출처 : (2019-01-28 ~ 2019-02-01)

Q.. 복잡하고도 어려운 SQL. 왜 배워야하나요? 그냥 작성된 것을 따라 써도 상관 없지 않나요?

A. 막연하게 따라쓰는 SQL은, 데이터베이스의 구조가 바뀌거나 새로운 데이터가 들어오기 시작하면 한두개씩 이상한 데이터가 조회되고, 의도치않은 결과가 나타나기 시작합니다. 곧, SQL문을 잘못 작성한 것이죠.

우리가 프로그래밍을 할 때, 코드를 복제해서 쓰지 않고 직접 배우듯, SQL 또한 DB로부터 올바른 데이터를 받을 수 있도록 작성하는 것을 배워야 합니다.

SQL에서 올바른 것은, 정확한 조회 결과 뿐만 아니라 성능(소요 시간을 줄이는 것)이 포함됩니다.

효율적인 SQL 사용을 위해선, ERD 사용이 적극 권장되나, 이는 추후에 다루도록 하겠습니다.

모든 RDBMS는 ANSI 표준을 따릅니다. 그러므로, Oracle이나 MSSQL 등의 문법 차이를 최소한으로 완화하려면, ANSI 표준을 위주로 문법을 익히는 것이 도움이 될 것입니다.

■ 데이터베이스란?

중복이 배제된 통합 데이터(Intergrated Data) 집합

컴퓨터화하여 저장된 데이터(Stored Data)

조직의 기능을 수행하는데 반드시 유지해야할 운영 데이터 (Operational Data)

한 조직에 있는 여러 응용 시스템들이 공동으로 소유하고 유지하며 이용하는 공용 데이터 (Shared Data)

■ 데이터베이스 목적

서로 다른 형태의 데이터(사용자 관점)의 통합

중복된 데이터의 일관성(Inconsistency) 유지

데이터의 정확성을 보장하는 무결성(Integrity) 유지

데이터 중복(Redundancy)의 최소화

회계도 SQL이 필수가 되는 추세

Oracle / Cybase가 널리 쓰이던 중, Cybase의 업데이트가 지연되던 중,

> 갑자기 MSSql 등장. (Cybase의 전문가들을 영입)

NOSQL DB (몽고DB?)

■ 관계형 Database의 정의 (Relational Database)

Key와 Value들의 관계를 table화시킨 전산정보 데이터베이스

Primary Key : 중복되어서는 안되며, Null을 가질 수 없음.

Foreign Key : 어떤 테이블의 기본 키를 다른 테이블에서 사용할 때 지정할 수 있는 키. 두 릴레이션 관계를 결정해주는 속성

■ SQL (Structured Query Language)

관계형 데이터베이스에 접근(생성, 변경, 삭제, 조회)하기 위한 비절차적 표준언어

1) 이해하기 쉬운 형태로 표현

2) 대화식 질의어로 사용 가능

3) 데이터 정의, 조작, 제어기능 제공

4) 레코드 집합 단위로 처리

5) 비절차적 언어

RBO 방식 / CBO 방식 (네비게이션 및 실시간 교통정보)

SQL Processing

■ 실습

<전체 선택>

SELECT \* FROM DEPT;

<일부 선택>

SELECT deptno Dept, loc\_code LOC

FROM dept;

<산술 연산>

SELECT deptno + 200 Dept, loc\_code LOC

FROM dept;

\* 산술 연산은 Table에 영향을 주지 않음 (불러온 후에 더해서 보여주는 것)

SELECT deptno + 200 \* 2 Dept, loc\_code LOC

FROM dept;

SELECT (deptno + 200) \* 2 Dept, loc\_code LOC

FROM dept;

<’Bonus’ Table>

SELECT \* FROM EMP;

Comm Column (Commition)이 Null인 경우가 있음.

SELECT \* FROM BONUS;

SELECT empno, sal, job, sal, ename, comm, comm + 100, nvl(comm, 0) + 300 from emp;

>> NVL을 통해, Null을 0으로 바꿔서 처리해주면 됨.

<별칭 (Aliasing)>

\_$# 3가지만 포함된 Aliasing 가능

하지만, 이름 규칙에 어긋나는 Alias는 Double Quotation으로 가능 (“”)

SELECT ename as "Name", sal, sal \* 12 as "Annual Salary" from emp;

<리터럴 문자열 사용>

SELECT ename || ‘ ’ || ‘is a’ || ‘ ‘ || job as “Employee Details” from emp;

SELECT ename Name, job Job from emp;

<DISTINCT, ALL>

<Order by & ASC, DESC>

SELECT ROWID, substr(rowid, 1, 6) Object\_Number,

substr(rowid, 7, 3) File\_Number,

substr(rowid, 10, 6) Block\_Number,

substr(rowid, 16, 3) Block\_ROWNumber

FROM emp

Order By절을 사용했을 때에는, Block Number 순으로 바꿔서 가져오며,

그렇지 않을 시 ROW ID 순으로 가져옴.

<WHERE>

Where 절에서는 Aliasing된 별칭을 사용불가.

(sql문은 작성된 순서와 해당 질의를 해독하는 순서가 다름.

작성 순서 : SELECT >> FROM >> WHERE

해석 순서 : FROM >> WHERE >> SELECT이므로.

Where가 먼저 수행되므로, Select의 Alias가 적용되기 전 해석되므로 참조가 불가)

<비교 연산자>

WHERE sal >= 1700

AND sal <= 1300

empno IN (7092, 7788, 7566)

-->

WHERE empno = 7902

OR empno = 7788

OR empno = 7566

문자 비교 연산자

\_ : 정확한 자리와 글자 수를 알고 있을 때 (한자리)

% : 정확한 자리와 글자 수를 모르고 있을 때

EMP Table에서 이름의 두번째 글자가 ‘A’인 사원의 이름, 급여, 업무 조회

SELECT ename, sal, job

FROM emp

WHERE ename LIKE ‘\_A%’;

영문 이름에 % 문자가 들어있는 인명 정보를 조회한다.

ESCAPE절

Ename like ‘%#%%’ ESCAPE ‘#’

>> ESCAPE 문자는 어느 문자로 지정되도 상관 없으며, 해당 문자 바로 뒤의 Wildcard 문자는 Wildcard 문자가 아닌 리터럴로 해석된다.

%%가 속한 경우에는

Ename like ‘%#%#%%’ ESCAPE ‘#’

Null과의 비교

NULL과 대소비교 및 상동비교시 그 어떤 데이터도 나오지 않음.

NULL은 일반적인 비교 연산으로 만나는 모든 DATA를 Null로 만듦

(IS NULL / IN NOT NULL 외에는 불가능)

(NVL NVL2로만 연산 가능)

<예제 풀이>

-- 예제 1

SELECT ENAME Name, JOB, SAL, DEPTNO

FROM EMP

WHERE SAL <= 1700

AND SAL >= 1300;

-- 예제 2

SELECT ename, job, sal, deptno

FROM emp

WHERE SAL BETWEEN 1700 AND 1300;

-- 아무런 결과도 나오지 않는다.

-- (BETWEEN 은 하한값을 먼저 지정해주어야 한다.)

<논리 연산자>

BONUS를 받는 사람 : Null 또는 0

WHERE COMM IS NULL

OR COMM = 0

WHERE NVL(COMM, 0) = 0

오류 케이스

WHERE COMM > 0 (보너스를 받는 사람)

--> WHERE COMM <= 0 (보너스를 받지 않는 사람 - 잘못된 케이스)

<예제>

--> 예제 1

SELECT EMPNO "No.", ENAME "NAME", SAL, DEPTNO

FROM EMP

WHERE JOB NOT IN('MANAGER', 'CLERK', 'ANALYST');

--> 예제 2

SELECT ENAME "Name", Sal, Job

FROM EMP

WHERE SAL NOT BETWEEN 1000 AND 3000;

--> 예제 3

Null은 ASC에서는 맨 마지막, DESC에서는 가장 처음에 있게 Default 설정되어있으므로,

ORDER BY Comm DESC NULLS LAST 처리를 해주면 됨.

(NULLS FIRST / NULLS LAST 구문으로 지정)

DISTINCT 등과 같은 절이 없으면, Table의 모든 Column으로 Order 가능

그러나, DISTINCT와 같이, SELECT절에 존재하는 Column으로만 Order 가능

ORDER BY 절은 어떤 상태에서든 마지막에 해석된다.

(Alias로 정렬도 가능)

SELECT empno, ename, job, deptno, sal

FROM EMP

ORDERBY 4, 5 DESC;

숫자는 SELECT문의 Column 순서…

<강평>

산술연산에서의 NULL은 복병

NULL은 산술연산의 결과를 무조건 NULL로 만든다.

NVL(Column, 0)을 통해서만 쓸 수 있다.

암시적 형변환

A char --> INDEX 있음

B NUMBER --> INDEX 있음

WHERE a = 10 --> 문자 Data보다 숫자 Data의 우선순위가 높으므로, 문제가 발생한다.

WHERE a = ’10’ --> 암시적 형변환이 일어나지 않음

WHERE b = 10 --> 암시적 형변환이 일어나지 않음

WHERE b = ’10’ --> 암시적 형변환이 발생하며, 숫자를 문자로 바꾸는 데에 제한은 없음. (quotation만 붙이면 되며, 오류 X)

A = to\_char(10) 또는 to\_number(a) = 10의 문장은 오류!

B = to\_number(’10’)는 정상

명시적 형변환

암시적인 형변환은 지양하고, 명시적인 형변환을 추구하는 것이 좋다.

To\_Char는, user들이 원하는 형식으로 꾸미기 위한 용도 (해당 Format을 맞춰주지 않았어도 에러가 잘 나지 않음).

To\_number, To\_date는 해당 Data를 주입, 연산하기 위해 해당 Data가 유효한지 검증하는 목적(해당 Format을 맞춰주지 않으면 Error)

<To\_Char>  
Location이 대한민국으로 설정되어있으면, 부동지역 통화 기호인 L 자리에 \이 옴.

<년도 형식>  
YY와 RR

--> 예제 6

SELECT TO\_CHAR(TO\_DATE(‘98’, ’RR’), ‘YYYY’) test1,

TO\_CHAR(TO\_DATE(‘05’, ’RR’), ‘YYYY’) test2,

TO\_CHAR(TO\_DATE(‘98’, ’YY’), ‘YYYY’) test3,

TO\_CHAR(TO\_DATE(‘05’, ’YY’), ‘YYYY’) test4,

FROM DUAL

--> 예제 7

SELECT ename, hiredate, TO\_CHAR(NEXT\_DAY(ADD\_MONTHS(hiredate, 6), 1), ‘yyyy/month/dd’) “NEXT 6 Month”

FROM emp;

Month Format 중에서 길이가 가장 긴 Data의 길이에 맞게 공백이 맞춰져서 공백이 생김.

FM이라는 Format을 사용하면 해당 공백을 제거하여 깔끔하게 표현이 가능.

SELECT ename, hiredate, TO\_CHAR(NEXT\_DAY(ADD\_MONTHS(hiredate, 6), 1), ‘FMyyyy/month/dd’) “NEXT 6 Month”

FROM emp;

INTERVAL DAY TO SECOND

<일반함수>

NULL 관련 함수 NVL, NULLIF, COALESCE

GREATEST와 LEAST는 NULL과 관련이 없지만, 해당 Argument 중 NULL이 포함되어있다면, 해당 비교를 할 수가 없으며, 결과값은 무조건 NULL로 수렴

--> 예제 1

본봉과 Bonus를 합산하여 출력하려는 부분인데,

Bonus가 NULL인 경우, 합산된 결과값이 NULL로 표기되므로, 해당 0으로 바꿔주는 작업이 필요.

NVL(BONUS, 0)

<DECODE / CASE>

DECODE를 전통적인 조건 처리 함수로 운용중이었으나, Equal 조건에 대해서만 사용이 가능함.

그러나, 다른 SQL에서는 CASE도 사용하고 있었고, ANSI 표준에 등록되어 해당 Case 함수를 채용.

(다른 DB로의 Migration이 용이해지길 위함.)

DECODE : CODE가 짧아서 굉장히 효율적

그러나, Equal 조건만 처리할 수 있는 단점.

다중 DECODE와 보정 함수로 인해 코드의 가독성과 효율이 많이 줄어들 수 있음.

CASE : 여러 번의 중복 Case도, 보정 함수도 필요없이 더욱 편리하게 사용할 수 있다.

--> 예제 10

Decode의 경우, sign 함수를 함께 사용해야하기 때문에, 한 행당 3회 이상의 함수를 거쳐야 한다.

그러므로, 효율성과 가독성이 떨어지게 됨.

SELECT ename, sal,

Case when sal < 1000 THEN ‘A’

When sal <2500 then ‘B’

Else ‘C’

Ens grade

From emp;

<실습2>

15. 1981 년도에 입사한 사원들의 이름과 입사일을 출력하시오 !

SQL> select ename, hiredate

from emp

where hiredate between TO\_DATE('81/01/01','RR/MM/DD')

and to\_date('81/12/31','RR/MM/DD');

and to\_date('81/12/31','RR/MM/DD');

부분에서, 1981/12/31 00시까지의 Data만 검색되므로,

12월 31일 자정 이후의 Data는 범위에 해당되지 않는다.