■ Hive QL 함수

함 수	내 용
COUNT(1), COUNT(*)	전체 데이터 건수 반환
COUNT(DISTINCT 칼럼)	유일한 칼럼값의 건수 반환
SUM(칼럼)	칼럼값의 합계를 반환
SUM(DISTINCT 칼럼)	칼럼값의 합계를 반환
AVG(칼럼)	칼럼값의 평균 반환
AVG(DISTINCT 칼럼)	유일한 칼럼값의 평균을 반환
MAX(칼럼)	칼럼의 최댓값을 반환
MIN(칼럼)	칼럼의 최솟값을 반환

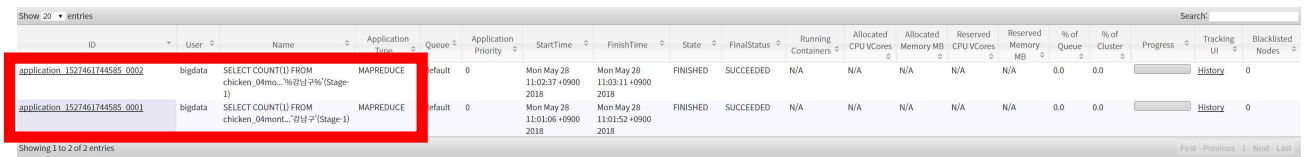
1. COUNT 함수를 이용해 강남구 전체에서 통화한 건수를 구함

```
SELECT COUNT(calls) AS call_counting
FROM chicken_04month
WHERE district LIKE '%강남구%';
```

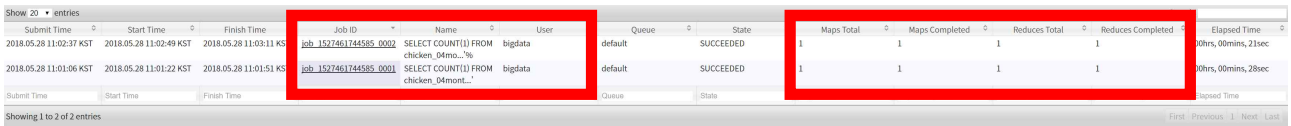
- 결과 확인

```
+----------------+
| call_counting  |
+----------------+
| 2699           |
+----------------+
1 row selected (40.489 seconds)
```

- 얀 실행 확인(http://server01:8088)



- 맵리듀스 잡 실행 확인(http://server01:19888)

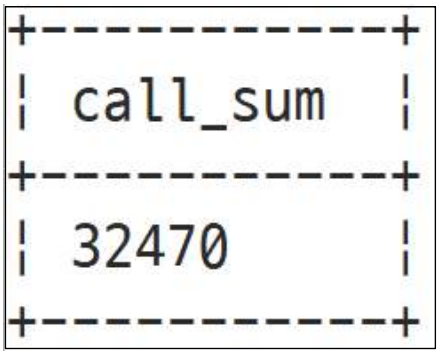


- 문제: 치킨집을 이용한 관악구 20대(age)의 통화건수를 구하세요.

2. SUM 함수를 이용해 강남구 전체에서 통화량을 구함

```
SELECT sum(calls) AS call_sum
FROM chicken_04month
WHERE district LIKE '%강남구%';
```

- 결과 확인



```
+------------+
| call_sum   |
+------------+
| 32470      |
+------------+
```

- 문제: 강남구의 60대이상 연령대(age)의 치킨집 통화량을 구하세요.

2. Group by

- 강남구의 각 연령별 치킨 주문건수

```
SELECT age, district, COUNT(*) AS chicken_count
FROM chicken_04month
WHERE district LIKE '%강남구%'
GROUP BY age, district;
```

```
+--------+----------+----------------+
|  age   | district | chicken_count  |
+--------+----------+----------------+
| 10대   | 강남구       | 370
| 20대   | 강남구       | 511
| 30대   | 강남구       | 522
| 40대   | 강남구       | 530
| 50대   | 강남구       | 432
| 60대이상 | 강남구       | 334
+--------+----------+----------------+
6 rows selected (39.989 seconds)
```

- 문제: 각 시군구(district)별 10대의 치킨 주문 통화량을 구하세요.

답 :

```
SELECT age, district, sum(calls) AS chicken_count
FROM chicken_04month
```

```
WHERE age LIKE '%10대%'
GROUP BY age, district;
```

```
SELECT gender, dayOfWeek, COUNT(dayOfWeek) AS chicken_count
FROM chicken_04month
GROUP BY gender, dayOfWeek
order by chicken_count DESC;
```

- AVG 함수를 이용한 요일 별 평균 치킨 주문량

```
SELECT dayOfWeek, AVG(calls) AS chicken_avg
FROM chicken_04month
GROUP BY dayOfWeek;
```

```
+------------+---------------------+
| dayofweek  |     chicken_avg     |
+------------+---------------------+
| 금         | 13.785620176924525  |
| 목         | 11.42471126607104   |
| 수         | 10.97646536412078   |
| 월         | 10.28333648571969   |
| 일         | 14.703012442698101  |
| 토         | 15.0076171875       |
| 화         | 10.665135507579237  |
+------------+---------------------+
7 rows selected (37.855 seconds)
```

- Double 형태로 출력됨
- 평균 주문량은 토요일이 가장 많음

- 시군구(district)별 가장 많은 통화 건수는?(MAX 함수 사용)

3. having도 가능

- 시군구

■ 데이터 정렬

1. ORDER BY

○ 전체 정렬

○ 맵리듀스 잡에서 하나의 리듀서만 실행

○ 정렬 대상 데이터가 클 경우 성능이 느려짐

- 가장 많은 전화 주문

```
SELECT *
FROM chicken_04month
ORDER BY calls DESC
LIMIT 5;
```

```
20180428    토    여    20대    서울특별시    영등포구    여의도동    치킨    242
20180421    토    여    20대    서울특별시    영등포구    여의도동    치킨    239
20180429    일    여    20대    서울특별시    영등포구    여의도동    치킨    236
20180407    토    여    40대    서울특별시    관악구    신림동    치킨    228
20180414    토    남    30대    서울특별시    관악구    신림동    치킨    227
```

2. SORT BY

○ 전체 정렬 대신 질의 성능을 높임

○ 여러개의 리듀서를 실행해 각 리듀서의 출력 결과를 정렬

```
SELECT *
FROM chicken_04month
SORT BY calls DESC
LIMIT 5;
```

```
20180428    토    여    20대    서울특별시    영등포구    여의도동    치킨    242
20180421    토    여    20대    서울특별시    영등포구    여의도동    치킨    239
20180429    일    여    20대    서울특별시    영등포구    여의도동    치킨    236
20180407    토    여    40대    서울특별시    관악구    신림동    치킨    228
20180414    토    남    30대    서울특별시    관악구    신림동    치킨    227
```

3. DISTRIBUTED BY

- 리듀스로 보낼 데이터 분류 기준

- 같은 키를 가진 레코드가 같은 리듀서로 보내짐

- SORT BY와 함께 사용할 경우 각 리듀서는 키가 중복되지 않고 정렬 수행

```
SELECT *
FROM chicken_04month c
DISTRIBUTE BY c.calls
SORT BY c.calls DESC
LIMIT 5;
```

■ 조인

○ 맵리듀스로 조인을 하면 수십 줄 이상의 클래스를 작성해야함

○ 하이브를 이용하면 간단하게 조인이 가능

○ 제약 사항이 있음

- EQ 조인만 지원
- EQ 조인: 두 테이블의 동일성을 비교 한 후 그 결과를 기준으로 조인
- 조인 서술자로 등호(=)만 사용
- FROM 절에 테이블 하나만 지정할 수 있음
- ON 키워드를 이용해 조인 처리

1. 내부 조인(INNER JOIN)

- film_rating과 films 두 개의 테이블을 내부 조인

- 영화별 평점 평균

```
SELECT * FROM(
SELECT f.filmTitle as title, AVG(fr.rating) as rating
FROM film_rating fr
JOIN films f ON (fr.film = f.filmNum)
GROUP BY f.filmTitle) as mv
order by mv.rating DESC;
```

```
인터스텔라      4.357142857142857
변호인  4.285714285714286
어벤져스        4.205882352941177
곡성    4.166666666666667
킹스맨: 시크릿 에이전트 4.166666666666667
캡틴 아메리카:시빌 워   4.115384615384615
다크 나이트 라이즈      4.107142857142857
겨울왕국        4.066666666666666
왕의남자        4.041666666666667
아바타  4.033333333333333
광해    4.0
미션임파서블:고스트프로토콜     4.0
괴물    3.9615384615384617
아이언맨3       3.9285714285714284
설국열차        3.9
트랜스포머      3.857142857142857
어벤져스:에이지오브울트론       3.8333333333333335
도둑들  3.8125
써니    3.730769230769231
```

2. 외부 조인

- 외부조인을 테스트 하기 위해 로컬 파일 films.csv 에서 첫번째 줄을 삭제
- 첫번째 줄에는 '명량' 영화가 있음

```
$ sed -e '1d' films.csv > films_new.csv
```

- 하이브 홈디렉토리 이동 후 하이브 접속

```
[bigdata@server01 example_data]$ cd /home/bigdata/apache-hive-2.3.3-bin
[bigdata@server01 apache-hive-2.3.3-bin]$ ./bin/hive
```

- film_rating2 테이블 생성

```
CREATE TABLE films2(filmNum int, filmTitle String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

- 파일 업로드

```
LOAD DATA local INPATH '/home/bigdata/example_data/films_new.csv'
OVERWRITE INTO TABLE films2;
```

- 외부 조인

```
SELECT fr.film, f.filmNum
FROM film_rating fr
LEFT OUTER JOIN films2 f ON (fr.film = f.filmNum)
WHERE fr.film = 1
limit 5;
```

```
1       NULL
1       NULL
1       NULL
1       NULL
1       NULL
Time taken: 64.219 seconds, Fetched: 5 row(s)
```

■ 쿼리 결과 파일/테이블로 저장

○ 쿼리 결과를 로컬에 파일로 저장

- LOCAL을 붙이지 않으면 HDFS에 저장됨
- 만약 저장 디렉토리에 다른 파일이 있으면 다 지워짐

- 강남구의 연령대별 치킨 주문 회수를 파일로 저장

```
INSERT OVERWRITE LOCAL DIRECTORY '/home/bigdata/hive_data/data'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
SELECT age, district, COUNT(*) AS chicken_count
FROM chicken_04month
WHERE district LIKE '%강남구%'
GROUP BY age, district;
```

- 파일이 저장된 경로로 이동

```
$ cd /home/bigdata/hive_data/data
```

- 000000_0로 파일이 생성되어있음

```
[bigdata@server01 data]$ cat 000000_0
```

```
[bigdata@server01 data]$ cat 000000_0
10대    강남구  370
20대    강남구  511
30대    강남구  522
40대    강남구  530
50대    강남구  432
60대이상        강남구  334
```

○ 테이블 생성 + 쿼리 결과 데이터 로딩

• 테이블 생성과 동시에 쿼리의 결과를 데이터로 저장

- 강남구의 연령대별 치킨 주문 회수를 하이브 테이블로 저장

```
create table age_chicken_calls2
as SELECT age, district, COUNT(*) AS chicken_count
FROM chicken_04month
WHERE district LIKE '%강남구%'
GROUP BY age, district;
```

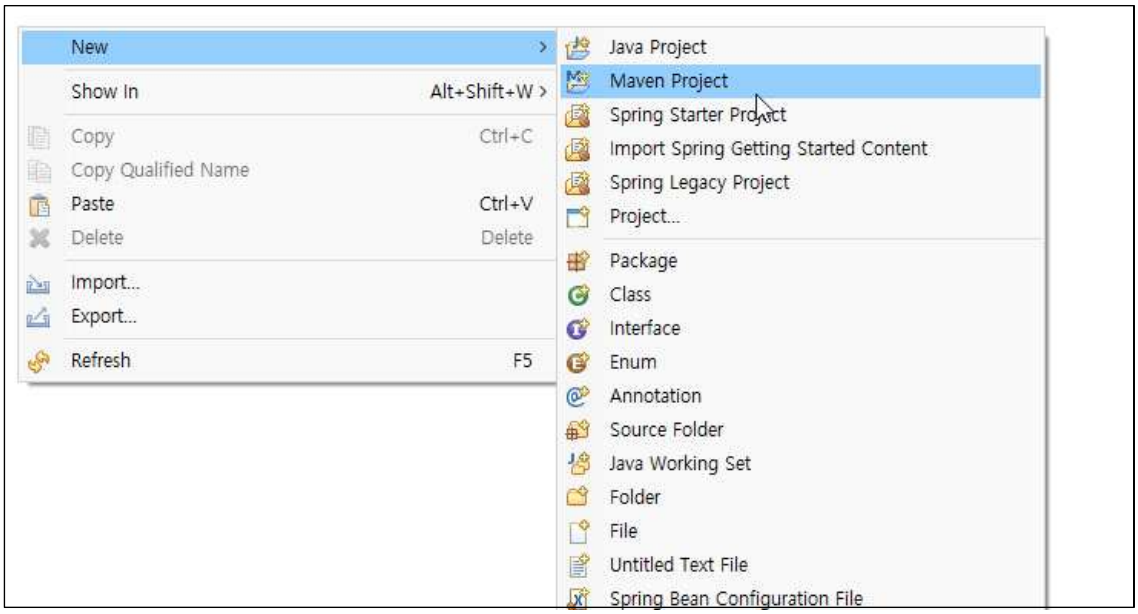
- 테이블 확인

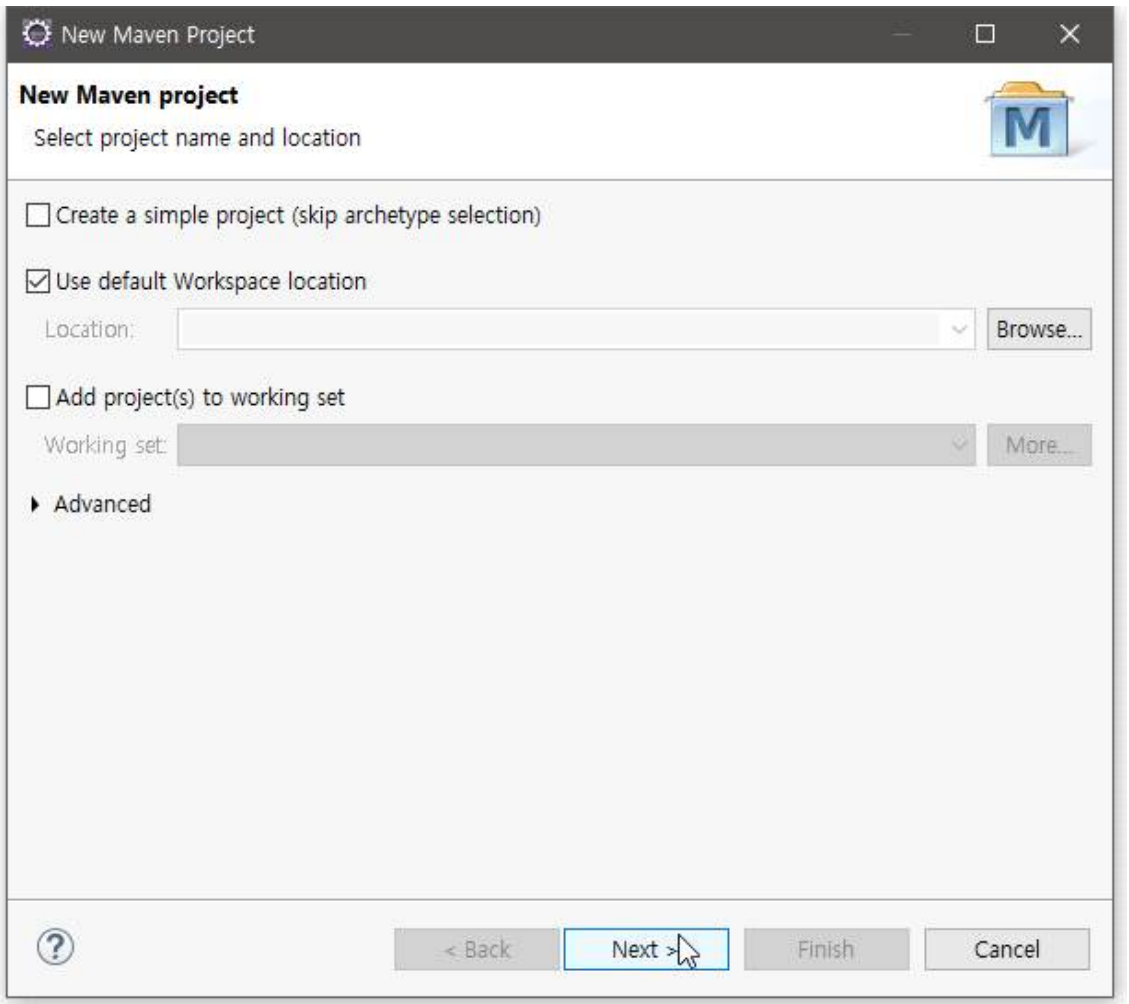
```
hive> select * from age_chicken_calls;
```

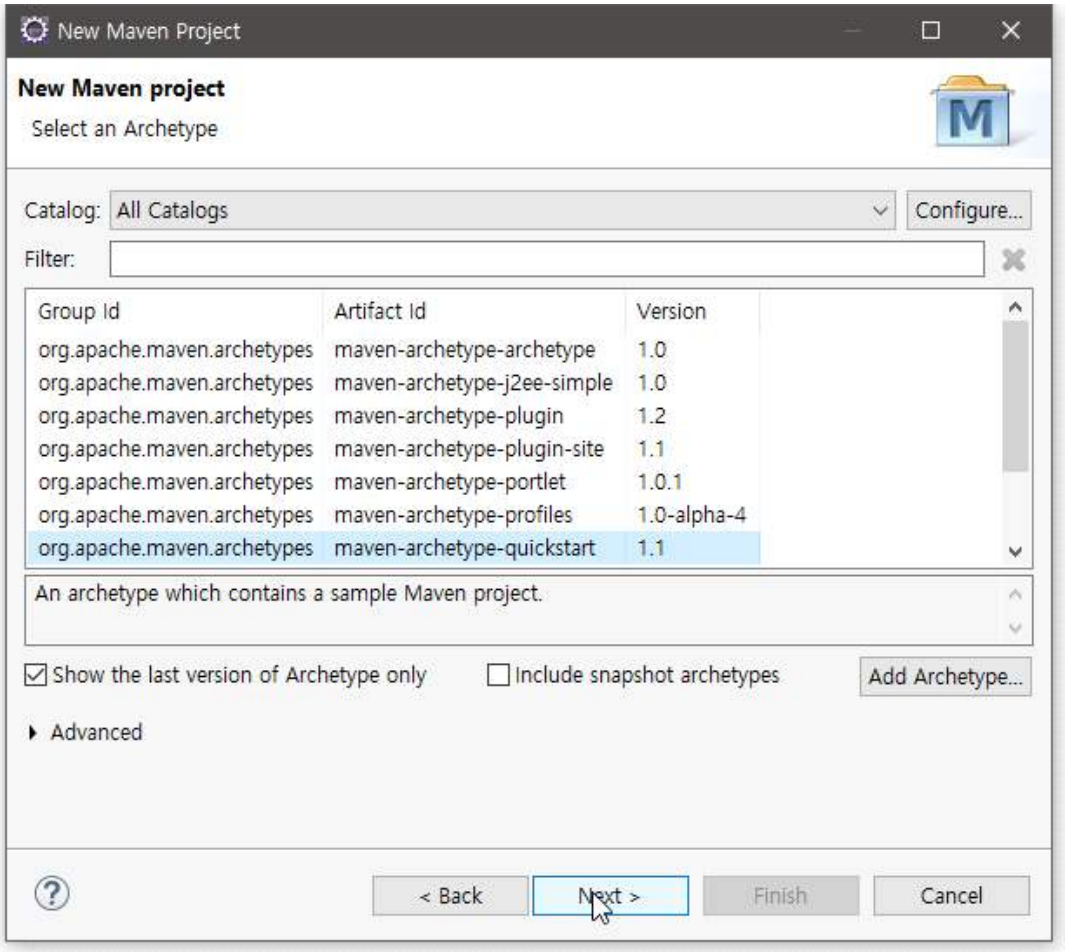
```
hive> select * from age_chicken_calls;
OK
10대    강남구  370
20대    강남구  511
30대    강남구  522
40대    강남구  530
50대    강남구  432
60대이상        강남구  334
Time taken: 0.331 seconds, Fetched: 6 row(s)
```

■ Hive 연동 자바 애플리케이션 개발

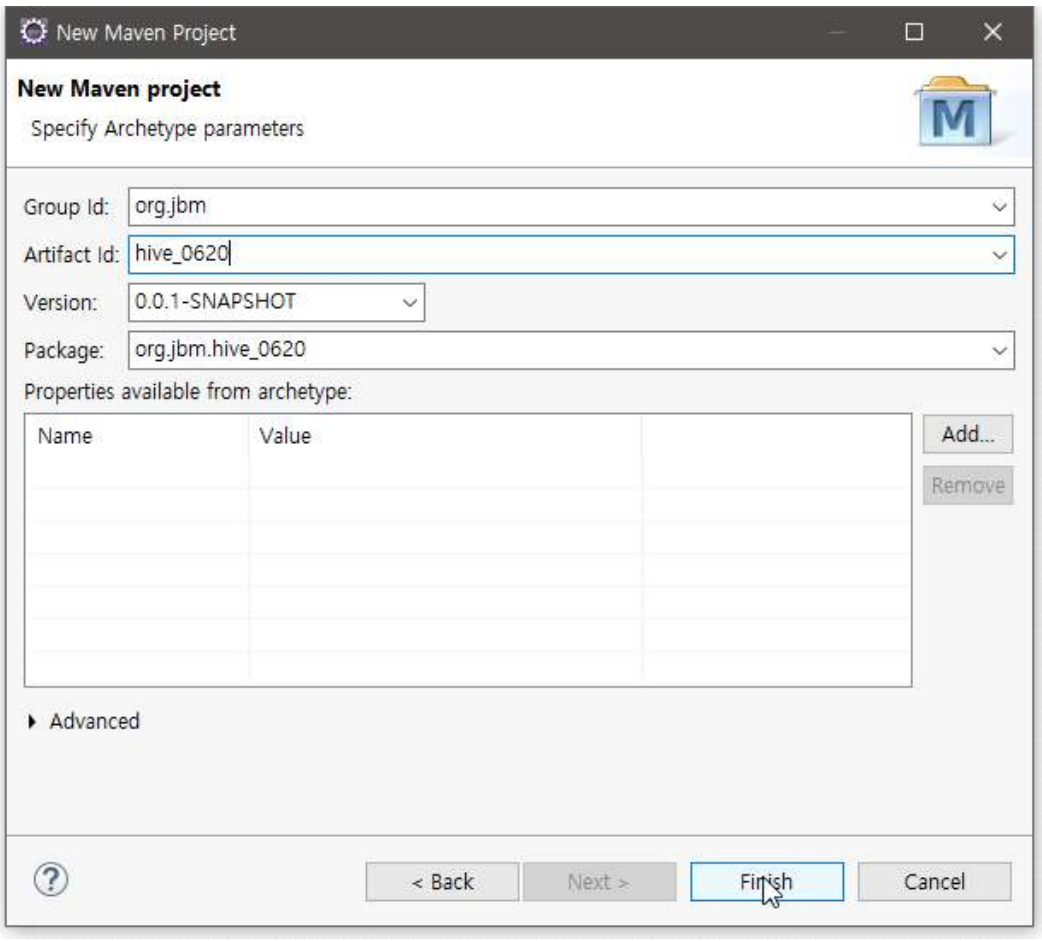
1) maven project로



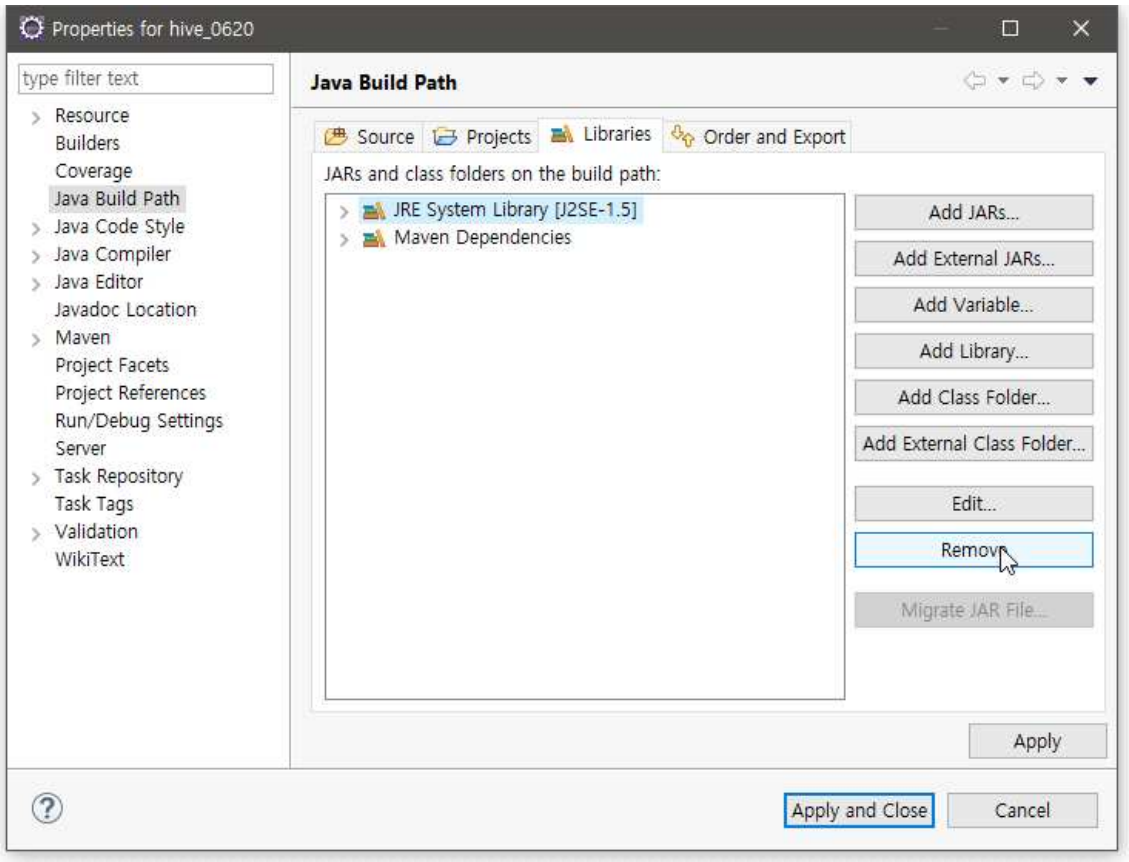


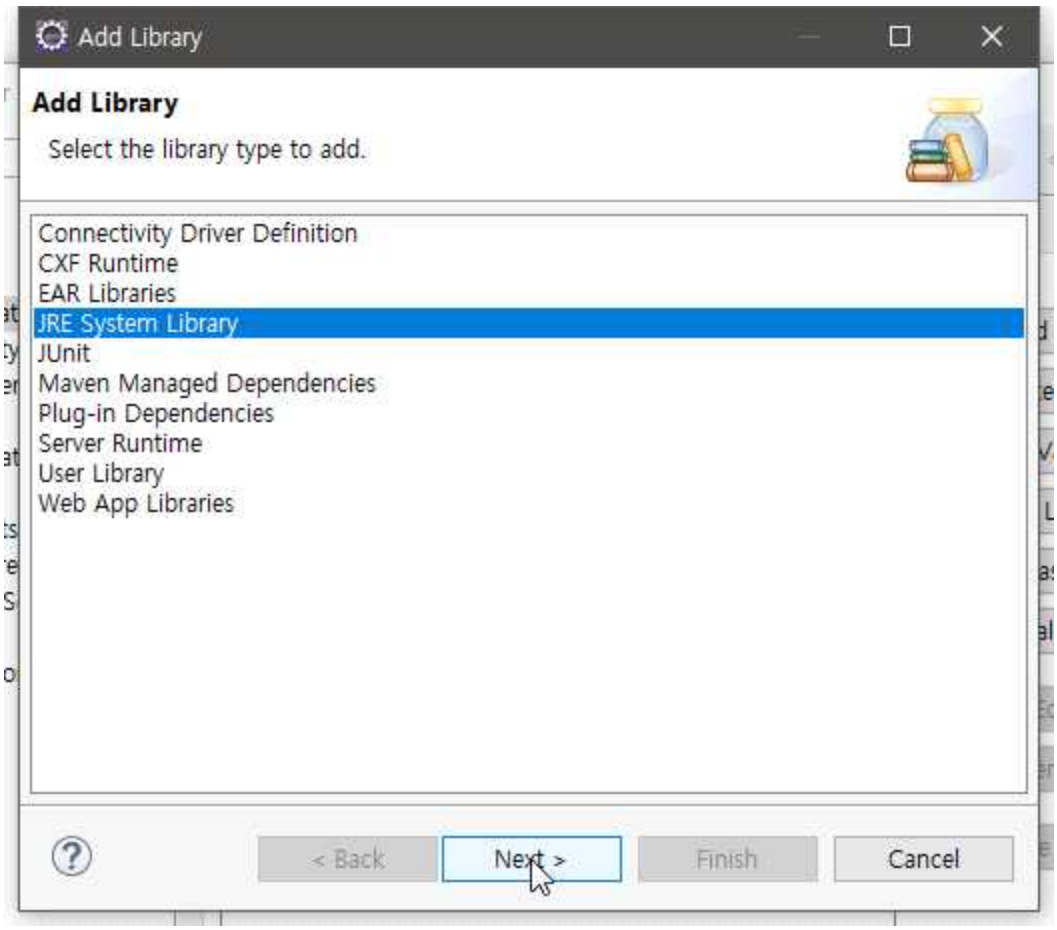


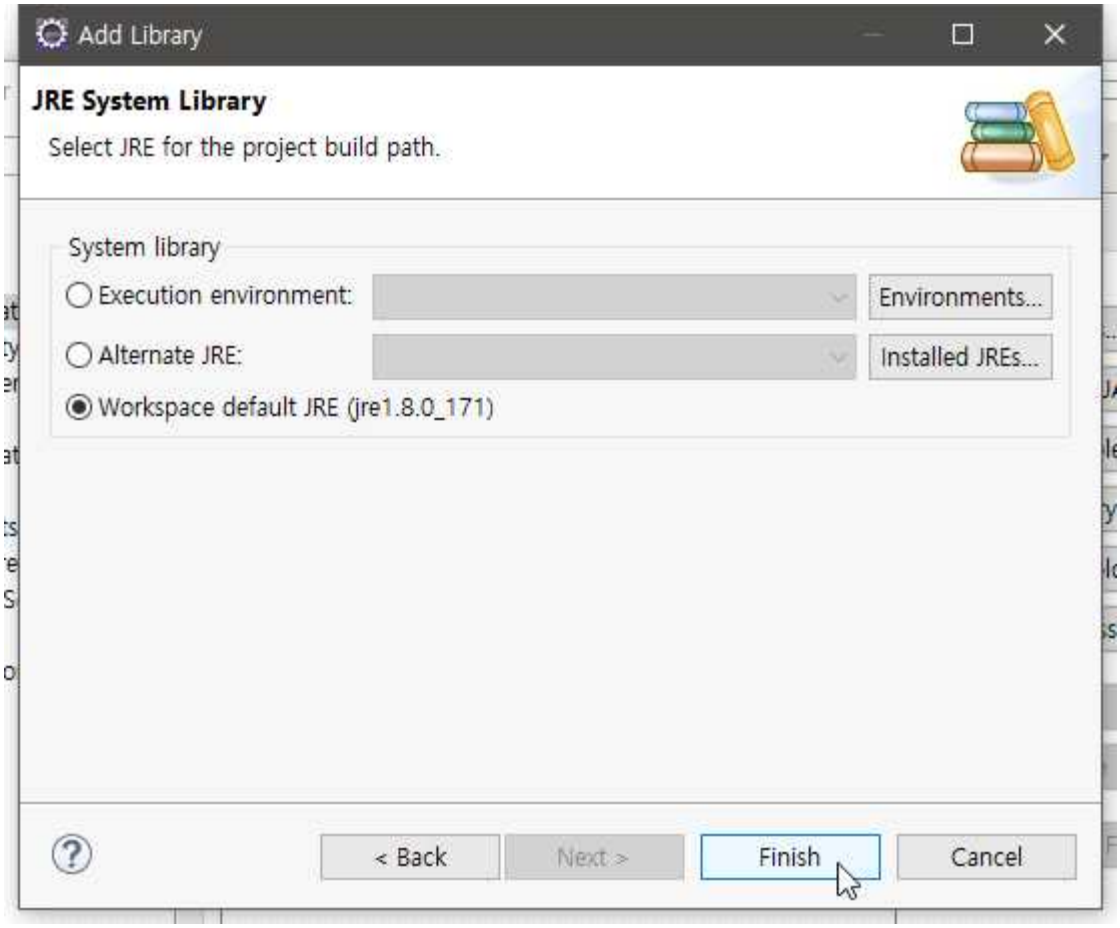
- artifact id를 hive_0620으로

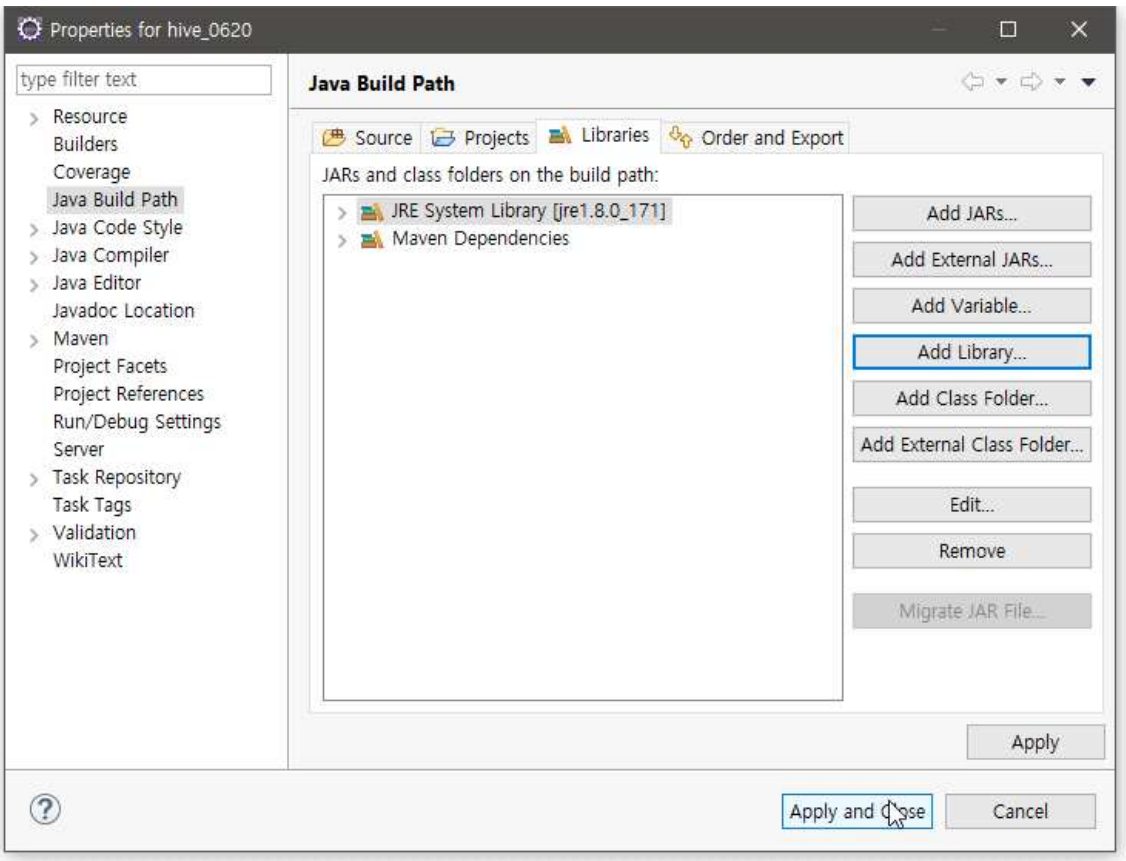


- 메이븐은 기본 자바버전 5버전으로 되어있기 때문에 8로 jre를 변경






Add Library
JRE System Library
Select JRE for the project build path.
System library
Execution environment:
Environments...
Alternate JRE:
Installed JREs...
Workspace default JRE (jre1.8.0_171)
< Back
Next >
Finish
Cancel

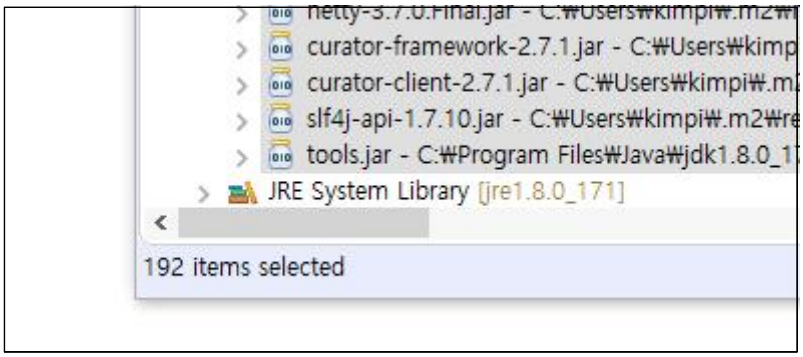

Properties for hive_0620
type filter text
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Maven
Project Facets
Project References
Run/Debug Settings
Server
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
JRE System Library [jre1.8.0_171]
Maven Dependencies
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Apply and Close
Cancel

- pom.xml을 수정

```
<profiles>
	<profile>
		<id>windows-profile</id>
		<activation>
			<activeByDefault>true</activeByDefault>
			<file>
				<exists>${JAVA_HOME}/lib/tools.jar</exists>
			</file>
		</activation>
		<properties>
			<toolsjar>${JAVA_HOME}/lib/tools.jar</toolsjar>
		</properties>
	</profile>
</profiles>

<dependencies>
	<!-- https://mvnrepository.com/artifact/org.apache.hive/hive-jdbc -->
	<dependency>
		<groupId>org.apache.hive</groupId>
		<artifactId>hive-jdbc</artifactId>
		<version>2.3.3</version>
	</dependency>
	<dependency>
		<groupId>jdk.tools</groupId>
		<artifactId>jdk.tools</artifactId>
		<version>jdk1.8.0</version>
		<scope>system</scope>
		<systemPath>${toolsjar}</systemPath>
	</dependency>
</dependencies>
```

- 192개의 라이브러리가 라이브러리 등록



2) 일반 JDBC 프로그래밍으로 가능

```
package org.jbm.hive_0620;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class HiveApp {
        public static void main(String[] args) throws Exception {

                Class.forName("org.apache.hive.jdbc.HiveDriver");

                Connection con = DriverManager.getConnection("jdbc:hive2://192.168.56.101:10000");
                Statement stmt = con.createStatement();
                // show tables
                String sql = "select count(*) from chickens";
                System.out.println("Running: " + sql);

                long start = System.currentTimeMillis();

                ResultSet res = stmt.executeQuery(sql);
                while (res.next()) {
                        System.out.println("행의 갯수 : " +res.getString(1)+"개");
                }

                long time = System.currentTimeMillis() - start;

                System.out.println("수행시간 : "+time + "ms");

        }
}
```

- 결과

```
Running: select count(*) from chickens
행의 갯수 : 71149개
수행시간 : 46311ms
```

- ex) 같은 구문을 스파크로 수행하면

```
18/06/20 02:54:47 INFO DAGScheduler: Jo
행의 갯수 : 71149개
수행시간 : 1183ms
18/06/20 02:54:47 INFO SparkContext: In
```

```
package org.jbm.hive_0620;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class HiveApp2 {

	public static void main(String[] args) throws Exception{

		Class.forName("org.apache.hive.jdbc.HiveDriver");

		Connection con = DriverManager.getConnection("jdbc:hive2://192.168.56.101:10000");
		Statement stmt = con.createStatement();
		// show tables
		String sql = "select calldate, dayofweek,gender,age, city, calls from chickens limit
10000,10";

		ResultSet rs = stmt.executeQuery(sql);

		while(rs.next()) {

			System.out.println("날짜:"+rs.getString(1)+" / "+rs.getString(2)+"요일 / 성별 :
" + rs.getString(3)+
				" / 연령대 : " + rs.getString(4) + " / 시도 : " +
rs.getString(5) + " / 통화수 : " + rs.getInt(6));

		}//while end

	}

}
```

- 결과

```
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 30대 / 시도 : 서울특별시 / 통화수 : 30
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 40대 / 시도 : 서울특별시 / 통화수 : 44
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 40대 / 시도 : 서울특별시 / 통화수 : 5
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 40대 / 시도 : 서울특별시 / 통화수 : 6
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 40대 / 시도 : 서울특별시 / 통화수 : 5
날짜:20180409 / 월요일 / 성별 : 남 / 연령대 : 40대 / 시도 : 서울특별시 / 통화수 : 18
날짜:20180409 / 월요일 / 성별 : 남 / 연령대 : 20대 / 시도 : 서울특별시 / 통화수 : 5
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 40대 / 시도 : 서울특별시 / 통화수 : 5
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 50대 / 시도 : 서울특별시 / 통화수 : 15
날짜:20180409 / 월요일 / 성별 : 여 / 연령대 : 50대 / 시도 : 서울특별시 / 통화수 : 5
```