하기(escaping)를 사용하지 못하게 할 것이다.
* UTF-8 문자의 코드포인트의 8진수 표현. 이것은 \0ooo 형식이며 *ooo*는 8진수 값이다. 예를 들면, --fields-terminated-by \001 는 ^A 문자를 말한다.
* UTF-8 문자의 코드포인트의 16진수 표현. 이것은 \0xhhh 형식이며 *hhh*는 16진수 값이다. 예를 들면, --fields-terminated-by \0x10 는 캐리지리턴 문자를 말한다.

기본적으로 구분자들은 필드에 대해서 콤마(,), 레코드에 대해서 개행(\n), 따옴표문자 제외, 이스케이프 문자 제외이다. 필드 데이터에 콤마나 개행문자를 포함하는 데이터베이스 레코드를 가져오는 경우 애매모호하고 구문분석할 수 없는 레코드가 될 수 있음을 주의하라. 애매모호하지 않은 구문분석을 위하여 따옴표문자와 이스케이프 문자는 사용 가능하게 되어야 한다. 예를 들면, --mysql-delimiters인자를 사용해서 말이다.

애매모호하지 않은 구분자들이 제공될 수 없다면, *묶음(enclosing) 문자*와 *회피(escaping) 문자*들을 사용하라. (선택적인) 묶음 문자와 회피 문자의 조합은 줄의 애매모호하지 않은 구문분석을 가능하게 할 것이다. 예를 들면, 다음의 값들을 포함하는 어떤 데이터셋의 하나의 컬럼을 가정해보자:

Some string, with a comma.  
Another "string with quotes"

다음의 인자들은 애매모호하지 않게 구문분석이 될 수 있는 구분자들을 제공할 것이다. :

$ sqoop import --fields-terminated-by , --escaped-by \\ --enclosed-by '\"' ...

(쉘이 묶음문자를 토막내지 않도록 하기 위해 홀따옴표로 인자를 묶었다는 것에 주목하라.)

위의 인자들에 의한 결과는 데이터셋에 다음과 같이 적용된다.:

"Some string, with a comma.","1","2","3"...  
"Another \"string with quotes\"","4","5","6"...

묶기와 회피하기의 전체 효과를 설명하기 위하여 추가적인 컬럼들("1","2","3" 등)이 있는 가져온 문자열이 위에 있다. 묶음문자는 구분자 문자가 가져온 텍스트 내에 나타날 때에만 반드시 필요하다. 그러므로 묶음문자는 선택적으로 명시될 수 있다.:

$ sqoop import --optionally-enclosed-by '\"' (위에 명시된 명령과 나머지 동일)...

위의 명령은 다음과 같은 가져오기 결과를 생성한다.:

"Some string, with a comma.",1,2,3...  
"Another \"string with quotes\"",4,5,6...

|  |  |
| --- | --- |
|  | Note |
|  | Hive가 회피 문자를 지원하더라도 개행 문자의 회피를 지원하지 않는다. 또한 묶여진 문자열 내에 필드 구분자를 포함하는 묶음문자에 대한 개념을 지원하지 않는다. 그러므로 Hive를 연동할 때 회피 문자와 묶음 문자의 도움 없이 애매모호하지 않은 필드와 레코드종료 구분자를 선택하기를 권한다. 이것은 Hive의 입력 구문분석 능력의 제한 때문이다. |

--mysql-delimiters 인자는 mysqldump 프로그램을 위한 기본 구분자를 사용하는 간편한 인자이다. (--direct를 사용하여) 직접모드 가져오기와 함께 mysqldump 구분자를 사용한다면, 매우 빠른 속도로 가져오기를 수행할 수 있다.

구분자의 선택이 텍스트모드 가져오기를 위해 가장 중요하듯이--as-sequencefile와 함께 SequenceFile로 가져오기를 하는 경우에도 여전히 관련성이 있다. 생성된 클래스의 toString() 메서드는 당신이 명시한 구분자들을 사용할 것이고, 차후의 출력 데이터의 포맷팅도 당신이 선택한 구분자를 따를 것이다.

**표 6. 입력 구문분석 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --input-enclosed-by <문자> | 필요한 필드를 묶는 문자를 설정한다. |
| --input-escaped-by <문자> | 입력의 회피문자를 설정한다. |
| --input-fields-terminated-by <문자> | 입력의 필드 구분자를 설정한다. |
| --input-lines-terminated-by <문자> | 입력의 줄종료 문자를 설정한다. |
| --input-optionally-enclosed-by <문자> | 필드 묶음 문자를 설정한다. |

Sqoop이 HDFS로 데이터를 가져올 때, Sqoop은 구분자로 포맷하는 가져오기를 수행하는 동안 생성하는 텍스트 파일을 재해석할 수 있는 Java 클래스를 생성한다. 구분자는 -fields-terminated-by와 같은 인자로 선택된다. 이 인자는 데이터가 디스크에 어떻게 쓰일 지와 생성된 parse() 메서드가 이 데이터를 어떻게 해석할 지를 제어한다. 메서드가 사용하는 구분자는 -input-fields-terminated-by 등의 인자를 사용함으로써 출력 인자와는 별도로 지정될 수 있다. 예를 들면, 구분자 집합으로 생성된 레코드를 구문분석하고 별도의 구분자 집합을 사용하여 서로 다른 파일 집합을 만들기 위하여 레코드들을 내어놓을 수 있는 클래스들을 생성할 때 유용하다.

**표 7. Hive 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --hive-home <디렉토리> | $HIVE\_HOME를 덮어쓴다. |
| --hive-import | 테이블을 Hive로 가져온다. (아무것도 설정되지 않으면, Hive의 기본 구분자를 사용한다.) |
| --hive-overwrite | Hive 테이블에 존재하는 데이터를 덮어쓴다. |
| --create-hive-table | 대상 Hive 테이블이 존재할 때 잡이 실패하도록 한다. 기본적으로 이 프로퍼티는 false이다. |
| --hive-table <테이블명> | Hive로 가져오기를 수행할 때 사용할 테이블명을 설정한다. |
| --hive-drop-import-delims | Hive로 가져오기를 수행할 때 문자열에 있는 *\n*, *\r*, *\01*를 없앤다. |
| --hive-delims-replacement | Hive로 가져오기를 수행할 때 문자열에 있는 *\n*, *\r*, *\01*를 사용자가 정의한 문자열로 대체한다. |
| --hive-partition-key | 파티션을 나눌 때 사용할 Hive 필드의 이름 |
| --hive-partition-value <값> | 해당 잡에서 Hive로 가져올 때 파티션 키로 사용되는 문자열 값. |
| --map-column-hive <매핑명> | 설정된 컬럼들에 대하여 SQL타입에서 Hive타입으로 매핑하는 기본값을 덮어쓴다. |

***7.2.10. 데이터를 하이브로 가져오기***

Sqoop의 가져오기 도구의 주 기능은 당신의 데이터를 HDFS 내의 파일로 업로드하는 것이다. 만약 당신이 HDFS클러스터와 관계가 있는 Hive 메타스토어를 가지고 있다면, Sqoop은 Hive 내의 데이터 레이아웃을 정의하기 위하여 CREATE TABLE 문을 생성하고 실행함으로써 Hive로 그 데이터를 가져올 수도 있다. Hive로 데이터를 가져오는 것은 Sqoop 명령줄에 **--hive-import** 옵션을 추가하면 될 정도로 단순하다.

Hive 테이블이 이미 존재한다면, 그 테이블이 대체되어야 함을 알리기 위해서 **--hive-overwrite** 옵션을 명시할 수 있다. 당신의 데이터가 HDFS로 가져오기 되거나 이 과정이 생략된 후, Sqoop은 Hive 타입으로 컬럼을 정의하는 CREATE TABLE 연산과 Hive의 웨어하우스 디렉토리로 데이터 파일을 이동시키는 LOAD DATA INPATH문을 포함하는 Hive 스크립트를 생성할 것이다.

이 스크립트는 Sqoop이 실행되는 서버 상의 Hive 설치본을 호출하여 실행될 것이다. Hive 설치본이 여러개 있거나 hive가 $PATH에 없다면, Hive 설치 디렉토리를 지정하는 **--hive-home**옵션을 사용하라. Sqoop은 이로부터 $HIVE\_HOME/bin/hive을 사용할 것이다.

|  |  |
| --- | --- |
|  | Note |
|  | 이 기능은 --as-avrodatafile, --as-sequencefile과 함께 사용할 수 없다. |

Hive가 회피 문자를 지원하더라도 개행 문자의 회피를 처리하지는 않는다. 또한 묶여진 문자열 내의 필드 구분자를 포함할 수 있는 묶음 문자의 개념을 지원하지 않는다. 그러므로 Hive와 연동할 때 회피 문자와 묶음 문자를 사용하지 않고 애매모호하지 않는 필드와 레코드 종료 구분자를 선택하기를 권한다. 이것은 Hive의 구문분석 능력의 제한 때문이다. Hive로 데이터를 가져올 때 --escaped-by, --enclosed-by, 또는 --optionally-enclosed-by를 사용하면, Sqoop은 주의 메시지를 출력할 것이다.

Hive의 기본 행 구분자(\n, \r 문자) 또는 컬럼 구분자(\01 문자)가 있는 문자열 필드를 데이터베이스의 행이 포함한다면, Hive는 Sqoop을 이용한 가져오기에 문제를 일으킬 것이다. Hive와 호환되는 텍스트 데이터를 제공하기 위하여 가져오기를 수행할 때 그 문자들을 없앨 수 있는 --hive-drop-import-delims 옵션을 사용할 수 있다. 또는 --hive-delims-replacement 옵션을 사용하여 그 문자들을 사용자가 정의한 문자열을 대체할 수 있다. 이 옵션들은 Hive의 기본 구분자를 사용할 때만 사용되어야 하며 다른 구분자가 명시되었을 때에는 사용할 수 없다.

Sqop은 Hive를 통하여 필드와 레코드 구분자를 넘길 것이다. 어떤 구분자도 설정하지 않고 --hive-import를 사용한다면, Hive의 기본설정과 일치시키기 위하여 필드 구분자는 ^A로 설정될 것이며 레코드 구분자는 \n로 설정될 것이다.

Hive에서 사용되는 테이블명은 기본적으로 근원 테이블명과 같다. --hive-table옵션을 사용하여 출력 테이블의 이름을 제어할 수 있다.

Hive는 보다 효율적인 쿼리 성능을 위하여 데이터를 파티션으로 나눌 수 있다. --hive-partition-key와 --hive-partition-value 인자를 명시함으로써 Sqoop 잡이 데이터를 Hive를 위하여 특정한 파티션으로 가져오기를 실행하도록 할 수 있다. 파티션 값은 어떤 문자열이 되어야 한다. 파티셔닝에 대하여 더 자세한 사항은 Hive 문서를 참조하라.

--compress와 --compression-codec옵션을 사용하면 압축된 테이블을 Hive로 가져올 수 있다. Hive로 가져온 압축 테이블에 대한 한 가지 단점은 병렬 맵 태스크의 처리를 위하여 많은 코덱들이 나누기(splitting)을 지원할 수 없다는 것이다. 그러나 lzop 코덱은 나누기를 지원한다. 이 코덱응로 테이블을 가져올 때, Sqoop은 올바른 InputFormat으로 Hive 테이블을 나누고 설정하기 위하여 자동적으로 파일들을 색인할 것이다. 현재 이 기능은 테이블의 모든 파티션들이 lzop 코덱으로 압축되었을 때 가능하다.

**표 8. HBase 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --column-family <군> | 가져오기를 위한 대상 컬럼 군을 설정한다. |
| --hbase-create-table | 이 인자가 명시되면 누락된 HBase 테이블을 생성한다. |
| --hbase-row-key <컬럼명> | 어떤 입력 컬럼을 행으 키로 사용할 지를 명시한다. |
| --hbase-table <테이블명> | HDFS 대신에 대상으로 사용할 HBase 테이블을 명시한다. |

***7.2.11. 데이터를 HBase로 가져오기***

Sqoop은 HDFS와 Hive 이외의 또 다른 가져오기 대상을 지원한다. Sqoop은 레코드를 HBase의 테이블로 가져올 수도 있다.

--hbase-table을 명시함으로써 Sqoop이 HDFS의 디렉토리가 아닌 HBase의 테이블로 가져오기를 실행하도록 한다. Sqoop은 --hbase-table인자에 명시한 테이블로 데이터를 가져올 것이다. 입력 테이블의 각 행은 출력 테이블의 행에 대한 HBase의 Put 연산으로 변형될 것이다. 각 행의 키는 입력의 컬럼으로부터 얻는다. 기본적으로 Sqoop은 행의 키 컬럼으로서 분산된(split-by) 컬럼을 사용할 것이다. 그것이 명시되지 않는다면, 근원 테이블의 주 키 컬럼을 알아내려고 할 것이다. --hbase-row-key로 행의 키 컬럼을 직접 명시할 수 있다. 각 출력 컬럼은 --column-family으로 명시되었을 때 동일한 컬럼 군에 위치하게 된다.

대상 테이블과 컬럼 군이 존재하지 않는다면, Sqoop 잡은 에러와 함께 종료된다. 가져오기를 실행하기 전에 대상 테이블과 컬럼 군을 미리 생성해 놓아야 한다. --hbase-create-table을 명시한다면, Sqoop은 HBase 설정의 기본 매개변수를 이용하여 대상 테이블과 컬럼 군이 없을 때 그것들을 생성할 것이다.

현재 Sqoop은 (텍스트 모드로 HDFS에 가져오는 것처럼) 각 필드를 문자열 표현으로 바꾸어 모든 값들을 HBase로 직렬화하고 대상 셀에 그 문자열을 UTF-8 바이트로 삽입한다.

**표 9. 코드 생성 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --bindir <디렉토리> | 컴파일된 오브젝트를 위한 출력 디렉토리 |
| --class-name <클래스명> | 생성된 클래스명을 설정한다. 이것은 --package-name을 덮어쓴다. --jar-file와 결합되면 입력 클래스를 설정한다. |
| --jar-file <파일명> | 코드 생성을 비활성화하고 명시된 jar를 사용한다. |
| --outdir <디렉토리명> | 생성된 코드를 위한 출력 디렉토리 |
| --package-name <패키지명> | 지정한 패키지에 자동생성된 클래스들을 넣는다. |
| --map-column-java <매핑명> | 설정된 컬럼에 대하여 SQL타입에서 Java타입으로 매핑하는 기본값을 덮어쓴다. |

전에 언급한 것처럼, 테이블을 HDFS로 가져올 때 만들어지는 부산물은 가져온 데이터를 처리할 수 있는 클래스이다. 그 데이터가 SequenceFile로 저장된다면,이 클래스는 데이터의 직렬화 컨테이너를 위해 사용될 것이다. 그러므로, 당신은 차후의 맵리듀스 데이터 처리에서 이 클래스를 사용해야 한다.

클래스는 테이블명을 따라 통상 이름이 지어진다. foo라는 이름의 테이블은 foo라는 이름의 클래스를 생성한다. 이 클래스명을 덮어쓰고 싶을 수 있다. 예를 들면, 테이블명이 EMPLOYEES이라면 대신에 --class-name Employee 으로 명시하고 싶을 수 있다. 비슷한 원리로 --package-name으로 패키지명을 명시할 수 있다. 다음의 가져오기는 com.foocorp.SomeTable라고 하는 클래스를 생성한다.:

$ sqoop import --connect <커넥트문자열> --table SomeTable --package-name com.foocorp

클래스를 위한 .java 소스 파일은 sqoop을 실행하는 현재의 작업 디렉토리에 써질 것이다. --outdir으로 출력 디렉토리를 제어할 수 있다. 예를 들면, --outdir src/generated/.

가져오기 프로세스는 소스 파일을 .class와 .jar 파일로 컴파일하며 이 파일들은 보통 /tmp 아래에 저장된다. --bindir으로 다른 대상 디렉토리를 선택할 수 있다. 예를 들면, --bindir /scratch.

가져오기를 수행할 때 사용할 컴파일된 클래스를 이미 가지고 있으며 가져오기 프로세스의 코드 생성 기능을 억제하려고 한다면, -jar-file, --class-name옵션들을 이용하여 기존의 jar와 class를 사용할 수 있다. 예를 들면:

$ sqoop import --table SomeTable --jar-file mydatatypes.jar \  
--class-name SomeTableType

이 명령은 mydatatypes.jar 로부터 SomeTableType 클래스를 로딩할 것이다.

**7.3. 예제 실시**

다음의 예제는 다양한 상황에서 가져오기 도구를 사용하는 방법을 보여준다.

corp 데이터 베이스의 EMPLOYEES 테이블에 대한 기본 가져오기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES

로그인을 요구하는 기본 가져오기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--username SomeUser -P  
Enter password: (hidden)

EMPLOYEES 테이블에서 특정 컬럼들을 선택하기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--columns "employee\_id,first\_name,last\_name,job\_title"

가져오기 동시성 제어하기 (8개의 병렬 태스크 사용하기):

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
-m 8

MySQL의 직접 고속 가져오기 모드 활성화하기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--direct

SequenceFile로 저장하고, 생성된 클래스명을 com.foocorp.Employee으로 설정하기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--class-name com.foocorp.Employee --as-sequencefile

텍스트 모드 가져오기에서 사용할 구분자 명시하기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--fields-terminated-by '\t' --lines-terminated-by '\n' \  
--optionally-enclosed-by '\"'

데이터를 Hive로 가져오기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--hive-import

신규 직원들만 가져오기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--where "start\_date > '2010-01-01'"

기본설정으로부터 분할 컬럼을 변경하기:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \  
--split-by dept\_id

가져오기가 성공했는지 검증하기:

$ hadoop fs -ls EMPLOYEES  
Found 5 items  
drwxr-xr-x - someuser somegrp 0 2010-04-27 16:40 /user/someuser/EMPLOYEES/\_logs  
-rw-r--r-- 1 someuser somegrp 2913511 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00000  
-rw-r--r-- 1 someuser somegrp 1683938 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00001  
-rw-r--r-- 1 someuser somegrp 7245839 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00002  
-rw-r--r-- 1 someuser somegrp 7842523 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00003  
  
$ hadoop fs -cat EMPLOYEES/part-m-00000 | head -n 10  
0,joe,smith,engineering  
1,jane,doe,marketing  
...

이미 테이블의 10만행을 가져온 상태에서 새로운 데이터에 대한 증가분 가져오기 수행하기:

$ sqoop import --connect jdbc:mysql://db.foo.com/somedb --table sometable \  
--where "id > 100000" --target-dir /incremental\_dataset --append

**8. sqoop-import-all-tables**

[8.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_2)

[8.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_2)

[8.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_2)

**8.1. 목적**

import-all-tables 도구는 테이블 집합을 RDBMS로부터 HDFS로 가져온다. 각 테이블의 데이터는 HDFS의 분리된 디렉토리에 저장된다.

import-all-tables 도구를 잘 사용하려면, 다음의 조건이 충족되어야 한다.:

* 각 테이블은 단일 컬럼의 주 키를 가져야 한다.
* 각 테이블의 모든 컬럼들을 가져오는 것이어야 한다.
* 기본으로 설정되지 않은 분할 컬럼을 사용해서는 안되며 WHERE절을 통하여 어떤 조건을 부여해서도 안된다.

**8.2. 문법**

$ sqoop import-all-tables (포괄적 인자) (가져오기 인자)  
$ sqoop-import-all-tables (포괄적 인자) (가져오기 인자)

Hadoop의 포괄적 인자는 가져오기 인자보다 우선해야 하지만, 가져오기 인자들은 서로 어떠한 순서로 입력되어도 좋다.

표 **10. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 커넥트 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버를 직접 명시한다. |
| --hadoop-home <디렉토리> | $HADOOP\_HOME을 명시한다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔로부터 비밀번호를 읽는다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 작동 중 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공할 선택적인 프로퍼티 파일 |

**표 11. 가져오기 제어 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --as-avrodatafile | Avro 데이터 파일로 데이터를 가져온다. |
| --as-sequencefile | 데이터를 SequenceFile로 가져온다. |
| --as-textfile | 데이터를 순수 텍스트로 가져온다. (기본) |
| --direct | 직접 고속 가져오기를 사용한다. |
| --direct-split-size <n> | 직접 모드로 가져올 때 매 n 바이트마다 입력 스트림을 나눈다. |
| --inline-lob-limit <n> | 인라인 LOB에 대하여 최대 크기를 설정한다. |
| -m,--num-mappers <n> | 병렬로 가져오기 위하여 n 개의 맵 태스크들을 사용한다. |
| --warehouse-dir <디렉토리> | 테이블 목적지의 HDFS 디렉토리의 부모 디렉토리 |
| -z,--compress | 압축을 활성화한다. |
| --compression-codec <c> | Hadoop 코덱을 사용한다. (기본 - gzip) |

이 인자들은 sqoop-import 도구에서 와 같은 방식으로 작동한다. 그러나 --table, --split-by, --columns, --where 인자들은 sqoop-import-all-tables 에서 사용할 수 없다.

**표 12. 출력줄 포맷팅 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --enclosed-by <문자> | 필요한 필드를 묶는(enclosing) 문자를 설정한다. |
| --escaped-by <문자> | 회피(escape) 문자를 설정한다. |
| --fields-terminated-by <문자> | 필드 분리 문자를 설정한다. |
| --lines-terminated-by <문자> | 줄종료 문자를 설정한다. |
| --mysql-delimiters | MySQL의 기본 구분자 집합을 사용한다. (필드: , 줄: \n escaped-by: \ optionally-enclosed-by: ') |
| --optionally-enclosed-by <문자> | 필드 묶음 문자를 설정한다. |

**표 13. 입력 구문분석 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --input-enclosed-by <문자> | 필요한 필드를 묶는 문자를 설정한다. |
| --input-escaped-by <문자> | 입력의 회피문자를 설정한다. |
| --input-fields-terminated-by <문자> | 입력의 필드 분리자를 설정한다. |
| --input-lines-terminated-by <문자> | 입력의 줄종료 문자를 설정한다. |
| --input-optionally-enclosed-by <문자> | 필드 묶음 문자를 설정한다. |

**표 14. Hive 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --hive-home <디렉토리> | $HIVE\_HOME를 덮어쓴다. |
| --hive-import | 테이블을 Hive로 가져온다. (아무것도 설정되지 않으면, Hive의 기본 구분자를 사용한다.) |
| --hive-overwrite | Hive 테이블에 존재하는 데이터를 덮어쓴다. |
| --create-hive-table | 대상 Hive 테이블이 존재할 때 잡이 실패하도록 한다. 기본적으로 이 프로퍼티는 false이다. |
| --hive-table <테이블명> | Hive로 가져오기를 수행할 때 사용할 테이블명을 설정한다. |
| --hive-drop-import-delims | Hive로 가져오기를 수행할 때 문자열에 있는 \n, \r, \01를 없앤다. |
| --hive-delims-replacement | Hive로 가져오기를 수행할 때 문자열에 있는 \n, \r, \01를 사용자가 정의한 문자열로 대체한다. |
| --hive-partition-key | 파티션을 나눌 때 사용할 Hive 필드의 이름 |
| --hive-partition-value <값> | 해당 잡에서 Hive로 가져올 때 파티션 키로 사용되는 문자열 값. |
| --map-column-hive <매핑명> | 설정된 컬럼들에 대하여 SQL타입에서 Hive타입으로 매핑하는 기본값을 덮어쓴다. |

**표 15. 코드 생성 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --bindir <디렉토리> | 컴파일된 오브젝트를 위한 출력 디렉토리 |
| --jar-file <파일명> | 코드 생성을 비활성화하고 명시된 jar를 사용한다. |
| --outdir <디렉토리> | 생성된 코드를 위한 출력 디렉토리 |
| --package-name <패키지명> | 지정한 패키지에 자동생성된 클래스들을 넣는다. |

import-all-tables 도구는 --class-name 인자를 지원하지 않는다. 그러나 모든 생성 클래스들이 위치할 패키지명을 --package-name으로 명시할 수 있다.

**8.3. 예제 실시**

corp 데이터베이스로부터 모든 테이블을 가져오기:

$ sqoop import-all-tables --connect jdbc:mysql://db.foo.com/corp

작동여부를 검증하기:

$ hadoop fs -ls  
Found 4 items  
drwxr-xr-x - someuser somegrp 0 2010-04-27 17:15 /user/someuser/EMPLOYEES  
drwxr-xr-x - someuser somegrp 0 2010-04-27 17:15 /user/someuser/PAYCHECKS  
drwxr-xr-x - someuser somegrp 0 2010-04-27 17:15 /user/someuser/DEPARTMENTS  
drwxr-xr-x - someuser somegrp 0 2010-04-27 17:15 /user/someuser/OFFICE\_SUPPLIES

**9. sqoop-export**

[9.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_3)

[9.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_3)

[9.3. 삽입 대 갱신](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_inserts_vs_updates)

[9.4. 내보내기와 트랙잭션](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_exports_and_transactions)

[9.5. 실패한 내보내기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_failed_exports)

[9.6. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_3)

**9.1. 목적**

export 도구는 파일 셋을 HDFS로부터 RDBMS로 내보낸다. 대상 테이블은 데이터베이스에 이미 존재해아 한다. 입력 파일들은 사용자 지정 구분자에 따라 레코드셋으로 읽어들여지고 구문분석 된다.

기본 동작은 레코드들을 데이터베이스로 주입시키는 INSERT 문의 집합으로 변형하는 것이다. “갱신 모드"에서 Sqoop은 데이터베이스에 존재하는 레코드들을 대체시키는 INSERT 문을 생성할 것이다.

**9.2. 문법**

$ sqoop export (포괄적 인자들) (export 인자들)  
$ sqoop-export (포괄적 인자들) (export 인자들)

Hadoop의 포괄적 인자들이 export 인자들보다 앞에 있어야 하지만 export 인자들끼리는 순서에 상관 없이 입력할 수 있다.

**표 16. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <dir> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

**표 17. 내보내기 제어 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --direct | 직접 고속 내보내기 모드를 사용한다. |
| --export-dir <디렉토리경로> | 내보내기를 위한 HDFS 소스 경로 |
| -m,--num-mappers <개수> | 병렬로 내보내기하기 위한 *n*개의 맵 태스크를 사용한다. |
| --table <테이블명> | 데이터를 채울 테이블명 |
| --update-key <컬럼명> | 갱신을 위하여 사용하는 고정(anchor) 컬럼. 하나 이상의 컬럼이 있을 경우, 콤마로 분리되는 컬럼 목록을 사용한다. |
| --update-mode <mode> | 데이터베이스에서 새 행들이 매칭되지 않는 키로 발견이 되었을 때 어떻게 갱신이 수행되는지를 명시한다.  mode에 해당하는 값은 updateonly(기본)과 allowinsert이다. |
| --input-null-string <널문자열> | 문자열 컬럼에 대하여 널로 해석될 문자열 |
| --input-null-non-string <널문자열> | 문자열이 아닌 컬럼에 대하여 널로 해석될 문자열 |
| --staging-table <스테이징 테이블명> | 목적 테이블에 삽입되기 전에 데이터가 스테이징될 테이블 |
| --clear-staging-table | 스테이징 테이블에 있는 어떤 데이터도 삭제될 수 있는지를 나타낸다. |
| --batch | 내부적으로 문장을 실행할 때 배치 모드를 사용한다. |

--table과 --export-dir 인자들이 필요하다. 이 인자들은 데이터베이스에 데이터를 채울 테이블과 근원 데이터를 포함하는 HDFS의 디렉토리를 명시한다.

디렉토리에 있는 파일의 수와 상관 없이 매퍼의 수를 제어할 수 있다. 내보내기 성능은 병렬(parallelism)의 정도에 달려있다. 기본적으로 Sqoop은 내보내기 프로세스를 위하여 병렬적으로 네 개의 태스크를 사용할 수 있다. 이것은 최적치가 아니며 당신이 정한 특정한 설정으로 실험해볼 필요가 있다. 추가적인 태스크는 동시적인 실행을 더 좋게 할 수 있다. 하지만 데이터베이스가 인덱스를 갱신하고 트리거를 동작시키는 등의 작업을 할 때 이미 병목현상이 발생하고 있다면 추가적인 부하는 성능을 떨어뜨릴 수 있다. --num-mappers 또는 -m 인자들은 사용되는 병렬의 정도에 해당하는 맵 태스크의 수를 제어한다.

MySQL은 mysqlimport 도구를 사용하여 내보내기를 위한 직접 모드를 역시 제공한다. MySQL로 내보내기를 할 때, 코드경로를 명시하기 위하여 --direct 인자를 사용하라. 이것은 표준 JDBC 코드경로보다 더 나은 성능을 보일 것이다.

|  |  |
| --- | --- |
|  | Note |
|  | MySQL에 대하여 직접 모드로 내보내기를 사용할 때, MySQL 대량 도구인mysqlimport가 작업 프로세스의 쉘 경로에 있어야만 한다. |

--input-null-string와 --input-null-non-string 인자들은 선택적이다. 만약 --input-null-string이 명시되지 않으면, 문자열 "null"은 문자열 형식의 컬럼에 대하여 널로 해석될 것이다. 만약 --input-null-non-string이 명시되지 않으면, 문자열 "null"과 빈 문자열은 문자열이 아닌 컬럼에 대하여 널로 해석될 것이다. -input-null-non-string이 명시된다면 다른 문자열과 더불어 빈 문자열은 문자열이 아닌 컬럼에 대하여 언제나 널로 해석된다.

Sqoop이 내보내기 프로세스를 다수의 트랜잭션으로 분해하기 때문에 내보내기 잡이 실패하여 데이터베이스에 부분적으로 커밋이 되는 경우가 발생할 수 있다. 이것 때문에 어떤 경우에는 삽입 충돌 때문에 차후의 잡이 실패하게 되거나 다른 경우에는 중복 데이트 문제를 일으키게 될 수 있다. 이 문제를 --staging-table 옵션을 통하여 스테이징 테이블을 명시함으로써 극복할 수 있는데, 이 옵션은 내보내어진 데이터를 스테이징하는 데에 사용되는 예비 테이블로 작동한다. 스테이징된 데이터는 최종적으로 단일 트랜잭션 내에서 목적 테이블로 이동한다.

스테이징 기능을 사용하기 위하여 내보내기 잡을 실행하기 전에 스테이징 테이블을 생성해야만 한다. 이 테이블은 대상 테이블과 구조적으로 동일해야 한다. 이 테이블은 내보내기 잡이 실행되기 전에 비어있어야 하거나 --clear-staging-table 옵션이 명시되어야 한다. 만약 스테이징 테이블이 데이터를 포함하고 있고 --clear-staging-table 옵션이 명시된다면, Sqoop은 내보내기 잡을 시작하기 전에 모든 데이터를 삭제할 것이다.

|  |  |
| --- | --- |
|  | Note |
|  | 데이터를 목적 테이블로 밀어넣기 전에 스테이징하는 지원 기능은 --direct 내보내기에서는 사용할 수 없다. 내보내기가 기존 데이터에 대하여 --update-key 옵션을 사용하여 동작되는 경우에도 사용할 수 없다. |

**9.3. 삽입 대 갱신**

기본적으로 sqoop-export는 어떤 테이블에 새로운 행을 추가한다. 각 입력 레코드는 INSERT 문으로 변형되어 대상 데이터베이스 테이블에 행을 추가한다. 만약 테이블이 제약성(예: 주 키 컬럼은 고유의 값을 가지는 것)을 가지고 있고 이미 데이터를 포함하고 있으면 이 제약성을 위반하는 레코드를 삽입하지 않도록 주의해야 한다. INSERT 문이 실패하면, 내보내기 프로세스는 실패한다. 결과를 받을 신규이면서 비어있는 테이블로 레코드를 내보내기 위해 이 방식이 주로 사용된다.

만약 --update-key 인자를 명시한다면 Sqoop은 데이터베이스 내의 기존 데이터셋을 대신 수정할 것이다. 각 입력 레코드는 기존 행을 수정하는 UPDATE 문으로 취급된다. 문장이 수정하는 행은 --update-key로 명시되는 컬럼명에 의해 결정된다. 예를 들면, 다음의 테이블 정의를 고려해보자.:

CREATE TABLE foo(  
id INT NOT NULL PRIMARY KEY,  
msg VARCHAR(32),  
bar INT);

아래와 같은 레코드들을 포함하는 HDFS 내의 어떤 데이터셋을 고려해보자.:

0,this is a test,42  
1,some more data,100  
...

sqoop-export --table foo --update-key id --export-dir /path/to/data --connect …을 실행하면 아래와 같이 해당 데이터를 기반으로 하는 SQL문을 실행하는 내보내기 잡이 실행된다.:

UPDATE foo SET msg='this is a test', bar=42 WHERE id=0;  
UPDATE foo SET msg='some more data', bar=100 WHERE id=1;  
...

UPDATE문이 어떤 행도 수정하지 않는다면, 이것은 에러가 아니다. 내보내기는 조용하게 진행될 것이다. (사실상 이것은 갱신위주의 내보내기는 데이터베이스에 새로운 행으로 삽입되지 않는다는 것을 의미한다.) 비슷한 방식으로, 만약 --update-key로 명시된 컬럼이 행들을 고유하게 인식하지 않고 단일 문장이 다수의 행을 갱신한다면, 이 조건 역시 탐지되지 않는다.

--update-key 인자는 콤마로 분리된 컬럼명의 목록으로 주어진다. 이 같은 경우에는 Sqoop이 기존의 어떤 레코드를 갱신하기 전에 이 목록으로부터 모든 키들을 매칭하려고 할 것이다.

만약 데이터베이스에 갱신할 행들이 있어서 갱신하거나, 아직 존재하지 않아서 삽입하려고 한다면, 대상 데이터베이스에 따라 allowinsert 모드로 --update-mode 인자를 명시할 수 있다.

**표 18. 입력 구문분석 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --input-enclosed-by <문자> | 필요한 필드를 묶는 문자를 설정한다. |
| --input-escaped-by <문자> | 입력의 회피문자를 설정한다. |
| --input-fields-terminated-by <문자> | 입력의 필드 분리자를 설정한다. |
| --input-lines-terminated-by <문자> | 입력의 줄종료 문자를 설정한다. |
| --input-optionally-enclosed-by <문자> | 필드 묶음 문자를 설정한다. |

**표 19. 출력줄 포맷팅 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --enclosed-by <문자> | 필요한 필드를 묶는 문자를 설정한다. |
| --escaped-by <문자> | 회피 문자를 설정한다. |
| --fields-terminated-by <문자> | 필드구분자 문자를 설정한다. |
| --lines-terminated-by <문자> | 줄종료 문자를 설정한다. |
| --mysql-delimiters | MySQL의 기본 구분자 집합을 사용한다. (필드: , 줄: \n escaped-by: \ optionally-enclosed-by: ') |
| --optionally-enclosed-by <문자> | 필드 묶음 문자를 설정한다. |

Sqoop은 데이터베이스로 내보내어지는 데이터를 포함하는 레코드 파일들을 구문분석하고 해석하기 위하여 코드를 자동으로 생성한다. 기본으로 설정된 구분자(개행문자로 분리된 레코드들에 대하여 콤마로 분리된 필드들)가 아닌 구분자와 함께 이 파일들이 생성된다면, 당신은 동일한 구분자를 다시 명시해야 하며 그렇게 함으로써 Sqoop이 파일들을 구문분석할 수 있다.

만약 올바르지 않은 구분자를 명시한다면, Sqoop은 줄 별로 충분한 컬럼들을 찾아내는 데에 실패할 것이다. 내보내기 맵 태스크는 ParseExceptions을 던지면서 실패할 것이다.

**표 20. 코드 생성 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --bindir <디렉토리> | 컴파일된 오브젝트를 위한 출력 디렉토리 |
| --class-name <클래스명> | 생성된 클래스명을 설정한다. 이것은 --package-name을 덮어쓴다. --jar-file와 결합되면 입력 클래스를 설정한다. |
| --jar-file <파일명> | 코드 생성을 비활성화하고 명시된 jar를 사용한다. |
| --outdir <디렉토리명> | 생성된 코드를 위한 출력 디렉토리 |
| --package-name <패키지명> | 지정한 패키지에 자동생성된 클래스들을 넣는다. |
| --map-column-java <매핑명> | 설정된 컬럼에 대하여 SQL타입에서 Java타입으로 매핑하는 기본값을 덮어쓴다. |

만약 내보내어진 레코드들이 이전의 가져오기의 결과로 생성되었다면, 원래의 생성된 클래스를 데이터를 도로 읽어들이는 데에 사용할 수 있다. 이 경우에는 --jar-file과 --class-name을 명시하면 구분자들을 명시할 필요가 없다.

기존의 생성된 코드를 사용하면 --update-key를 함께 쓸 수 없다. 갱신모드 내보내기로 갱신을 수행하기 위해서는 새로운 코드 생성이 필요하다. 당신은 --jar-file을 사용할 수 없으며 기본으로 설정되지 않는 구분자들을 완벽히 명시해야만 한다.

**9.4. 내보내기와 트랙잭션**

내보내기는 병렬로 여러 개의 쓰기로 수행된다. 각 쓰기는 데이터베이스에 대한 분리된 연결을 사용한다. 즉, 서로에 대하여 분리된 트랜잭션을 가진다. Sqoop은 문장 당 100개의 레코드까지 삽입하기 위하여 다중 행 INSERT 문법을 사용한다. 쓰기 태스크 내의 해당 트랜잭션으로서 모든 100개의 문장은 모든 10,000개의 행에 대하여 커밋을 일으키며 커밋된다. 이것은 틀림없이 트랜잭션 버퍼가 경계가 없이 늘어나지 않으며 메모리부족의 조건을 야기시키게 한다. 그러므로 내보내기는 원자성 프로세스가 아니다. 내보내기가 끝나기 전에 내보내기의 부분적인 결과를 볼 수 있을 것이다.

**9.5. 실패한 내보내기**

내보내기는 아래와 같은 여러가지 이유들로 실패할 수도 있다.:

* (하드웨어 장애 또는 서버 소프트웨어 오류 때문에) Hadoop 클러스터에서 데이터베이스로의 연결 상실
* 일관성의 제약성을 위반하는 행을 INSERT하려고 함 (예를 들면, 중복되는 주 키 값을 삽입하는 것)
* HDFS 근원 데이터로부터의 불완전하거나 손상된 레코드를 구문분석하려고 함
* 올바르지 않은 구분자를 사용하여 레코드를 구문분석하려고 함
* (부족한 RAM 또는 디스크 공간과 같은) 용량 이슈

만약 내보내기 맵 태스크가 위와 같은 이유 또는 다른 이유 때문에 실패한다면, 내보내기 잡이 실패하게 된다. 실패한 내보내기의 결과는 정의되지 않는다. 각 내보내기 맵 태스크는 분리된 트랜잭션으로 동작한다. 게다가 개별적인 맵 태스크는 정기적으로 해당 트랜잭션을 커밋한다. 만약 어떤 태스크가 실패하면, 해당 트랜잭션은 롤백될 것이다. 부분적으로 완전한 내보내기를 달성하기 위하여 이전에 커밋된 트랜잭션은 데이터베이스에 고스란히 유지될 것이다.

**9.6. 예제 실시**

bar라고 하는 어떤 테이블을 채우는 기본적인 내보내기는 다음과 같다.:

$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar \  
--export-dir /results/bar\_data

이 예제는 내의 파일들을 취하여 db.example.com의 foo 데이터베이스의 bar 테이블에 해당 파일들의 내용을 주입시킨다. 대상 테이블은 데이터베잉스에 이미 존재해야 한다. Sqoop은 기존 내용과는 관계 없이 INSERT INTO 작업들을 수행한다. 만약 Sqoop이 데이터베이스에 제약성을 위반하는 행들을 삽입하려고 한다면(예를 들면, 특정 주 키 값이 이미 존재하는 경우), 내보내기는 실패할 것이다.

**10. 저장된 잡**

가져오기와 내보내기는 동일 명령을 여러번 내리면서 반복적으로 수행될 수 있다. 특히 증가분 가져오기 기능을 이용할 때 이것은 예상되는 시나리오이다.

Sqoop은 이 프로세스를 용이하게 하기 위하여 *저장된 잡*을 정의하도록 한다. 저장된 잡은 나중에 Sqoop 명령이 실행되는 데에 필요한 설정 정보를 기록한다. sqoop-job 도구의 섹션에서 저장된 잡을 생성하고 이용하는 방법을 설명할 것이다.

기본적으로 잡 명세는 $HOME/.sqoop/에 저장되는 전용 레포지토리에 저장된다. Sqoop이 공유 *메타스토어*를 대신 사용하도록 설정할 수 있으며 메타스토어는 공유 클러스터에 걸쳐서 다수의 사용자들에게 저장된 잡을 사용 가능하도록 한다. 메타스토어를 시작하는 방법은 sqoop-metastore 도구의 섹션에서 다루고 있다.

**11. sqoop-job**

[11.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_4)

[11.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_4)

[11.3. 저장된 잡과 비밀번호](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_saved_jobs_and_passwords)

[11.4. 저장된 잡과 증가분 가져오기](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_saved_jobs_and_incremental_imports)

**11.1. 목적**

잡 도구는 저장된 잡을 생성하고 이용할 수 있도록 한다. 저장된 잡은 잡을 명세할 때 이용되는 매개변수들을 기억한다. 그래서 저장된 잡은 잡의 핸들을 통하여 그 잡을 작동시킴으로써 재실행될 수 있다.

저장된 잡이 증가분 가져오기를 수행하는 데에 구성된다면, 가장 최근에 가져오기된 행들에 대한 상태가 저장된 잡 내에 갱신되어 잡이 지속적으로 가장 최근의 행들만 가져올 수 있도록 한다.

**11.2. 문법**

$ sqoop job (포괄적 인자들) (잡 인자들) [-- [보조도구명] (보조도구 인자들)]  
$ sqoop-job (포괄적 인자들) (잡 인자들) [-- [보조도구명] (보조도구 인자들)]

Hadoop의 포괄적 인자들이 잡 인자들보다 선행하지만, 잡 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 21. 잡 관리 옵션들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --create <잡 아이디> | 명시된 잡 아이디(명)으로 새로운 저장된 잡을 정의한다. --로 분리된 두번째 Sqoop 명령줄이 명시되어야 하며 이것은 저장된 잡을 정의한다. |
| --delete <잡 아이디> | 저장된 잡을 삭제한다. |
| --exec <잡 아이디> | --create로 정의된 어떤 잡이 주어지면, 그 저장된 잡을 실행한다. |
| --show <잡 아이디> | 저장된 잡에 대한 매개변수들을 보여준다. |
| --list | 모든 저장된 잡의 목록을 조회한다. |

--create 액션을 통하여 저장된 잡을 생성한다. 이 작업은 --와 그 뒤를 따르는 도구명과 인자들을 필요로 한다. 도구와 그 인자들은 저장된 잡의 기초를 형성한다. 다음을 생각해보자.:

$ sqoop job --create myjob -- import --connect jdbc:mysql://example.com/db \  
--table mytable

이것은 나중에 실행될 myjob이라고 하는 이름의 잡을 생성한다. 잡이 실행되는 것이 아니다. 잡은 저장된 잡의 목록에서 바로 확인이 가능하다.:

$ sqoop job --list  
Available jobs:  
myjob

show 액션으로 어떤 잡의 설정을 검사할 수 있다.:

$ sqoop job --show myjob  
Job: myjob  
Tool: import  
Options:  
----------------------------  
direct.import = false  
codegen.input.delimiters.record = 0  
hdfs.append.dir = false  
db.table = mytable  
...

그리고 설정에 만족한다면, exec로 그 잡을 실행할 수 있다.:

$ sqoop job --exec myjob  
10/08/19 13:08:45 INFO tool.CodeGenTool: Beginning code generation  
...

exec 액션은 --의 뒤에 인자들을 제공하여 저장된 잡의 인자들을 덮어씌울 수 있게 한다. 예를 들면, 데이터베이스가 사용자명을 요구하도록 변경된다면, 아래와 같이 사용자명과 비밀번호를 명시할 수 있다.:

$ sqoop job --exec myjob -- --username someuser -P  
Enter password:  
...

**표 22. 메타스토어 연결 옵션들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --meta-connect <jdbc-uri> | 메타스토어에 연결하기 위해 쓰이는 JDBC 커넥트 문자열을 명시한다. |

기본적으로 전용 메타스토어는 $HOME/.sqoop 내에 인스턴스화가 된다. 만약 sqoop-metastore 도구로 호스팅되는 메타스토어를 설정했다면, --meta-connect 인자를 명시함으로써 메타스토어에 연결할 수 있다. 이것은 가져오기를 위해 데이터베이스에 연결할 때 쓰이는 것처럼 JDBC 커넥트 문자열이다.

conf/sqoop-site.xml에서 이 주소로 sqoop.metastore.client.autoconnect.url를 설정할 수 있으며 원격 메타스토어를 사용하기 위하여 --meta-connect를 사용할 필요는 없다. 이 매개변수는 전용 메타스토어를 당신의 홈 디렉토리가 아닌 파일시스템 상의 어떤 위치로 이동시킬 때 수정될 수도 있다.

만약 sqoop.metastore.client.enable.autoconnect을 false값으로 설정한다면, 명시적으로 -meta-connect를 사용해야만 한다.

**표 23. 공통 옵션들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --help | 사용법을 출력한다. |
| --verbose | 작업 중 더 많은 정보를 출력한다. |

**11.3. 저장된 잡과 비밀번호**

Sqoop 메타스토어는 보안성이 있는 자원이 아니다. 다수의 사용자가 그 내용에 접근할 수 있다. 이러한 이유로 Sqoop은 메타스토어에 비밀번호를 저장하지 않는다. 만약 비밀번호를 요구하는 잡을 생성하면, 그 잡을 실행할 때마다 비밀번호 입력을 위한 프롬프트를 볼 것이다.

설정에서 sqoop.metastore.client.record.password를 true로 설정함으로써 메타스토어에서 비밀번호를 활성화시킬 수 있다.

**11.4. 저장된 잡과 증가분 가져오기**

증가분 가져오기는 가장 최근의 가져오기의 참조 값에 대하여 *검사 컬럼* 내의 값들을 비교함으로써 수행된다. 예를 들면, --check-column id, --last-value 100과 함께 --incremental append 인자가 명시되면, id > 100인 모든 행들이 가져오기가 될 것이다. 증가분 가져오기가 명령줄에서 실행된다면, 차후의 증가분 가져오기 작업에서 --last-value로 명시되는 값은 참조를 위하여 화면에 출력될 것이다. 증가분 가져오기가 저장된 잡으로 실행된다면, 이 값은 저장된 잡 내에 함유되어 있을 것이다. sqoop job --exec someIncrementalJob을 차후에 실행할 때, 이전에 가져온 행들보다 새로운 행들만을 계속 가져올 것이다.

**12. sqoop-metastore**

[12.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_5)

[12.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_5)

**12.1. 목적**

metastore 도구는 Sqoop이 공유된 메타데이터 저장소를 호스팅할 수 있도록 구성한다. 다수의 사용자와 (또는) 원격 사용자들은 이 메타스토어 내에 정의되는 (sqoop job으로 생성된) 저장된 잡을 정의하고 실행할 수 있다.

클라이언트는 sqoop-site.xml 내에서 또는 --meta-connect 인자로 메타스토어에 연결되도록 설정되어야 한다.

**12.2. 문법**

$ sqoop metastore (포괄적 인자들) (메타스토어 인자들)  
$ sqoop-metastore (포괄적 인자들) (메타스토어 인자들)

Hadoop의 포괄적 인자들이 메타스토어 인자들보다 선행하지만, 메타스토어 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 24. 메타스토어 관리 옵션들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --shutdown | 동일 서버 상에서 실행중인 메타스토어 인스턴스를 내린다. |

sqoop-metastore을 실행하면 현재의 서버 상에 공유된 HSQLDB 데이터베이스 인스턴스를 시작한다. 클라이언트는 이 메타스토어에 접속하고 실행을 위하여 사용자들 간에 공유될 수 있는 잡들을 생성할 수 있다.

디스크 상에서 메타스토어 파일들의 위치는 conf/sqoop-site.xml 파일 내의 sqoop.metastore.server.location 속성에 의해서 제어된다. 이것은 지역 파일시스템 상의 특정 디렉토리를 가리켜야 한다.

메타스토어는 TCP/IP 상에서 사용가능하다. 포트는 sqoop.metastore.server.port 설정 매개변수에 의하여 제어되며 기본값은 16000이다.

클라이언트는 sqoop.metastore.client.autoconnect.url 또는 -meta-connect를 jdbc:hsqldb:hsql://<server-name>:<port>/sqoop 값으로 명시하여 메타스토어에 연결해야 한다. 예를 들면, jdbc:hsqldb:hsql://metaserver.example.com:16000/sqoop.

이 메타슽어는 Hadoop 클러스터 내의 서버 또는 네트워크 상의 어느 곳에서도 호스팅될 수 있다.

**13. sqoop-merge**

[13.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_6)

[13.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_6)

**13.1. 목적**

병합 도구는 하나의 데이터셋의 엔트리들이 더 오래된 데이터 셋의 엔트리들을 덮어써야 하는 두 개의 데이터셋을 합칠 수 있도록 한다. 예를 들면, 증가분 가져오기가 last-modified 모드로 작동할 때, 각 데이터셋에서 잇따라 나타나는 HDFS의 다중 데이터셋들을 생성할 수 있다. merge 도구는 각각의 주 키에 대하여 가장 최신이며 사용 가능한 레코들들을 취함으로써 두 개의 데이터셋을 하나로 "편평하게 만든다”.

**13.2. 문법**

$ sqoop merge (포괄적 인자들) (병합 인자들)  
$ sqoop-merge (포괄적 인자들) (병합 인자들)

Hadoop의 포괄적 인자들이 병합 인자들보다 선행하지만, 병합 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 25. 병합 옵션들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --class-name <클래스명> | 병합 잡이 수행되는 동안 사용되며 레코드에 특정한 클래스명을 명시한다. |
| --jar-file <파일명> | 레코드 클래스를 로딩할 jar파일명을 명시한다. |
| --merge-key <컬럼명> | 병합 키로 사용할 컬렴명을 명시한다. |
| --new-data <경로> | 더 새로운 데이터셋의 경로를 명시한다. |
| --onto <경로> | 더 오래된 데이터셋의 경로를 명시한다. |
| --target-dir <경로> | 병합 잡의 출력을 위히나 대상 경로를 명시한다. |

merge 도구는 더 새로운 데이터셋과 더 오래된 데이터셋을 입력으로서 두 개의 디렉토리를 취하는 맵리듀스 잡을 실행한다. 이 데이터셋들은 --new-data와 --onto로 각각 명시된다. 맵리듀스 잡의 출력은 --target-dir로 명시되는 HDFS 내의 디렉토리에 위치할 것이다.

데이테셋을 병합할 때, 각 레코드에 유일한 주 키 값이 있다고 가정한다. 주 키의 컬럼은 --merge-key로 명시된다. 동일 데이터셋 내의 다중 행은 동일한 주 키를 가져서는 안되며 그렇지 않으면 데이터 손실이 발생할 수 있다.

데이터셋을 구문분석하고 키 컬럼을 추출하기 위해서는 이전의 가져오기에서 자동생성된 클래스가 사용되어야 한다. -class-name과 --jar-file로 클래스명과 jar 파일명을 명시해야 한다. 만약 이것을 사용할 수 없다면 codegen 도구를 이용하여 그 클래스를 재생성 해낼 수 있다.

병합 도구는 일반적으로 date-last-modified 모드를 증가분 가져오기가 끝난 후 실행된다(sqoop import --incremental lastmodified …).

두 개의 증가분 가져오기가 수행된다고 가정해보자. 더 오래된 데이터가 older라고 하는 HDFS 디렉토리에 내에 있으며, 더 새로운 데이터가 newer라고 하는 HDFS 디렉토리 내에 있을 때, 이것들은 아래와 같이 병합될 수 있다.:

$ sqoop merge --new-data newer --onto older --target-dir merged \  
--jar-file datatypes.jar --class-name Foo --merge-key id

위 명령은 맵리듀스 잡을 실행할 것이다. 잡에서는 각 행의 id 컬럼의 값이 행을 조인하는 데에 사용된다. 여기서 newer 데이터셋 내의 행은 older 데이터셋 내의 행보다 우선적으로 사용될 것이다.

이것은 SequenceFile-, Avro-, 텍스트기반의 증가분 가져오기에서 사용될 수 있다. 더 새롭고 오래된 데이터셋의 파일 유형은 반드시 같아야 한다.

**14. sqoop-codegen**

[14.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_7)

[14.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_7)

[14.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_4)

**14.1. 목적**

codegen 도구는 가져온 레코들들을 캡슐화하고 해석하는 Java 클래스들을 생성한다. 어떤 레코드의 Java 정의는 가져오기 프로세스의 일부분으로서 인스턴스화되지만 별도로 수행될 것이다. 예를 들면, Java 소스가 유실되면 재생성할 수 있다. 어떤 클래스의 새로운 버전은 필드나 다른 것들 간의 서로 다른 구분자를 사용하여 생성될 수 있다.

**14.2. 문법**

$ sqoop codegen (포괄적 인자들) (codegen 인자들)  
$ sqoop-codegen (포괄적 인자들) (codegen 인자들)

Hadoop의 포괄적 인자들이 codegen 인자들보다 선행하지만, codegen 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 26. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <디렉토리> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

**표 27. 코드 생성 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -bindir <디렉토리> | 컴파일된 오브젝트를 위한 출력 디렉토리 |
| -class-name <클래스명> | 생성된 클래스명을 설정한다. 이것은 --package-name을 덮어쓴다. --jar-file와 결합되면 입력 클래스를 설정한다. |
| -jar-file <파일명> | 코드 생성을 비활성화하고 명시된 jar를 사용한다. |
| -outdir <디렉토리명> | 생성된 코드를 위한 출력 디렉토리 |
| -package-name <패키지명> | 지정한 패키지에 자동생성된 클래스들을 넣는다. |
| -map-column-java <매핑명> | 설정된 컬럼에 대하여 SQL타입에서 Java타입으로 매핑하는 기본값을 덮어쓴다. |

**표 28. 출력줄 포맷팅 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -enclosed-by <문자> | 필요한 필드를 묶는(enclose) 문자를 설정한다. |
| -escaped-by <문자> | 회피(escaping) 문자를 설정한다. |
| -fields-terminated-by <문자> | 필드구분자 문자를 설정한다. |
| -lines-terminated-by <문자> | 줄종료 문자를 설정한다. |
| -mysql-delimiters | MySQL의 기본 구분자 집합을 사용한다. (필드: , 줄: \n escaped-by: \ optionally-enclosed-by: ') |
| -optionally-enclosed-by <문자> | 필드 묶음(enclosing) 문자를 설정한다. |

**표 29. 입력 구문분석 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -input-enclosed-by <문자> | 필요한 필드를 묶는 문자를 설정한다. |
| -input-escaped-by <문자> | 입력의 회피문자를 설정한다. |
| -input-fields-terminated-by <문자> | 입력의 필드 구분자를 설정한다. |
| -input-lines-terminated-by <문자> | 입력의 줄종료 문자를 설정한다. |
| -input-optionally-enclosed-by <문자> | 필드 묶음 문자를 설정한다. |

**표 30. Hive 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -hive-home <디렉토리> | $HIVE\_HOME를 덮어쓴다. |
| -hive-import | 테이블을 Hive로 가져온다. (아무것도 설정되지 않으면, Hive의 기본 구분자를 사용한다.) |
| -hive-overwrite | Hive 테이블에 존재하는 데이터를 덮어쓴다. |
| -create-hive-table | 대상 Hive 테이블이 존재할 때 잡이 실패하도록 한다. 기본적으로 이 프로퍼티는 false이다. |
| -hive-table <테이블명> | Hive로 가져오기를 수행할 때 사용할 테이블명을 설정한다. |
| -hive-drop-import-delims | Hive로 가져오기를 수행할 때 문자열에 있는 *\n*,*\r*, *\01*를 없앤다. |
| -hive-delims-replacement | Hive로 가져오기를 수행할 때 문자열에 있는 *\n*,*\r*, *\01*를 사용자가 정의한 문자열로 대체한다. |
| -hive-partition-key | 파티션을 나눌 때 사용할 Hive 필드의 이름 |
| -hive-partition-value <값> | 해당 잡에서 Hive로 가져올 때 파티션 키로 사용되는 문자열 값. |
| -map-column-hive <매핑명> | 설정된 컬럼들에 대하여 SQL타입에서 Hive타입으로 매핑하는 기본값을 덮어쓴다. |

만약 Hive 인자들이 코드 생성 도구에 제공된다면, Sqoop은 테이블을 생성하고 데이터를 로딩하기 위하여 HQL 문을 포함하는 파일을 생성한다.

**14.3. 예제 실시**

기업 데이터베이스의 employees 테이블에 대하여 레코드 해석 코드를 재생성한다.:

$ sqoop codegen --connect jdbc:mysql://db.example.com/corp \  
--table employees

**15. sqoop-create-hive-table**

[15.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_8)

[15.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_8)

[15.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_5)

**15.1. 목적**

create-hive-table 도구는 HDFS에 이전에 가져오기 되었던 데이터베이스 테이블 또는 가져오기 될 예정인 테이블에 기반한 테이블의 정의를 Hive 메타스토어에 채운다. 이것은 선행하는 가져오기를 실행하지 않고 sqoop-import의 --hive-import 단계를 효과적으로 수행한다.

만약 데이터가 HDFS에 이미 로딩되어 있다면, 그 데이터를 Hive에 가져오기 위한 파이프라인을 끝내는 데에 이 도구를 사용할 수 있다. 이 도구로 Hive 테이블을 생성할 수도 있다. 그러면 데이터는 사용자에 의하여 실행된 처리 단계가 끝난 후 목적대상에 가져오기 되고 채워진다.

**15.2. 문법**

$ sqoop create-hive-table (포괄적 인자들) (create-hive-table 인자들)  
$ sqoop-create-hive-table (generic-args) (create-hive-table-args)

Hadoop의 포괄적 인자들이 create-hive-table 인자들보다 선행하지만, create-hive-table 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 31. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <디렉토리> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

**표 32. Hive 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -hive-home <디렉토리> | $HIVE\_HOME를 덮어쓴다. |
| -hive-overwrite | Hive 테이블에 존재하는 데이터를 덮어쓴다. |
| -create-hive-table | 대상 Hive 테이블이 존재할 때 잡이 실패하도록 한다. 기본적으로 이 프로퍼티는 false이다. |
| -hive-table <테이블명> | Hive로 가져오기를 수행할 때 사용할 테이블명을 설정한다. |
| --table | 정의를 읽어들일 데이터베이스 테이블 |

**표 33. 출력줄 포맷팅 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -enclosed-by <문자> | 필요한 필드를 묶는(enclose) 문자를 설정한다. |
| -escaped-by <문자> | 회피(escaping) 문자를 설정한다. |
| -fields-terminated-by <문자> | 필드구분자 문자를 설정한다. |
| -lines-terminated-by <문자> | 줄종료 문자를 설정한다. |
| -mysql-delimiters | MySQL의 기본 구분자 집합을 사용한다. (필드: , 줄: \n escaped-by: \ optionally-enclosed-by: ') |
| -optionally-enclosed-by <문자> | 필드 묶음(enclosing) 문자를 설정한다. |

Hive로 가져올 때 사용되는 출력 포맷팅 인자들과 함께 enclosed-by 또는 escaped-by 구분자를 사용하지 말라. Hive는 현재 그것들을 구문분석할 수 없다.

**15.3. 예제 실시**

employees라고 하는 데이터베이스 테이블을 기반으로 하는 정의로 emps라고 하는 테이블을 Hive로 정의해보자.:

$ sqoop create-hive-table --connect jdbc:mysql://db.example.com/corp \  
--table employees --hive-table emps

**16. sqoop-eval**

[16.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_9)

[16.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_9)

[16.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_6)

**16.1. 목적**

eval 도구는 사용자가 데이터베이스에 대하여 간단한 SQL 쿼리를 빠르게 실행할 수 있도록 한다. 결과는 콘솔에 출력된다. 이것은 사용자가 기대하는 데이터를 가져오는 것을 확실히 하기 위하여 가져오기 쿼리를 미리 볼 수 있도록 한다.

**16.2. 문법**

$ sqoop eval (포괄적 인자들) (eval 인자들)  
$ sqoop-eval (포괄적 인자들) (eval 인자들)

Hadoop의 포괄적 인자들이 eval 인자들보다 선행하지만, eval 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 34. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <디렉토리> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

**표 35. SQL 평가 인자들:**

|  |  |
| --- | --- |
| 인자 | 설명 |
| -e,--query <문장> | SQL 내의 *문장*을 실행한다. |

**16.3. 예제 실시**

employees 테이블로부터 열 개의 레코드를 선택하자.:

$ sqoop eval --connect jdbc:mysql://db.example.com/corp \  
--query "SELECT \* FROM employees LIMIT 10"

foo 테이블에 행을 삽입하자.:

$ sqoop eval --connect jdbc:mysql://db.example.com/corp \  
-e "INSERT INTO foo VALUES(42, 'bar')"

**17. sqoop-list-databases**

[17.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_10)

[17.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_10)

[17.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_7)

**17.1. 목적**

서버에 있는 데이터베이스 스키마들의 목록을 출력한다.

**17.2. 문법**

$ sqoop list-databases (포괄적 인자들) (list-databases 인자들)  
$ sqoop-list-databases (포괄적 인자들) (list-databases 인자들)

Hadoop의 포괄적 인자들이 list-databases 인자들보다 선행하지만, list-databases 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 36. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <디렉토리> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

**17.3. 예제 실시**

MySQL 서버의 가용한 데이터베이스 스키마들의 목록을 조회해보자.:

$ sqoop list-databases --connect jdbc:mysql://database.example.com/  
information\_schema  
employees

|  |  |
| --- | --- |
|  | Note |
|  | 이 명령어는HSQLDB, MySQL, Oracle에 대해서만 동작한다. Oracle을 사용할 때 데이터베이스에 연결하는 사용자는 DBA 권한을 가져야 한다. |

**18. sqoop-list-tables**

[18.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_11)

[18.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_11)

[18.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_8)

**18.1. 목적**

데이터베이스에 있는 테이블들으리 목록을 출력한다.

**18.2. 문법Syntax**

$ sqoop list-tables (포괄적 인자들) (list-tables 인자들)  
$ sqoop-list-tables (포괄적 인자들) (list-tables 인자들)

Hadoop의 포괄적 인자들이 list-tables 인자들보다 선행하지만, list-tables 인자들은 서로 순서에 관계 없이 입력될 수 있다.

**표 37. 공통 인자들**

|  |  |
| --- | --- |
| 인자 | 설명 |
| --connect <jdbc-uri> | JDBC 연결 문자열을 명시한다. |
| --connection-manager <클래스명> | 사용할 연결 관리자 클래스를 명시한다. |
| --driver <클래스명> | 사용할 JDBC 드라이버 클래스를 별도로 명시한다. |
| --hadoop-home <디렉토리> | $HADOOP\_HOME을 덮어 쓴다. |
| --help | 사용법을 출력한다. |
| -P | 콘솔에서 비밀번호를 읽어들인다. |
| --password <비밀번호> | 인증 비밀번호를 설정한다. |
| --username <사용자명> | 인증 사용자명을 설정한다. |
| --verbose | 동작 시 더 많은 정보를 출력한다. |
| --connection-param-file <파일명> | 연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다. |

**18.3. 예제 실시**

"corp" 데이터베이스 내에 가용한 테이블들의 목록을 출력해보자.:

$ sqoop list-tables --connect jdbc:mysql://database.example.com/corp  
employees  
payroll\_checks  
job\_descriptions  
office\_supplies

**19. sqoop-help**

[19.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_12)

[19.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_12)

[19.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_9)

**19.1. 목적**

Sqoop 내에서 가용한 도구들의 목록을 출력하고 사용법을 설명한다.

**19.2. 문법**

$ sqoop help [도구명]  
$ sqoop-help [도구명]

도구명이 제공되지 않으면(예를 들면, 사용자가 sqoop help라고 실행하면), 가용한 도구의 목록이 출력된다. 도구명과 함께 사용하면, 해당 도구의 사용법이 콘솔에 표시된다.

**19.3. 예제 실시**

가용한 도구들의 목록을 출력해보자.:

$ sqoop help  
usage: sqoop COMMAND [ARGS]  
  
Available commands:  
codegen Generate code to interact with database records  
create-hive-table Import a table definition into Hive  
eval Evaluate a SQL statement and display the results  
export Export an HDFS directory to a database table  
  
...  
  
See 'sqoop help COMMAND' for information on a specific command.

import 도구의 사용법을 표시해보자.:

$ bin/sqoop help import  
usage: sqoop import [GENERIC-ARGS] [TOOL-ARGS]  
  
Common arguments:  
--connect <jdbc-uri> Specify JDBC connect string  
--connection-manager <class-name> Specify connection manager class to use  
--driver <class-name> Manually specify JDBC driver class to use  
--hadoop-home <dir> Override $HADOOP\_HOME  
--help Print usage instructions  
-P Read password from console  
--password <password> Set authentication password  
--username <username> Set authentication username  
--verbose Print more information while working  
  
Import control arguments:  
--as-avrodatafile Imports data to Avro Data Files  
--as-sequencefile Imports data to SequenceFiles  
--as-textfile Imports data as plain text (default)  
...

**20. sqoop-version**

[20.1. 목적](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_purpose_13)

[20.2. 문법](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_syntax_13)

[20.3. 예제 실시](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_example_invocations_10)

**20.1. 목적**

Sqoop의 버전 정보를 표시한다.

**20.2. 문법**

$ sqoop version  
$ sqoop-version

**20.3. 예제 실시**

버전을 조회해보자.:

$ sqoop version  
Sqoop {revnumber}  
git commit id 46b3e06b79a8411320d77c984c3030db47dd1c22  
Compiled by aaron@jargon on Mon May 17 13:43:22 PDT 2010

**21. 호환성 정보**

[21.1. 지원 데이터베이스](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_supported_databases)

[21.2. MySQL](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_mysql)

[21.2.1. zeroDateTimeBehavior](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_zerodatetimebehavior)

[21.2.2. UNSIGNED 컬럼](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_unsigned_literal_columns)

[21.2.3. BLOB과 CLOB 컬럼](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_blob_literal_and_literal_clob_literal_columns)

[21.2.4. Direct-mode 트랜잭션](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_direct_mode_transactions)

[21.3. Oracle](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle)

[21.3.1. 날짜와 시간](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_dates_and_times)

[21.4. Hive에서의 스키마 정의](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_schema_definition_in_hive)

Sqoop은 데이터베이스에 연결하기 위하여 JDBC를 사용하고 공표된 표준들을 가능한 한 많이 지킨다. 표준에 부합하는 SQL을 지원하지 않는 데이터베이스에 대하여 Sqoop은 기능성을 제공하기 위하여 대체 코드경로를 사용한다. 일반적으로 Sqoop은 많은 데이터베이스와 호환되는 것으로 여겨지지만 단지 몇 개에 대해서만 시험한다.

그럼에도 불구하고 Sqoop을 구현할 때, 데이터베이스에 특정한 여러 개의 결정들이 이루어졌으며 몇몇의 데이터베이스에 대하여 표준을 확장하는 추가적인 설정이 필요하다.

이 섹션에서 Sqoop으로 시험된 데이터베이스들, Sqoop이 표준과 관련하여 각 데이터베이스를 다루는 데 있어서의 예외사항, Sqoop 내에서 가용하며 데이터베이스에 특정한 설정들을 설명한다.

**21.1. 지원 데이터베이스**

JDBC가 공통 API를 통하여 프로그램이 다른 많은 데이터베이스에 접근하도록 하는 호환성 레이어인 반면, 각 데이터베이스에서 활용되는 SQL 언어 상의 미묘한 차이는 Sqoop이 모든 데이터베이스를 독창적으로 사용할 수 없거나 데이터베이스를 비효율적인 방법으로 사용할 수 있다는 것을 의미할 것이다.

When you provide a connect string to Sqoop에 커넥트 문자열을 제공했을 때, Sqoop은 적절하고 벤더에 특정하고 로직을 결정하기 위하여 프로토콜 scheme을 검사한다. 만약 Sqoop이 주어진 데이터베이스에 대하여 알고 있다면 자동적으로 동작할 것이다. 그렇지 않다면, --driver를 통하여 로딩할 드라이버 클래스를 명시할 필요가 있을 것이다. 이 방법은 데이터베이스를 접근하기 위하여 표준 SQL을 사용할 포괄적인 코드경로를 사용할 것이다. Sqoop은 일부 데이터베이스들에게 더 빠르고 JDBC 기반이 아닌 접근 매커니즘을 제공하기도 한다. 이 방법은 --direct 매개변수를 명시함으로써 가능하게 할 수 있다.

Sqoop은 다음의 데이터베이스들에 대하여 벤더에 특정한 지원을 포함한다.:

|  |  |  |  |
| --- | --- | --- | --- |
| 데이터베이스 | 버전 | --direct 지원? | 커넥트 문자열 매치 |
| HSQLDB | 1.8.0+ | 아니오 | jdbc:hsqldb:\*// |
| MySQL | 5.0+ | 예 | jdbc:mysql:// |
| Oracle | 10.2.0+ | 아니오 | jdbc:oracle:\*// |
| PostgreSQL | 8.3+ | 예 (내보내기만) | jdbc:postgresql:// |

Sqoop은 목록에 표시된 데이터베이스들의 더 오래된 버전들과도 동작할 것이다. 하지만 위에 명시한 버전으로만 시험했다.

만약 Sqoop이 내부적으로 데이터베이스를 지원한다고 하더라도 클라이언트의 $SQOOP\_HOME/lib 경로에 데이터베이스 벤더의 JDBC 드라이버를 여전히 설치할 필요가 있다. Sqoop은 클라이언트의 $SQOOP\_HOME/lib의 jar파일들로부터 클래스들을 로딩할 수 있으며 Sqoop이 실행하는 맵리듀스 잡의 일부로 그 클래스들을 사용할 수 있다. 더 오래된 버전들과는 달리 서버의 Hadoop 라이브러리 경로에 JDBC jar파일을 더이상 설치할 필요가 없다.

**21.2. MySQL**

[21.2.1. zeroDateTimeBehavior](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_zerodatetimebehavior)

[21.2.2. UNSIGNED 컬럼](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_unsigned_literal_columns)

[21.2.3. BLOB과 CLOB 컬럼](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_literal_blob_literal_and_literal_clob_literal_columns)

[21.2.4. Direct-mode 트랜잭션](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_direct_mode_transactions)

JDBC 드라이버: [MySQL Connector/J](http://www.mysql.com/downloads/connector/j/)

MySQL v5.0 이상은 Sqoop이 매우 잘 지원한다. Sqoop은 mysql-connector-java-5.1.13-bin.jar로 시험되었다.

***21.2.1. zeroDateTimeBehavior***

MySQL은 DATE 컬럼에 대하여 SQL에 대한 비표준 확장인 '0000-00-00\' 값들을 허용한다. JDBC를 통하여 통신할 때, 이 값들은 세 가지의 다른 방식 중 하나로 다루어진다.:

* NULL로 변환한다.
* 클라이언트에서 예외를 던진다.
* 가장 가까운 날짜('0001-01-01\')로 반올림 한다.

커넥트 문자열의 zeroDateTimeBehavior 속성을 사용하여 그 행위를 명시한다. 만약 zeroDateTimeBehavior이 명시되지 않는다면, Sqoop은 convertToNull 행위를 사용할 것이다.

이 행위를 오버라이딩할 수 있다. 예를 들면:

$ sqoop import --table foo \  
--connect jdbc:mysql://db.example.com/someDb?zeroDateTimeBehavior=round

***21.2.2. UNSIGNED 컬럼***

MySQL에서 UNSIGNED 타입의 컬럼들은 0과 2^32 (4294967295) 사이의 값들을 가질 수 있다. 그러나 데이터베이스는 INTEGER로 데이터 타입을 보고할 것이며 이 타입은 -2147483648과 \+2147483647 사이의 값을 가질 수 있다. Sqoop은 현재2147483647 이상의 UNSIGNED 값들을 가져오기할 수 없다.

***21.2.3. BLOB과 CLOB 컬럼***

Sqoop의 직접 모드는 BLOB, CLOB, 또는 LONGVARBINARY 컬럼들의 가져오기를 지원하지 않는다. 이 컬럼들에 대해서는 JDBC 기반의 가져오기를 사용하고 가져오기 도구에 --direct 인자를 제공하지 않는다.

***21.2.4. Direct-mode 트랜잭션***

성능을 위하여 각 라이터는 내보내어지는 데이터의 약 모든 32MB 마다 현재 트랜잭션을 커밋할 것이다. 어떠한 도구에 특정한 인자들을 사용하기 전에 다음의 인자를 명시함으로써 이것을 제어할 수 있다.: -D sqoop.mysql.export.checkpoint.bytes=size, 여기서 *size*는 바이트 값이다. 중간 체크포인트를 사용하지 않기 위해 0으로 *size*를 설정하라. 그러나 내보내기 되고 있는 개별 파일들은 서로에 독립적으로 계속적으로 커밋될 것이다.

|  |  |
| --- | --- |
|  | *중요* |
|  | *-D parameter=value 형태인 Sqoop에 대한 어떤 인자라도 Hadoop의 포괄적인 인자들이며 도구에 특정적인 어떤 인자들(예를 들면, --connect, --table 등)보다도 앞에 나타나야 한다는 점을 명심하자.* |

**21.3. Oracle**

[21.3.1. 날짜와 시간](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_dates_and_times)

JDBC 드라이버: [Oracle JDBC Thin Driver](http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_112010.html) - Sqoop은 ojdbc6.jar와 호환된다.

Sqoop은 Oracle 10.2.0 Express Edition과 시험되었다. Oracle은 SQL에 대하여 ANSI 표준과 다른 방식의 접근과 비표준 JDBC 드라이버에 있어서 주목할 만 하다. 그러므로, 몇 가지의 기능이 다르게 동작한다.

***21.3.1. 날짜와 시간***

Oracle JDBC는 TIMESTAMP 값으로 DATE와 TIME SQL 타입들을 표현한다. Oracle 데이터베이스 내의 어떠한 DATE 컬럼들은 Sqoop에서 TIMESTAMP로 가져오기 될 것이고, Sqoop이 생성한 코드는 java.sql.Timestamp 필드로 이 값들을 저장할 것이다.

데이터를 데이터베이스로 도로 내보내기를 할 때, Sqoop은 텍스트 필드들을 (yyyy-mm-dd HH:MM:SS.ffffffff 형태로) TIMESTAMP 타입으로서 구문분석할 것이다. 당신이 이 필드들이 yyyy-mm-dd의 JDBC 날짜 회피(escape) 포맷으로 포맷되기를 기대한다고 하더라도 말이다. Oracle로 내보내진 날짜는 완전한 타임스탬프로서 포맷될 것이다.

Oracle은 또한 날짜/시간 추가 타입으로 TIMESTAMP WITH TIMEZONE와 TIMESTAMP WITH LOCAL TIMEZONE도 포함한다. 이 타입들을 지원하기 위하여 사용자의 세션 시간대가 명시되어야 한다. 기본적으로 Sqoop은 Oracle에 "GMT" 시간대를 명시할 것이다. Sqoop 잡을 실행할 때, 명령줄에서 oracle.sessionTimeZone Hadoop 속성을 명시함으로써 이 설정을 오버라이딩할 수 있다. 예를 들면:

$ sqoop import -D oracle.sessionTimeZone=America/Los\_Angeles \  
--connect jdbc:oracle:thin:@//db.example.com/foo --table bar

Hadoop 매개변수(-D …)는 *포괄적인 인자들*이며 반드시 도구에 특정적인 인자들(--connect, --table 등)보다 앞에 있어야 함을 주지하라.

세션 시간대 문자열에 대하여 적법한 값들은 다음의 사이트에 열거되어 있다. <http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14225/applocaledata.htm#i637736>.

**21.4. Hive에서의 스키마 정의**

Hive 사용자들은 SQL 타입과 Hive 타입 간의 일대일 매핑이 없다는 점을 주지할 것이다. 일반적으로 직접 매핑(예를 들면, DATE, TIME, TIMESTAMP)을 갖지 않는 SQL 타입들은 Hive에서 STRING으로 강제될 것이다. NUMERIC과 DECIMAL SQL 타입들은DOUBLE로 강제될 것이다. 이러한 경우, Sqoop은 정밀한 값의 손실을 알려주기 위하여 로그 메세지에 경고를 표시한다.

**22. 지원 얻기**

일반적인 정보는 <http://incubator.apache.org/sqoop/>에 있다.

<https://issues.apache.org/jira/browse/SQOOP>에서 이슈 추적기에 Sqoop의 버그를 보고하라.

Sqoop 사용법에 관한 질문과 토론을 [sqoop-user mailing list](http://incubator.apache.org/sqoop/mail-lists.html)로 보내주어야 한다.

포럼에 문의하기 전에 가능한 한 많은 디버깅 정보를 얻기 위하여 --verbose 플래그로 Sqoop 잡을 실행하라. 실행하고 있는 Hadoop 버전(hadoop version) 뿐만 아니라 sqoop version에 의해 반환되는 문자열도 보고하라.

**23. 트러블슈팅**

[23.1. 일반적인 트러블슈팅 프로세스](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_general_troubleshooting_process)

[23.2. 특화된 트러블슈팅 팁](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_specific_troubleshooting_tips)

[23.2.1. Oracle: 연결 리셋 에러](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle_connection_reset_errors)

[23.2.2. Oracle: Case-Sensitive 카탈로그 쿼리 에러](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle_case_sensitive_catalog_query_errors)

23.2.3. MySQL: 연결 실패

**23.1. 일반적인 트러블슈팅 프로세스**

Sqoop을 실행하는 동안 마주치는 실패를 트러블슈팅하기 위하여 다음 단계를 따라야 한다.

* --verbose 옵션을 명시하고 동일 명령을 다시 실행하여 verbose 출력을 작동시켜라. 이것은 명백한 에러를 분별하기 위하여 검사될 수 있는 콘솔 상의 디버그 아웃풋을 만들어낸다.
* 특정한 실패가 기록되지는 않았는지를 보기 위하여 Hadoop의 태스크 로그를 보라. 태스크 실행 중 일어나는 실패가 정확하게 콘솔에 전달되지 않는 가능성이 있다.
* 필요한 입력 파일들 또는 입력/출력 테이블들이 있고, Sqoop이 실행하고 데이터베이스에 연결할 때 쓰이는 사용자 계정으로 접근이 될 수 있는지 확인하라. 필요한 파일들이나 테이블은 제공되지만 Sqoop이 연결하는 데에 쓰이는 특정 사용자 계정이 이 파일들에 접근하기 위한 필요 권한들을 갖고 있지 않을 수도 있다.
* 만약 Hive 테이블이나 파티션을 채우는 것과 같은 복잡한 동작을 하고 있다면, 어디서 문제가 실제로 발생하는지 알아보기 위하여 해당 잡을 두 개의 분리된 잡으로 쪼개어 시도해보라. 예를 들면, 만약 어떤 Hive 테이블을 생성하고 채우는 가져오기가 실패하고 있다면, 이것을 두 단계로 분해할 수 있다. 첫번째로 가져오기를 단독적으로 실행하고, 두번째로 create-hive-table 도구를 사용하여 가져오기 없이 Hive 테이블을 생성한다. 이 방법이 Hive 테이블을 채우는 본래의 유스케이스를 설명하는 것은 아니지만, 일반적인 내보내기 또는 Hive 테이블을 생성하고 채우는 데 있어서의 문제를 좁히는 데에 도움이 된다.
* 해당 문제에 관련된 키워드로 메일링 리스트 아카이브를 검색하라. 거기에서 논의되는 내용 중에서 당신의 문제를 해결하거나 임시조치 하는 데에 도움이 되는 해결방법을 찾을 가능성이 있다.

**23.2. 특화된 트러블슈팅 팁**

[23.2.1. Oracle: 연결 리셋 에러](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle_connection_reset_errors)

[23.2.2. Oracle: Case-Sensitive 카탈로그 쿼리 에러](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_oracle_case_sensitive_catalog_query_errors)

[23.2.3.](http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html#_mysql_connection_failure) MySQL: 연결 실패

***23.2.1. Oracle: 연결 리셋 에러***

문제: Oracle을 위한 기본 Sqoop 커넥터를 사용할 때, 데이터가 전송되지만 맵리듀스 잡이 수행되는 동안 아래와 같이 많은 에러가 보고된다.:

11/05/26 16:23:47 INFO mapred.JobClient: Task Id : attempt\_201105261333\_0002\_m\_000002\_0, Status : FAILED  
java.lang.RuntimeException: java.lang.RuntimeException: java.sql.SQLRecoverableException: IO Error: Connection reset  
at com.cloudera.sqoop.mapreduce.db.DBInputFormat.setConf(DBInputFormat.java:164)  
at org.apache.hadoop.util.ReflectionUtils.setConf(ReflectionUtils.java:62)  
at org.apache.hadoop.util.ReflectionUtils.newInstance(ReflectionUtils.java:117)  
at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:605)  
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:322)  
at org.apache.hadoop.mapred.Child$4.run(Child.java:268)  
at java.security.AccessController.doPrivileged(Native Method)  
at javax.security.auth.Subject.doAs(Subject.java:396)  
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1115)  
at org.apache.hadoop.mapred.Child.main(Child.java:262)  
Caused by: java.lang.RuntimeException: java.sql.SQLRecoverableException: IO Error: Connection reset  
at com.cloudera.sqoop.mapreduce.db.DBInputFormat.getConnection(DBInputFormat.java:190)  
at com.cloudera.sqoop.mapreduce.db.DBInputFormat.setConf(DBInputFormat.java:159)  
... 9 more  
Caused by: java.sql.SQLRecoverableException: IO Error: Connection reset  
at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:428)  
at oracle.jdbc.driver.PhysicalConnection.<init>(PhysicalConnection.java:536)  
at oracle.jdbc.driver.T4CConnection.<init>(T4CConnection.java:228)  
at oracle.jdbc.driver.T4CDriverExtension.getConnection(T4CDriverExtension.java:32)  
at oracle.jdbc.driver.OracleDriver.connect(OracleDriver.java:521)  
at java.sql.DriverManager.getConnection(DriverManager.java:582)  
at java.sql.DriverManager.getConnection(DriverManager.java:185)  
at com.cloudera.sqoop.mapreduce.db.DBConfiguration.getConnection(DBConfiguration.java:152)  
at com.cloudera.sqoop.mapreduce.db.DBInputFormat.getConnection(DBInputFormat.java:184)  
... 10 more  
Caused by: java.net.SocketException: Connection reset  
at java.net.SocketOutputStream.socketWrite(SocketOutputStream.java:96)  
at java.net.SocketOutputStream.write(SocketOutputStream.java:136)  
at oracle.net.ns.DataPacket.send(DataPacket.java:199)  
at oracle.net.ns.NetOutputStream.flush(NetOutputStream.java:211)  
at oracle.net.ns.NetInputStream.getNextPacket(NetInputStream.java:227)  
at oracle.net.ns.NetInputStream.read(NetInputStream.java:175)  
at oracle.net.ns.NetInputStream.read(NetInputStream.java:100)  
at oracle.net.ns.NetInputStream.read(NetInputStream.java:85)  
at oracle.jdbc.driver.T4CSocketInputStreamWrapper.readNextPacket(T4CSocketInputStreamWrapper.java:123)  
at oracle.jdbc.driver.T4CSocketInputStreamWrapper.read(T4CSocketInputStreamWrapper.java:79)  
at oracle.jdbc.driver.T4CMAREngine.unmarshalUB1(T4CMAREngine.java:1122)  
at oracle.jdbc.driver.T4CMAREngine.unmarshalSB1(T4CMAREngine.java:1099)  
at oracle.jdbc.driver.T4CTTIfun.receive(T4CTTIfun.java:288)  
at oracle.jdbc.driver.T4CTTIfun.doRPC(T4CTTIfun.java:191)  
at oracle.jdbc.driver.T4CTTIoauthenticate.doOAUTH(T4CTTIoauthenticate.java:366)  
at oracle.jdbc.driver.T4CTTIoauthenticate.doOAUTH(T4CTTIoauthenticate.java:752)  
at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:366)  
... 18 more

해결방법: 이 문제는 맵태스크가 실행되는 호스트 상의 고속 난수 생성 장치가 없기 때문에 주로 발생한다. 전형적인 리눅스 시스템 상에서 java.security 파일의 다음 속성을 설정하여 다루어질 수 있다.:

java.security.egd=file:/dev/../dev/urandom

java.security 파일은 $JAVA\_HOME/jre/lib/security 디렉토리 아래에서 찾을 수 있다. 다른 방법으로서 이 속성은 아래와 같은 방법으로 명령줄에 명시될 수 있다.:

-D mapred.child.java.opts="\-Djava.security.egd=file:/dev/../dev/urandom"+

***23.2.2. Oracle: Case-Sensitive 카탈로그 쿼리 에러***

문제: Oracle과 함께 쓰다보면 Sqoop이 컬럼명을 이해할 수 없는 에러를 마주칠 수 있다. 이것은 Sqoop이 Oracle을 위하여 사용하는 카탈로그 쿼리가 사용자명과 테이블명의 올바른 대소문자 표기를 요구하기 때문에 발생한다.

--hive-import를 사용할 때, NullPointerException이 발생하는 예를 들어보자.:

1/09/21 17:18:49 INFO manager.OracleManager: Time zone has been set to  
GMT  
11/09/21 17:18:49 DEBUG manager.SqlManager: Using fetchSize for next  
query: 1000  
11/09/21 17:18:49 INFO manager.SqlManager: Executing SQL statement:  
SELECT t.\* FROM addlabel\_pris t WHERE 1=0  
11/09/21 17:18:49 DEBUG manager.OracleManager$ConnCache: Caching  
released connection for jdbc:oracle:thin:  
11/09/21 17:18:49 ERROR sqoop.Sqoop: Got exception running Sqoop:  
java.lang.NullPointerException  
java.lang.NullPointerException  
at com.cloudera.sqoop.hive.TableDefWriter.getCreateTableStmt(TableDefWriter.java:148)  
at com.cloudera.sqoop.hive.HiveImport.importTable(HiveImport.java:187)  
at com.cloudera.sqoop.tool.ImportTool.importTable(ImportTool.java:362)  
at com.cloudera.sqoop.tool.ImportTool.run(ImportTool.java:423)  
at com.cloudera.sqoop.Sqoop.run(Sqoop.java:144)  
at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:65)  
at com.cloudera.sqoop.Sqoop.runSqoop(Sqoop.java:180)  
at com.cloudera.sqoop.Sqoop.runTool(Sqoop.java:219)  
at com.cloudera.sqoop.Sqoop.runTool(Sqoop.java:228)  
at com.cloudera.sqoop.Sqoop.main(Sqoop.java:237)

**해결방법:**

1. Sqoop이 접속할 때 사용하는 사용자명을 (사용자명이 인용문자 내에 대소문자가 혼합된 형태가 아니라면) 대문자로 명시하라.
2. 작업 대상의 테이블명을 (테이블명이 인용문자 내에 대소문자가 혼합된 형태가 아니라면) 대문자로 명시하라.

***23.2.3. MySQL: 연결 실패***

문제: MySQL 테이블을 Sqoop으로 가져오는 데에 있어서 만약 당신이 네트워크에 걸쳐서 MySQL 데이터베이스에 접근하기 위해 필요한 권한이 없다면, 아래의 연결 실패를 겪게 될 것이다.

Caused by: com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

해결방법: 첫번째로 Sqoop을 실행하고 있는 노드로부터 데이터베이스로 연결할 수 있는지를 검증하라.:

$ mysql --host=<IP 주소> --database=test --user=<사용자명> --password=<비밀번호>

이 방법이 통한다면, 클라이언트 네트워크 설정이나 보안/인증 설정에 있어서의 문제들을 배제시킨다.

/etc/my.cnf 파일에 서버의 네트워크 포트를 추가하라.:

[mysqld]  
port = xxxx

Sqoop을 통하여 연결하기 위한 사용자 계정을 설정하라. 네트워크에 걸쳐 해당 데이터베이스에 접근할 수 있도록 사용자에게 권한을 허용하라.: (1.) 루트 계정으로 MySQL에 로그인한다. mysql -u root -p<ThisIsMyPassword>. (2.) 다음의 명령을 입력한다.:

mysql> grant all privileges on test.\* to 'testuser'@'%' identified by 'testpassword'

이렇게 하면 테스트사용자가 어떤 IP주소에서도 MySQL 서버에 접근할 수 있도록 하는 것을 알아둔다. 이 방법이 통할지라도 운영환경에서는 권고되지 않는다. 설치 토폴로지에 기반하여 필요한 권한을 허용하기 위하여 DBA와 상의해보기를 권고한다.

만약 데이터베이스 서버의 IP주소가 변경된다면, 그 IP주소가 서버 내에서 정적인 호스트명과 바인딩되어 있지 않은 경우 Sqoop으로 전달되는 커넥트 문자열도 변경될 필요가 있다.