**DT 기반 과제를 수행하기 위한 HiveQL**

**(본문)**

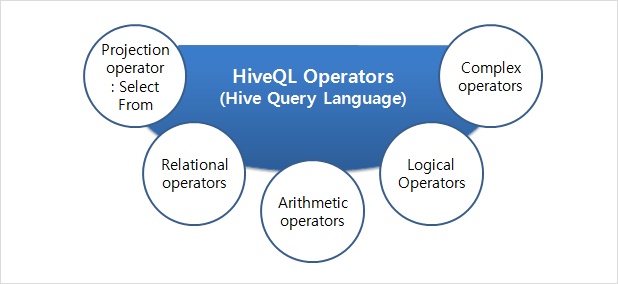
**HIVE : SQL for Hadoop!**

: JAVA는 모르지만 SQL은 잘 아는 개발자, 분석가를 위해 Hadoop MapReduce를 쓸 수 있도록 Facebook이 처음 개발하고 나중에 Apache Open Source 프로젝트로 더욱 발전한 툴입니다.

(대용량 데이터 전처리하는데 HIVE를 사용하면 생산성이 올라갑니다.)

* **Hadoop** : 하둡은 대량의 자료를 처리할 수 있는 대규모 컴퓨터 클러스터에서 동작하는 분산 애플리케이션을 지원하는 오픈 자바 소프트웨어 프레임워크 입니다.
* **MapReduce** : 대용량 데이터를 분산 처리하기 위한 목적으로 개발된 프로그래밍 모델입니다. 임의의 순서로 정렬된 데이터를 분산 처리(Map)하고 이를 다시 합치(Reduce)는 과정을 거칩니다.
* **Apache** : 월드와이드웹(WWW ; World Wide Web) 서버용 소프트웨어입니다.

**HiveQL Operations : 프로젝션 연산자, 비교 연산자, 산술 연산자, 논리 연산자, 복합 연산자**



1. **프로젝션 연산자**

아래와 같은 테이블이 있다고 하자!!

mydb.my\_table (

var\_1 STRING COMMENT ‘var\_1 is this’,

var\_2 FLOAT COMMMENT ‘var\_2 is that’,

var\_3 INT COMMENT ‘var\_3 is those’,

var\_4 ARRAY<STRING> COMMENT ‘array is….’,

var\_5 MAP <STRING, FLOAT> COMMENT ‘map is…’,

var\_6 STRUCT<aa:STRING, bb:FLOAT, cc:INT>COMMENT ‘ structure is…’

~~(불라불라)~~ )

* SELECT var\_nm FROM table 에서 데이터 형태에 따라 데이터명을 달리하여 원하는 형태의 데이터를 my\_table로부터 얻을 수 있습니다.

1. **비교 연산자(Relational Operators)**

|  |  |  |
| --- | --- | --- |
| **연산자** | **데이터형** | **설명** |
| A = B | all primitive types | A와 B가 같으면 TRUE, 그렇지 않으면 FALSE |
| A != B | all primitive types | A와 B가 같지 않으면 TRUE, 그렇지 않으면 FALSE |
| A < B | all primitive types | A가 B보다 작으면 TRUE, 그렇지 않으면 FALSE |
| A<=B | all primitive types | A가 B보다 작거나 같으면 TRUE, 그렇지 않으면 FALSE |
| A > B | all primitive types | A가 B보다 크면 TRUE, 그렇지 않으면 FALSE |
| A>=B | all primitive types | A가 B보다 크거나 같으면 TRUE, 그렇지 않으면 FALSE |
| A IS NULL | all types | A가 NULL 이면 TRUE, 그렇지 않으면 FALSE |
| A IS NOT NULL | all types | A가 NULL 아니면 TRUE, 그렇지 않으면 FALSE |
| A LIKE B | Strings | Sting 패턴 A가 B와 동일하면 TRUE, 그렇지 않으면 FALSE |
| A RLIKE B | Strings | A 또는 B가 NULL이면 NULL, A의 어떤 substring이라도 B와 동일하면 TRUE, 그렇지 않으면 FALSE |

Ex.)

**SELECT** \* **FROM** my\_table **WHERE** var\_1 = ‘HongGu’

**SELECT** \* **FROM** my\_table **WHERE** var\_3 >= 100

**SELECT** \* **FROM** my\_table **WHERE BETWEEN** var\_2 >= 0.1 **AND** var\_2 <= 2.0

1. **산술 연산자(Arithmetic Operators)**

|  |  |  |
| --- | --- | --- |
| **연산자** | **데이터형** | **설명** |
| A + B | all number types | 더하기 (adding A and B) |
| A – B | all number types | 빼기 (substracting B from A) |
| A \* B | all number types | 곱하기 (multiplying A and B) |
| A / B | all number types | 나누기 (dividing B from A) |
| A % B | all number types | 나눈 후의 나머지 반환  (reminder resulting from diving A by B) |
| A & B | all number types | 비트 AND 연산 (bitwise AND of A and B) |
| A | B | all number types | 비트 OR 연산 (bitwise OR of A and B) |
| A ^ B | all number types | 비트 XOR 연산 (bitwise XOR of A and B) |
| ~A | all number types | 비트 NOT 연산 (bitwise NOT of A) |

Ex.)

**SELECT** (var\_1 + var\_2) **AS** var\_1\_2\_sum **FROM** my\_table;

1. **논리 연산자(Logical Operators)**

|  |  |  |
| --- | --- | --- |
| **연산자** | **데이터형** | **설명** |
| A AND B  (A && B) | boolean | A와 B 모두 TRUE이면 TRUE, 그렇지 않으면 FALSE |
| A OR B  (A || B) | boolean | A 또는 B, 또는 A와 B가 모두 TRUE이면 TRUE, 그렇지 않으면 FALSE |
| NOT A  (!A) | boolean | A가 FALSE이면 TRUE, 그렇지 않으면 FALSE |

Ex.)

**SELECT** \* **FROM** my\_table **WHERE** var\_3 >= 100 && var\_2 <= 2.0

1. **복합 연산자(Complex Operators)**

|  |  |  |
| --- | --- | --- |
| **연산자** | **데이터형** | **설명** |
| A[n] | A : Array  n : int | Array A에서 n번째 값을 출력.  첫번째 값은 index 0 부터 시작함.  ( \* 참고 : R은 index 1 부터 시작함) |
| M[key] | M : Map<K, V>  Key has type K | Key에 해당하는 Value를 출력 |
| S.x | S : STRUCT | STRUCT의 x field를 출력 |

Ex.)

**SELECT** var\_1, var\_4[0] **FROM** my\_table; -- table에 있는 var\_4 데이터 형태가 배열

**SELECT** var\_1, var\_5[“key1”] **FROM** my\_table; -- table에 있는 var\_5 데이터 형태가 Map

**SELECT** var\_1, var\_6.aa **FROM** my\_table; -- table에 있는 var\_6 데이터 형태가 구조체

(\* 구조체 : type이 다른 변수들을 하나의 틀 안에 넣어 나타내는 표현. 즉 종류가 다른 여러 과일들을 담을 수 있는 과일바구니라고 생각하면 쉽습니다!)

**Hive Functions : 내장함수(Hive built-in functions), 수학함수(Hive arithmetic functions), 집계함수(Hive aggregate functions)**

: Hive에 내장되어 있는 함수들로 각각의 기능들은 다양합니다. 필요시 마다 아래 정리한 표 혹은 아래 사이트에서 필요한 함수를 찾아 사용하는 것을 추천합니다.

1. **내장 함수 (Hive Built-in functions)**

|  |  |  |
| --- | --- | --- |
| **반환 데이터형** | **내장 함수** | **설명** |
| BIGINT | round(double a) | double 데이터형 a의 반올림한 값을 BIGINT 데이터형으로 반환  예) round(4.8) = 5 |
| BIGINT | ceil(double a) | double 데이터형 a보다 크거나 같은 값 중 가장 작은 BIGINT 값을 반환  예) ceil(4.8) = 5 |
| DOUBLE | round(double a, N) | double 데이터형 a의 N개 소수점 이하의 값 까지를 반올림하여 double 데이터형으로 반환  예) round(4.816, 2) = 4.82 |
| DOUBLE | floor(double a) | double 데이터형 a보다 같거나 작은 값 가장 큰 BIGINT 값을 반환  예) floor(4.8) = 4 |
| DOUBLE | rand(),  rand(int seed) | 행에서 행으로 변하는 무작위 숫자를 반환 |
| string | concat(string A, string B, …) | string A와 string B를 연속으로 붙여서 반환 |
| string | concat\_ws  (sperator, string A, string B, …) | string A와 string B를 구분자(separator)를 두어서 연속으로 붙여서 반환 |
| string | substr(string A, int start) | string A를 시작 색인 지점부터 끝까지 잘라서 반환.   * 색인은 1부터 시작함.   예) substr(‘20180830’, 5) = ‘0830’ |
| string | Substr  (string A, int start, int length)) | string A를 시작 색인 지점부터 int length만큼 잘라서 반환.   * 색인은 1부터 시작함.   예) substr(‘20180830’, 5, 2) = ‘08’ |
| string | repeat(string A, n) | string A를 n회 반복하여 반환. |
| string | upper(string A)  or ucase(string A) | string A의 모든 문자를 대문자로 반환.  예) upper(‘Love’) = ‘LOVE’ |
| string | lower(string A)  or lcase(string A) | string A의 모든 문자를 소문자로 반환.  예) lower(‘Love’) = ‘love’ |
| string | trim(string A) | string A의 양끝의 공백을 제거하여 반환.  예) trim(‘\_\_Love\_\_’) = ‘Love’ |
| string | ltrim(string A) | string A의 왼쪽 시작부분의 공백을 제거하여 반환.  예) ltrim(‘\_\_Love\_\_’) = ‘Love\_\_’ |
| string | rtrim(string A) | string A의 오른쪽 끝부분의 공백을 제거하여 반환.  예) rtrim(‘\_\_Love\_\_’) = ‘\_\_Love’ |
| string | space(n) | n개의 공백을 반환. |
| string | regexp\_replace  (string A, string B, string C) | string A에서 string B의 부분 문자열(substring)과 일치하는 Java 정규표현식을 string C로 교체하여 반환.  예) regexp\_replace(‘Love’, ‘oe’, ‘i’) => returns ‘Livi’ |
| string | from\_unixtime  (int unixtime) | Unix 기준시간(1970-01-01 00:00:00 UTC)을 => 현재 시스템 시간대의 초단위 시간 표현식 포맷 (“1970-01-01 00:00:00”)으로 변환. |
| string | to\_date(string timestamp) | timestamp 문자열에서 년/월/날짜를 반환.  예) to\_date(“1970-01-01 00:00:00”) = “1970-01-01” |
| string | get\_json\_object  (string json\_string, string path) | json string으로부터 json 객체를 추출하며, 추출한 json string을 반환함. Json string이 유효하지 않으면 NULL을 반환함. |
| int | length(string A) | string A의 길이를 반환. |
| int | size(Map<K.V>) | Map<K.V> 의 요소 개수를 반환. |
| int | size(Array<T>) | Array<T> 의 요소 개수를 반환. |
| int | year(string date) | timestamp 문자열에서 년(year)을 반환.  예) year(“1970-01-01”) = 1970 |
| int | month(string date) | timestamp 문자열에서 월(month)을 반환.  예) month(“1970-01-01”) = 01 |
| Int | day(string date) | timestamp 문자열에서 일(day)을 반환.  예) day(“1970-01-02”) = 2 |
| ARRAY<string> | spilt(string A, pattern) | pattern이 나타날 때마다 string A를 분할하여 문자열의 배열을 반환. |
| ARRAY<ARRAY <string>> | sentences(string A, lang, locale) | string A를 단어의 배열로 이루어진 문장의 배열로 반환. lang, locale 옵션을 설정하지 않을 경우 기본 locale 사용. |
| MAP<string, string> | str\_to\_map  (string A, delim1, delim2) | delim1을 Key-Value 쌍의 구분자로 사용하고 delim2를 Key와 Value의 구분자로 사용하여 string A를 parsing 한 후에 Map을 생성함. |
| value of <type> | cast  (<expr> as <type>) | 표현<expr>을 <type>으로 형변환하며, 변환에 실패하면 NULL을 반환함.  예) cast(‘1’ as BIGINT) => 문자열 ‘1’을 BIGINT로 형변환 |
| BOOLEAN | in\_file  (string A, file B) | file B에 string A가 들어 있으면 TRUE를 반환. |

1. **수학 함수 (Hive Arithmetic functions)**

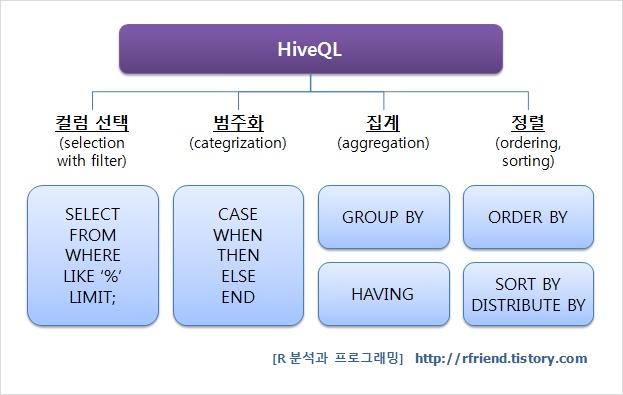
|  |  |  |
| --- | --- | --- |
| **반환 데이터형** | **수학 함수** | **설명** |
| DOUBLE | exp(double A) | double A의 지수(exponential) 함수값을 반환 |
| DOUBLE | ln(double A) | double A의 자연로그(natural logarithm)값을 반환. |
| DOUBLE | log10(double A) | double A의 상용로그(common logarithm)값을 반환. |
| DOUBLE | log2(double A) | double A의 밑을 2로 하는 로그값을 반환 |
| DOUBLE | log(base, double A) | double A의 밑을 base로 하는 로그값을 반환(base는 double 데이터형) |
| DOUBLE | pow(double A, p)  or  power(double A, p) | double A의 p 거듭제곱(power)한 값을 반환(p는 double 데이터형) |
| DOUBLE | sqrt(double A) | double A의 제곱근(square root)을 반환 |
| DOUBLE | abs(double A) | double A의 절대값(absolute value)을 DOUBLE 데이터형으로 반환. |
| DOUBLE | Pmod  (double 1, double 2) | double 1을 double 2로 모듈러 연산 후 양수 DOUBLE 값을 반환. |
| DOUBLE | sin(double A) | double A의 sine 값을 DOUBLE 데이터형의 radian 값으로 반환. |
| DOUBLE | asin(double A) | double A의 arc-sine 값을 DOUBLE 데이터형의 radian 값으로 반환 |
| DOUBLE | cos(double A) | double A의 cosine 값을 DOUBLE 데이터형의 radian 값으로 반환 |
| DOUBLE | acos(double A) | double A의 arc-cosine 값을 DOUBLE 데이터형의 radian 값으로 반환 |
| DOUBLE | tan(double A) | double A의 tangent 값을 DOUBLE 데이터형의 radian 값으로 반환 |
| DOUBLE | atan(double A) | double A의 arc-tangent 값을 DOUBLE 데이터형의 radian 값으로 반환 |
| DOUBLE | degrees(double A) | double A radian 값을 각도로 변환하여 반환 |
| DOUBLE | radians(double A) | double A 각도 값을 radian 값으로 변환하여 반환 |
| DOUBLE | positive(double A) | double A의 양의 DOUBLE 데이터형 값을 반환 |
| DOUBLE | negative(double A) | double A의 음의 DOUBLE 데이터형 값을 반환 |
| DOUBLE | r() | 지수형 상수값 2.7182818~을 DOUBLE 데이터형으로 반환 |
| DOUBLE | pi | 파이 상수값 3.141592~ 을 DOUBLE 데이터형으로 반환 |
| INT | pmod(INT 1, INT 2) | INT 1을 INT 2로 모듈러 연산(modular arithmetic)한 후 양수 INT 값을 반환 (음이 아닌 나머지를 반환) |
| INT | positive(INT i) | INT i의 양의 INT 데이터형 값을 반환 |
| INT | negative(INT i) | INT i의 음의 INT 데이터형 값을 반환 |
| FLOAT | sign(double A) | double A가 양수이면 부동소수점형 1.0을 반환, 음수이면 부동소수점형 -1.0을 반환, 모두 아니면 부동소수점형 0.0을 반환 |
| string | bin(BIGINT i) | BIGINT i의 이진수(binary)값을 문자열로 반환 |
| string | hex(BIGINT i) | BIGINT i의 16진수(hexadecimal)값을 문자열로 반환 |
| string | hex(string S) | string S의 각 문자를 16진수 값으로 변환하여 문자열로 반환 |
| string | unhex(i) | hex(string S) 함수와 반대로 동작 |
| string | conv(BIGINT i, from\_base, to\_base) | BIGINT I 를 from\_base 진수에서 to\_base 진수로 변환하여 문자열로 반환 (from\_base와 to\_base는 정수형) |
| string | conv(string S, from\_base, to\_base) | string S를 from\_base 진수에서 to\_base 진수로 변환하여 문자열로 반환 (from\_base와 to\_base는 정수형) |

1. **집계 함수 (Hive Aggregate functions)**

|  |  |  |
| --- | --- | --- |
| **반환 데이터형** | **집계 함수** | **설명** |
| BIGINT | count(\*) | NULL값을 포함한 행의 총 개수를 반환 |
| BIGINT | count(expr) | 주어진 표현식이 NULL이 아닌 행의 개수를 반환 |
| BIGINT | count(DISTINCT expr[, expr\_.]) | 주어진 표현식의 값이 유일 하면서 NULL이 아닌 행의 개수를 반환 |
| DOUBLE | sum(col) | 그룹 내 요소들의 합을 반환. |
| DOUBLE | sum(DISTINCT col) | 그룹 내 열의 고유한 값들의 합을 반환. |
| DOUBLE | avg(col) | 그룹 내 요소들의 평균을 반환 |
| DOUBLE | avg(DISTINCT col) | 그룹 내 열의 고유한 값들의 평균을 반환. |
| DOUBLE | min(col) | 그룹 내 열의 최소값을 반환 |
| DOUBLE | max(col) | 그룹 내 열의 최대값을 반환 |
| DOUBLE | variance(col)  or var\_pop(col) | 특정 열의 모집단 분산을 반환 |
| DOUBLE | var\_samp(col) | 특정 열의 표본 분산을 반환 |
| DOUBLE | stddev\_pop(col) | 특정 열의 모집단 표준편차를 반환 |
| DOUBLE | stddev\_samp(col) | 특정 열의 표본 표준편차를 반환 |
| DOUBLE | covar\_pop(col1, col2) | 두개 열(col1, col2)의 공분산을 반환 |
| DOUBLE | covar\_samp(col1, col2) | 두개 열(col1, col2)의 표본 공분산을 반환 |
| DOUBLE | corr(col1, col2) | 두개 열(col1, col2)의 상관관계를 반환 |
| DOUBLE | percentile  (int\_expr, p) | p percent에 해당하는 int\_expr 백분위수 값을 반환. p는 0과 1사이의 DOUBLE 데이터형 값임.  예) p가 0.25이면 전체 값을 크기 순으로 오름차순 정렬하였을 때 25%에 해당하는 값을 반환  참고) 백분위수(percentile)는 총 값의 백분율이 해당 값보다 작거나 같은 측정치임. |
| DOUBLE | percentile\_approx  (int\_expr, p, NB) | p percent에해당하는 int\_expr의 근사 백분위수 값을 반환. NB는 추정을 위한 Histogram bins의 수임.  (NB default 값 : 10,000) |
| DOUBLE | percentile\_approx  (int\_expr, [p1, …], NB) | p1, p2, … percent에 해당하는 int\_expr의 근사 백분위수 값을 반환 |
| ARRAY  (DOUBLE) | percentile  (int\_expr, [p1, …]) | p1, p2, … percent에 해당하는 int\_expr 백분위수 값을 ARRAY 데이터형으로 반환함. |

(참고 웹사이트) <http://rfriend.tistory.com/213?category=653567>

**HiveQL**



* SQL문 데이터 읽히는 순서

간략하게 말하면 **FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY** 순서 입니다.

FROM에서 읽어온 테이블에 WHERE 조건식에 해당하는 데이터가 아직 SELECT에서 선택되지 않았기 때문에 중첩을 사용하여 데이터를 불러오고 원하는 조건에 데이터를 사용할 수 있습니다!

* **중첩 SELECT 문**

Ex.)

**FROM** (

**SELECT** var\_1, var\_2, (var\_1 \* var\_2) **AS** var\_1\_2\_multi

**FROM** my\_table

)a

**SELECT** a.var\_1, a.var\_2

**WHERE** a.var\_1\_2\_multi > 100

**LIMIT** 100

요약 : my\_table 에서 var\_1, var\_2, (var\_1 \* var\_2) 을 선택한 데이터를 a라(치환 개념) 합니다. a 중에서 곱한 값이 100보다 큰 값을 가지는 var\_1, var\_2를 100개의 행으로 제한하여 데이터(a.var\_1, a.var\_2)를 추출합니다.

* **LIKE (Substring 문자열 일치 여부 확인)**

Ex.)

**SELECT** var\_1, var\_2, var\_3

**FROM** my\_table

**WHERE** var\_2 LIKE ‘%CC’ ; => var\_2에 ‘CC’로 끝나는 문자열이 들어있는 row 선택

**SELECT** var\_1, var\_2, var\_3

**FROM** my\_table

**WHERE** var\_2 LIKE ‘AA%’ ; => var\_2에 ‘AA’로 시작하는 문자열이 들어있는 row 선택

**SELECT** var\_1, var\_2, var\_3

**FROM** my\_table

**WHERE** var\_2 LIKE ‘%BB%’ ; => var\_2에서 ‘BB’를 포함하는 문자열이 들어있는 row 선택

* **CASE .. WHEN .. THEN … ELSE … END (범주형 변수 생성) , 조건문!!**

Ex.)

**SELECT** var\_1, var\_2,

**CASE**

**WHEN** var\_3 < 100.0 **THEN** ‘L’

**WHEN** var\_3 >= 100 **AND** var\_3 < 1000 **THEN** ‘M’

**ELSE** ‘H’

**END AS** var\_3\_cd

**FROM** my\_table

요약 : SELECT에서 선택한 var\_3 데이터 값이 100보다 작으면 ‘L’, 100보다 크거나 같고 1000보다 작으면 ‘M’ 두 조건 모두에 해당하지 않으면 ‘H’ 라는 값을 var\_3\_cd 라는 컬럼에서 출력합니다. (AS Hong 처럼 AS 뒤에 Hong은 Hong 이라는 새로운 컬럼을 만든다는 뜻.)

* **GROUP BY, HAVING. {집계(aggregation), 그룹 내 조건절(HAVING)}**

Ex.)

**SELECT** prd\_cd, max(prd\_price) **AS** prd\_max\_price

**FROM** prd\_tr

**WHERE** prd\_cd = ‘ABC’

**GROUP BY** prd\_cd

**HAVING** max(prd\_price) >= 10000

요약 : prd\_tr 테이블에서 prd\_cd = ‘ABC’ 조건인 prd\_cd를 소그룹화하고 소그룹화 내에서 prd\_price의 max값이 10000보다 크거나 같은 조건에 해당되는 데이터를 추출합니다.

* **ORDER BY (쿼리 결과 집합에 대한 전체 정렬(Total Ordering), 오랜 시간 소요)**

Ex.)

**SELECT** prd\_cd, prd\_id, sum(prd\_rev) **AS** prd\_rev\_sum

**FROM** prd\_tr

**WHERE** prd\_cd **IN** (‘ABC’, ‘DEF’, ‘GHI’)

**GROUP BY** prd\_cd, prd\_id

**ORDER BY** prd\_cd **ASC**, prd\_id **DESC**

요약 : prd\_tr 테이블에서 prd\_cd가 ‘ABC’, ‘DEF’, ‘GHI’ 조건인 데이터를 prd\_cd, prd\_id로 소그룹화 하고 prd\_cd 는 오름차순으로 prd\_id 는 내림차순으로 정렬하여 데이터를 추출합니다.

* **SORT BY, DISTRIBUTE BY (각 REDUCER에서 데이터 정렬(Local Ordering), 상대적으로 빠름)**

Ex.)

**SELECT** a.prd\_cd, a.prd\_id, sum(a.prd\_price) **AS** prd\_rev\_sum

**FROM** prd\_tr a

**WHERE** prd\_cd **IN** (‘ABC’, ‘DEF’, ‘GHI’)

**DISTRIBUTE BY** a.prd\_cd – 같은 prd\_cd 를 가진 데이터를 같은 reducer로 보냄.

**GROUP BY** a.prd\_cd, a.prd\_id

**SORT BY** a.prd\_cd **ASC**, a.prd\_id **DESC**

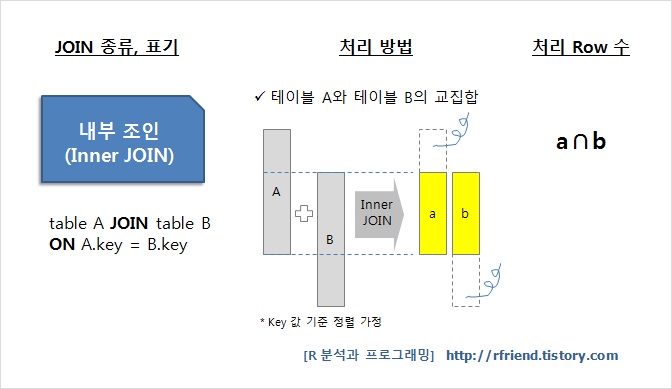
요약 : ORDER BY 와 공통점은 데이터를 정렬하는데 있습니다. 차이점은 DISTRIBUTE BY로 데이터를 reducer로 보낸 후 데이터를 SORT BY로 정렬하기 때문에 속도가 상대적으로 빠릅니다.

쿼리문에 대한 설명은 위 쿼리문 설명과 거의 동일하기 때문에 생략합니다.

**HiveQL 조인문!**

두 개 이상의 테이블을 서로 연결해서 분석할 때 사용하는 문장!

* **내부 조인 (Inner Join)**



[쿼리 예시]

**SELECT** a.var1, a.var2, b.var2

**FROM** mytable\_1 a **JOIN** mytable\_2 b

**ON** a.var1 = b.var1

**WHERE** a.prd\_cd = ‘ABC’ **AND** b.prd\_cd = ‘DEF’

요약 : mytable\_1 은 a, mytable\_2는 b!! / a의 var1과 b의 var1이 같을 때 a.prd\_cd가 ‘ABC’ 이고 b.prd\_cd가 ‘DEF’인 데이터 a.var1, a.var2, b.var2를 추출합니다.

**SELECT** a.var1, a.var2, a.var3, b.var4

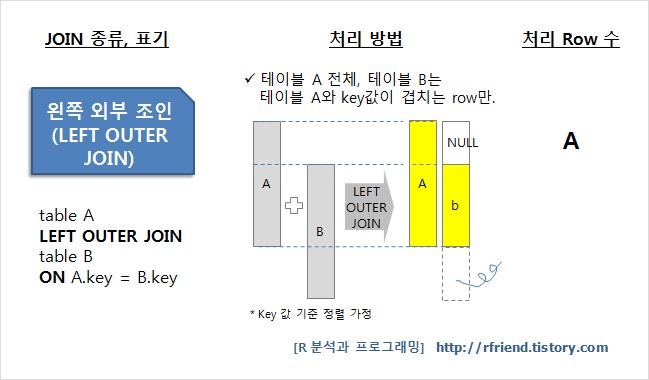
**FROM** mytable\_1 a **JOIN** mytable\_2 b

**ON** a.var1 = b.var1 **AND** a.var2 = b.var2

**WHERE** a.prd\_cd = ‘ABC’

요약 : 이전 예제와 내용은 거의 비슷합니다. 다만, a.var1 = b.var1 뿐만 아니라 a.var2 = b.var2의 조건도 만족해야 하는 데이터를 추출해야 합니다.

* **왼쪽 외부 조인 (Left Outer Join)**



[쿼리 예시]

**SELECT** a.var1, a.var2, a.var3, b.var4

**FROM** mytable\_1 a **LEFT OUTER JOIN** mytable\_2 b

**ON** a.var1 = b.var1 **AND** a.var2 = b.var2

**WHERE** a. prd\_cd = ‘ABC’

요약 : 위 Inner Join 예시가 이해된다면 left outer join 예시도 이해가 쉽습니다.

* **오른쪽 외부 조인 (Right Outer Join)**



[쿼리 예시]

**SELECT** a.var1, a.var2, a.var3, b.var4

**FROM** mytable\_1 a **Right Outer Join** mytable\_2 b

**ON** a.var1 = b.var1 **AND** a.var2 = b.var2

**WHERE** a.prd\_cd = ‘ABC’

요약 : Left outer join과는 반대로 오른쪽에 있는 테이블의 전체가 들어갑니다.

* **완전 외부 조인 (Full Outer Join)**



[쿼리 예시]

**SELECT** a.var1, a.var2, a.var3, b.var4

**FROM** mytable\_1 a **FULL OUTER JOIN** mytable\_2 b

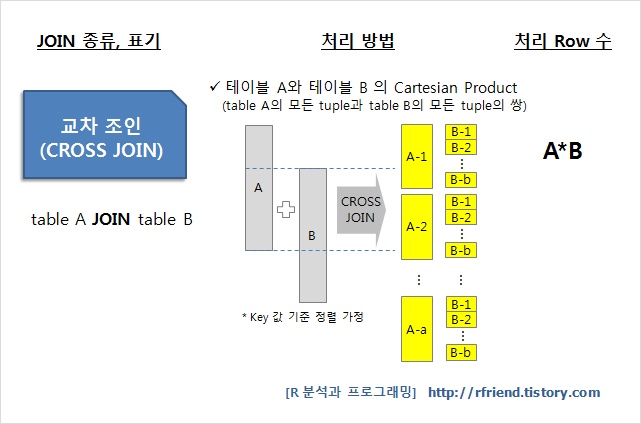
**ON** a.var1 = b.var1 **AND** a.var2 = b.var2

**WHERE** a.prd\_cd = ‘ABC’

요약 : Full Outer Join은 두 테이블 전체의 데이터를 포함합니다.

이 외에도 교차 조인(Cross Join), Map-Side Join 이 있는데 참고용으로 그림만 참조하겠습니다.

* **교차 조인(Cross Join)**



* **Map-Side join**

