<데이터베이스 실습 복습 정리>

160926~161005(DDL to DML / Constraint)
DDL은 테이블이나 제한 조건을 만드는 문장이고, DML은 Insert, Delete, Update, Merge에 관련되어서 쓰는 문장이다. 또한 이러한 문장을 추가하는데 있어서 무리가 가지 않게 하는 Constraint 개념도 숙지를 하는 것이 목표이다. 이번 개념은 햇깔리는 것이 더러 있으니 조심해서 공부해두도록 하자.

---

6. DDL(테이블 정의) at 160926

1) Create Table 기본

Create 문은 영어로 만들다, 창조를 하다라는 뜻이다. 말 뜻대로 무언갈 만드는데 쓰는 문장인 것을 숙지하였을 것이다. 우리가 일단 배울 내용은 Create Table 문장이다. 다음과 같은 문장을 살펴보자.

```
Create Table ILLIONAIRE
(Member_ID Number(3) Constraint Ill_PK Primary Key, Member_Name Varchar2(20) Not Null, Enterence_Date Date Default Sysdate);
```

데이터를 추가할 경우에는 다음과 같은 순서대로 입력을 해둬야 한다.
(컬럼이름) (데이터이름) (제약조건) (Default인 경우)
이 순서에 대해서는 어디 가서라도 숙지를 해둬야 도움이 된다. 잊지 말도록 하자.(물론 제약 조건은 컬럼들에 대해 정의가 완료된 후에 Constraint Ill_PK Primary Key(Member_ID) 이런 식으로 추가가 가능하다는 점도 미리 알아두자.)

데이터에 대한 종류로서는 다음과 같다. 무엇이 있는지에 대해서는 이미 알 것으로 판단하고 다시 익혀보도록 하자.
Number : 숫자 데이터를 뜻한다. 그냥 이를 이용하게 된다면 숫자의 범위는 상관없이 꼴리는 대로 추가하면 된다.
Number(7, 2) : 숫자 데이터를 뜻하는 건 다 알지만, 7자리로 표현되는 숫자 이내에서만 저장이 가능하다.(즉 1000만 이내.) 그리고 뒤에 2를 적어두게 되면 소수점이 뭐가 됐든 간에 무조건 뒤에 붙게 해준다.(1000000.00 이런 식으로)
Date : 말 그대로 날짜형 데이터이다. 이는 방금 배운 Session 중에서 NLS_Date_Format에 따라서 출력하게끔 해준다.
Char(10) : 10글자 이내의 글자만 저장이 가능하다. 그렇지만 Varchar2에 비해서 10글자 그대로 용량을 차지해서 용량 낭비라는 단점이 있다.
Varchar2(10) : 10글자 이내의 글자만 저장이 가능하다. 10글자 이내에 저장된 문자들에 대해서만 용량을 차지하기 때문에 용량 절약에 적합한 데이터이다.

제약 조건에 관련되어서는 1학기 때 배워왔던 Constraint 문장으로 설명하겠다. 이는 쉽게 이야기해서 컬럼에 제약 조건을 넣어주는 뜻이다. 제약조건은 하단부에 내려가 보면 자세히 설명하겠다.

여기서 Default 값은 아무런 값이 추가가 안 되는 경우에 비상으로 추가하는 개념이다. 그래서 Enterence_Date가 아무런 값이 없다면 Sysdate(즉, 현재 날짜)가 지정이 되게끔 하는 뜻이다.

테이블의 이름에 관련되어서는 다음과 같이 정의가 된다.
-> 테이블 이름은 꼭 알파벳으로 시작이 가능. 특수문자(\n, \‘, \“ 이런 거)도 사용이 가능하나 “”으로 감싸야 하니 써먹진 말자.
-> 30Byte 크기(즉 한글로 8~10글자, 영어로 30글자)로 정의가 가능하다.
-> 한 명의 사용자가 다른 오브젝트들의 이름과 중복으로 사용이 불가하다. 재미있는 건 HR 사용자가 test 테이블을 만들어도 다른 사용자가 사용이 가능하다.
-> Select, From, Sysdate(즉 키워드)으로는 절대로 못 짓는다. 쉽게 말해서 자바에서 변수 이름을 int, double로 못한다는 원리로 생각하면 된다.
-> 테이블은 물론 한글로서도 작성이 가능한건 다 알 것이다. 그러나 원만한 실습을 위해선 한글보단 영어로 연습을 해보는 걸 권장한다.

2) Create Table AS(테이블 복사)

AS는 방금 배웠던 Alias의 줄임말이기도 하지만 여기서는 ~로서 저장(All Save)을 하라는 뜻이다. 일단 간략하게 다음과 같은 문장을 살펴보겠다.

```
Create Table Jobs_Plus1
AS
  Select * from jobs;
```

이처럼 작성을 하게 된다면 jobs 테이블에 있는 모든 내용들(즉 컬럼이든 튜플이든 모든 다)을 Jobs_Plus에 추가하는 개념으로 생각하면 된다. 물론 다음과 같이 컬럼들을 따로 정의해서 복사도 가능하니 참조하자.

```
Create Table Jobs_Plus2
AS
  Select Job_ID, Job_Title from jobs;
```

여기서 재미있는 사실은 튜플들은 안 가져와도 되고 컬럼만 그대로 써먹고 싶다면 다음과 같이 작성을 해도 무방하다.

```
Create Table Jobs_Plus3
AS
  Select *
  from jobs
  where 1=2; // 이처럼 작성을 하게 된다면 컬럼의 요소들만 복사하는 개념으로 생각하면 된다.
```

물론 Where 문에 여러분들이 원하는 조건에 대해 작성을 하고 새로운 테이블에 저장이 가능하다!!!

```
Create Table Jobs_Plus4
AS
  Select job_id, job_title
  from jobs
  where job_id Like ‘IT\_%’ escape ‘\’;
```

(주의사항) 여러분들이 꼴리는 컬럼을 하고 싶다면 다음과 같이 큰 따옴표는 굳이 안 쓰도록 하자. 쓰게 된다면 “JOB_아이디”로 컬럼을 써야 하는 일이 벌어진다. 다만 여러분만의 컬럼을 쓰고 싶다면 AS를 통해서 쓰되, 큰 따옴표를 생략하게 되면 좋을 것이다.

```
Create Table Jobs_Plus4
AS
  Select job_id JOB_아이디, job_title JOB_제목
  from jobs
  where job_id Like ‘IT\_%’ escape ‘\’;
```

3) Alter Table로 컬럼/테이블 다루기

여러분들이 가령 테이블 이름을 수정하거나 컬럼 이름을 수정하거나 데이터 형태를 바꾸거나 컬럼을 없애는 경우 주로 써먹는 문장이니 참조하도록 하자.

3-1) 새로운 컬럼 추가하기

```
Alter Table Subway
Add (Population2 Number(10));
```

이처럼 작성을 하게 된다면 새로운 컬럼이 형성이 된다. 물론 제약조건도 뒤에 작성해서 추가를 할 수 있으니 참조하도록 하자.(다만 Foreign Key는 컬럼을 추가하고 난 뒤 따로 Add Constraint로 해야 된다...)

3-2) 컬럼 이름 변경/테이블 이름 변경

```
Alter Table Subway
Rename Column Open_Date to OpenDay;
```

이처럼 작성을 하게 된다면 Open_Date 컬럼의 이름이 OpenDay로 변경이 된다. 물론 컬럼도 변경이 가능한데 테이블도 변경이 안 된다는 건 말도 안된다. 다음과 같이 작성해주면 테이블의 이름이 변경된다.

```
Alter Table Subway
Rename Subway To SeoulSubway9;
```

Rename의 구성요소는 다음과 같다고 생각하면 된다.

Rename (Column) 본래이름 To 바꿀 이름

3-3) 컬럼 데이터 형식 변경

```
Alter Table Subway
Modify(Stn_ID Number(5));
```

Modify 키워드를 이용하게 된다면 각 요소에 대한 정보를 다시 재정의를 해주는 역할이라고 생각하면 된다. 그래서 Modify(컬럼이름 변경할_데이터형식)을 이용해줘서 변경을 하면 되지만 주의해야 할 점이 있다. 만일 그 컬럼에 대한 튜플들이 저장되어 있다면 이에 대해 호환을 시키는데 영향이 있으니 적당한 결과를 생각해서 변경하자.

3-4) 컬럼 없애기

```
Alter Table Subway
Drop Column Population2;
```

이처럼 작성을 하게 된다면 제약 조건이 걸려 있지 않는 이상 정상적으로 삭제가 된다. 그러나 Foreign Key 제약 조건에 걸린 테이블에 대해서는 삭제가 제대로 실행이 안 될 것이다. 그래서 다음과 같이 키워드를 조금 추가해줘서 삭제를 하면 완벽히 삭제가 된다.

```
Alter Table Jobs
Drop Column Job_ID Cascade Constraints;
```

여기서 Cascade라는 뜻은 영향을 받는 것을 없애는 것. 즉 이 컬럼에 대해서 연쇄 반응을 하는 제약 조건에 대해서도 제거한다는 뜻이다. 그래서 만일 참조되는 테이블 내의 참조 되는 컬럼(즉 부모 테이블 내의 컬럼)에 대해 제거를 하게 된다면 그 컬럼에 관련된 Constraints(제약조건)에 대해서도 제거를 해서 외래키 제약 조건까지 없앤다는 뜻이다. 그래서 본래 Job_ID라는 컬럼은 삭제가 불가능했지만 이를 이용해 제대로 삭제가 가능하게 되는 것이다.

3-5) 읽기 전용 테이블로 변환하기

```
Alter Table Subway Read Only;
```

이처럼 간략하게 작성을 하게 된다면 본래 테이블은 Insert, Delete, Update 문을 쓸 수 있게 되었지만 읽기 전용 테이블에 관련해서는 제어를 못하게 되는 문장이다. 그래서 이를 작성하게 되면 테이블 내의 값들에 대한 제어를 못한다고 생각하면 된다. 그래서 다음과 같이 원래대로 돌려놓을 수 있으니 참조하도록 하자.

```
Alter Table Subway Read Write;
```

4) Truncate VS Delete from table

우리가 방금 배웠던 Trunc 문이 있을 것이다. Trunc라는 뜻은 버리다, 걍 없애다 라는 뜻으로 생각을 하면 된다.

```
Truncate Table 테이블_이름
```

이를 작성하게 된다면 모든 튜플에 대해 없애버리겠다는 뜻으로 판단하면 된다. 물론 Delete from 테이블_이름으로 해도 모든 튜플들이 없어진다. 허나 두 녀석들에 대해 비교를 한다면 Truncate Table는 DDL이고, Delete from은 DML이다. DDL는 커밋이든 롤백이든 나발이든 상관을 안 하고 그 사용자가 가진 테이블에 대해 그대로 적용을 하게 된다. 그러나 DML는 커밋을 통해서 저장하고 롤백을 하면 되니깐 전체가 삭제되진 않게 된다. 그래서 본래대로 돌아가고 싶다면 Delete 문을 이용하도록 하자.

5) Drop Table

```
Drop Table 테이블_이름
```

이를 작성하게 된다면 테이블_이름의 테이블의 모든 정보가 싹 다 날라 간다고 생각하면 된다. 컬럼이든 튜플이든 모든 다 없어지니깐 신중히 생각해야 하지만 재미있는 사실은 이 테이블은 Recyclebin 이라는 테이블로 돌아가게 된다. 그래서 완전히 버려진 것이 아니라고 생각하면 된다. Recyclebin의 정보는 다음과 같이 작성하게 되면 볼 수 있다.

```
Select Original_name, operation, droptime
from recyclebin;
```

이를 작성하게 되면 현재 버려진 테이블에 관련되어서 보여준다. 버린_테이블_이름, Drop, 버려진_시간에 대해 확인을 할 수 있으니 참조하도록 하자.

또한 테이블도 Flashback를 통해서 원상복귀를 할 수 있으니 참조하자.

```
Flashback Table LeeTable to before drop;
```

Flashback Table 테이블_이름 to before drop를 이용하게 된다면 휴지통에 버려진 테이블이 본래로 복귀하게 되니 만일 실수로 테이블을 버렸다면 이 문장을 활용해보도록 하자. 물론 뒤에 Rename to 바꿀_이름을 해주면 버려진 본래 테이블의 이름도 바꿀 수 있으니 참조하자. 그러나 이를 작성하게 된다면 어떻게 될까? 우리가 휴지통을 들어가면 비우기를 하면 아예 파일이 없어지게 된다. 이런 원리를 이용해서 다시 작성을 해보겠다.

```
Drop Table LeeTable purge;
```

purge라는 뜻은 제거하다, 몸과 마음을 청산하다 이런 뜻이다. 그래서 본래 Drop은 테이블을 숨기는 개념으로 생각하고 purge를 통해서 완전히 버려지는 것이다. 그래서 테이블을 지울 때 신중하게 생각해서 purge를 이용하도록 하자. 안 그러면 그 테이블에 관련되어 복귀하려면... 차라리 새로 만드는 과정이 더 빠를 것이다.

6) 데이터 딕셔너리(중간고사에선 변별력을 위해 선택형으로 나올 듯 하니 공부해두자...)

우리가 문장을 작성하면서 데이터 딕셔너리라는 녀석을 가끔 본 적이 있을 것이다. 이는 쉽게 이야기해서 통장을 만든다면 내 자신이 아닌 이상 터치를 하게 된다면 타인이 비밀 번호를 모르는 이상 보안성을 강화시킨다고 생각을 하면 된다. 그래서 어느 정보가 있는지에 대해 숙지를 간략히 해둘 필요가 있다.
- 오라클 DB의 메모리 구조와 파일에 대한 구조 정보들
- 제약 조건 정보들
- 각 오브젝트들이 이용하는 공간 정보
- 사용자 정보
- 권한 or 프로파일, 롤에 대한 정보
- 감사(Audit, 그 아리가토 아니다...)에 대한 정보

이러한 정보들 이외에도 여러 정보들이 있지만 데이터 딕셔너리의 정보를 두 가지로 나뉘어서 저장을 하게 된다. Data Dictionary View, Base Table 두 가지로 나뉘는데 세부 정보에 대해 알아보도록 하자.

Base Table : Db를 생성하는 시점에 자동으로 만들어진다. 여기서 DBA(즉 디비 관리자)도 이에 관련되어서는 터치가 불가능하다. 즉 연대장이든 사단장이든 부대 관련정보는 정보과장(혹은 장교)만이 터치를 할 수 있다고 생각을 하면 된다. 허나 이러한 정보는 Select를 이용해서 살펴볼 수는 있는데 Data Dictionary View를 통해서만 가능한 일이다. 또한 DDL를 통해서 변경할 시에는 SERVER Process가 사용자를 대신해 해당 딕셔너리 내용을 변경시켜준다고 생각하면 된다.

Data Dictionary View : 이를 수행하기 위해서는 디비 관리자가 Catalog.sql이란 파일을 따로 수행을 해줘야 가능하다. 그래서 수동적으로 db를 생성하기 위해 디비 사용자의 손에 따라 달려있다고 생각하면 된다. 이는 2가지로 나뉘게 되니 참조하자.

-> Static Data Dictionary View : 그 속에 있는 내용들이 자동으로 변경되는 것이 아닌 수동적으로 변경해줘야 하는 개념이다. DBA_, ALL_, USER_로 시작하는 경우는 거의 이거로 생각하면 된다. 물론 DBA_로 시작하는 경우는 디비 관리자 아닌 이상 터치가 불가능하다. 또한 ALL_로 시작하는 것은 모든 사용자가 접근 가능하다고 생각하면 된다.

-> Dynamic Data Dictionary View : 해당 정보를 Control File이나 현재 메모리에서 조회를 해서 새롭게 항상 갱신되어 있다고 생각하면 된다. V$로 시작하는 모든 테이블은 대부분 이거로 생각하면 된다.

---

7. DML 활용하기 at 160928

Insert문, Delete문, Update문, Merge문...
이것이 전부 테이블 내에 있는 튜플들을 다룰 수 있는 DML 문장으로 생각하면 된다. 우선 Insert 문부터 천천히 정리해보자.

1) Insert 문

Insert 문은 말 그대로 삽입하는 문장으로 생각하면 된다. Insert 문은 다음과 같이 작성하는 것으로 이미 숙지하고 있을 것이다.

```
Insert Into Employees(Employee_id, last_name, first_name)
Values(100, '김밥', '천국');
```

물론 컬럼을 따로 정의하지 않았더라도 모든 컬럼의 내용을 숙지하고 있다면 굳이 작성해주지 않아도 된다.

```
Insert Into Subway
Values('K155', 'K245', '수원',
TO_DATE('1905/1/11', 'YYYY/MM/DD'), 46258,
'Y');
```

특정 컬럼들만 추가하기 위해서는 위 방법을 이용하도록 하고, 아래 방법은 모든 컬럼에 관련해서 추가를 할 수 있으니 참조하도록 한다. 물론 날짜형 데이터를 추가하는데 있어서 **TO_DATE를 적절히 사용해서 추가하는 연습**도 해줘야 한다.

2) Insert Into 활용(1) 튜플들을 복사하기

어느 테이블에 담겨진 튜플들을 새로운 테이블에 추가하는 방법이 있다. 이는 SELECT를 이용하면 된다. 다음과 같은 문장을 살펴보자.

```
Insert Into SubwayInstance
select stn_id, name, population
from Subway
where Trunc(Stn_id, -2)=900;
```

이처럼 작성을 하게 된다면 9호선 역(원래 역 번호 백의 자리가 그 노선을 나타낸다.)들에 관련된 역사에 대해 저장을 하는 역할을 하고 stn_id, name, population 3개의 컬럼에 관련된 내용이 저장하게 된다. 물론 Create Table AS를 이용하는 방법도 있긴 하나 그건 여러분의 선택이다. commit를 이용해서 저장해두고 싶다면 이를 작성해보는 연습을 해보는 것도 나쁘지 않다.

3) Insert Into 활용(2) All 키워드 활용

2-1) 다른 테이블에 추가할 요소를 각 테이블에 동시에 추가하기

```
Insert All
Into 삽입할 테이블1 Values(컬럼들)
Into 삽입할 테이블2 Values(컬럼들)
select * from 찾아볼_테이블
```

이를 이용해서 작성을 하게 된다면 필요한 컬럼들을 골라내서 추가를 해주면 된다. 그래서 다음과 같은 문장을 사례로 참조를 해보겠다.

```
Insert All
Into Employees1
Values(employee_id, last_name, salary)
Into Employees2
Values(employee_id, last_name, salary)
select *
from employees
where department_id IN(10, 20, 50);
```

이렇게 작성하게 된다면 employees에 있는 컬럼을 참조하게 된다면 모든 컬럼을 참조해서 employee_id, last_name, salary 3개의 컬럼을 각각 추가해줌으로서 문장이 종결나게 된다.

2-2) 데이터를 직접 입력해서 추가하기

```
Insert All
Into Employees1
Values(102, '이용주', 30000)
```

Into Employees2
Values(104, '김호창', 120000);
select * from dual;
dual 테이블은 가상의 테이블로서 일단 이를 지정해 줘야지 값을 추가하는데 있어서 문제가 발생하지 않는다. 각각 한 행의 값을 직접 입력해서 추가할 경우에는 이를 작성해서 이용해 보길 바란다.

2-3) 데이터를 조건별로 나누어서 추가하기
Insert All
When salary Between 10000 and 20000
then Into Employees1(컬럼요소생략)
Values(employee_id, last_name, salary)
When salary Between 5000 and 9999
then Into Employees2(컬럼요소생략)
Values(employee_id, last_name, salary)
select *
from employees;
이렇게 작성하게 된다면 employees에 있는 테이블을 참조해서 employee_id, last_name, salary 컬럼에 대해 월급별로 조건을 나누어서 각 테이블에 저장하는 역할을 해주니 다음과 같은 문장을 외워두길 바란다.
Insert All
When (조건1)
then into 테이블1(컬럼요소)
Values()
...

3) Update/Delete 문

Update 문은 당연히 알겠지만 컬럼에 대한 요소에 대해 갱신할 때 쓰는 문장이다. 다음과 같은 구조로 되어 있으니 굳이 설명하진 않겠으나 조건을 안 쓰게 된다면 테이블의 컬럼에 대해 변경할 값으로 모두 바뀌는 일이 발생하게 되니 주의하도록 하자.
Update employees
set salary=12345
where employee_id=130;

물론 Delete 문은 다음과 같이 작성하면 손쉽게 작성을 할 수 있다. 그렇지만 조건에 대해서 아무 것도 작성하지 않으면 모든 튜플이 없어져 버리는 Truncate와 같은 일이 일어나니 유두리있게 알아서 잘 쓰자.
Delete from employees
where hire_date<=TO_DATE('YYYY/MM/DD', '2013/10/2');

4) Merge 문

Merge 문은 생각보다 너무 길어서 이해를 하면서 숙달하는 연습이 필요로 하다. 필자도 공부를 해보면서 제일 어려웠던 개념이 바로 Merge 문이다. 그래서 이를 제대로 활용하기 위해서는 원리를 이용해서 숙달해보는 연습을 해보자.
Merge Into 저장할_테이블 (A.K.A. 여기선 저장으로)
  Using 참조_테이블 (A.K.A. 여기선 참조로)
  On(저장.같은값=참조.같은값)

When Matched then
  Update Set 저장.바꿀컬럼=참조.원래컬럼
When Not Matched Then
  Insert Values(참조.추가할 컬럼들)
이처럼 작성을 하게 된다면 저장될 테이블 내의 값들에 대해서 참조 테이블의 값을 추가하는 내용이고, 같은 값이 있는 경우에는 저장될 테이블의 값을 참조 테이블의 값으로 바꿔주는 연습을 해봤다. 그러나 이 문장에 대해서는 많은 연습을 요하니 꼭 해보길 바란다.
이번 문제에는 지난번 과제 10번을 참조해서 풀이를 해보면서 중요 개념을 잡아보는 연습을 해보겠다.
10. SAL_HISTORY_2 테이블과 SAL_HISTORY 테이블을 병합한다. SAL_HISTORY_2 테이블의 행이 SAL_HISTORY 테이블과 일치하면, SAL_HISTORY 테이블의 급여 값을 SAL_HISTORY_2 테이블에 있는 값으로 수정한다. SAL_HISTORY_2에 있는 행이 SAL_HISTORY에 없으면 해당 행을 삽입한다.

-> 여기서 우리가 저장할 테이블과 참조할 테이블에 관련되어서 숙지를 해둬야 한다. 저장할 테이블은 바로 Sal_History로 작성을 해야 한다. 그리고 참조할 테이블은 Sal_History_2로 쓰면 된다. 그래서 Sal_HIstory를 기준으로 병합을 해보겠다.
Merge Into Sal_History sh1
  Using Sal_History_2 sh2
  on (sh1.employee_ID=sh2.empID)
When Matched then
  Update Set sh1.salary=sh2.sal
When Not Matched Then
  Insert Values(sh2.EmpID, sh2.Hiredate, sh2.sal);
이처럼 작성하게 된다면 각각 같은 요소가 들어가 있는 것은 이미 숙지했을 것이다. 그러나 식이 하도 햇깔려서 생각보다 풀이하는데 어려웠을 것이다. 그래서 다음과 같은 문제를 풀어본다면 우선 Sal_History에 분명히 병합을 해야 한다는 점은 이미 다 알고 있을 것이다. 그래서 이곳에 Sal_History_2에 있는 값을 추가 시켜주고, 같은 값이 있다면 sh2.sal에 있는 값으로 고쳐주는 문장을 한 번 작성을 해줌으로서 어렵게 풀었을 것이다. 혹여 모르니 차후에 시간난다면 이 문장을 중점적으로 연습을 해보는 시간을 갖도록 하자.

5) Rollback to the commit

commit 문장은 한 마디만 치게 된다면 여태껏 적용된 값들에 대해 저장을 시켜주는 역할을 한다. 그래서 이를 작성해 주고 차후에 중간에 사용자가 만들다가 savepoint s1을 만들어 주는 경우가 더러 있을 것이다. 그런 경우에는 s1까지 해왔던 내용이 저장되는 것이 아니라는 점을 알려주는 바이다. commit를 안 한 이상 여기는 중간 저장점이다. 그래서 rollback을 한다면 결코 commit한 지점으로 돌아간다. 그리고 참고로 사용자가 했던 지점까지 복귀를 하고 싶다면 rollback to s1을 이용해 보도록 하자.
그리고 rollback, commit, savepoint 문을 모아둔 문장을 TCL(Transaction Control Language)라고 칭한다. 데이터를 변경하는 명령어들을 모아둔 세트를 뜻함으로서 차후에 데이터베이스 프로젝트를 이해하는 과정이 필요로 할 때 롤백, 커밋, 세이브 포인트 모르고 가면 차후에 힘들 것이다.
---

8. Constraint at 161006

우리는 테이블 내에 아무런 값을 추가하는 경우를 방지하기 위해서 제약조건에 대해서 공부를 해둘 필요가 있다. 이 시간에는 제약조건이 어떠한게 있는지에 대해서 배워왔고, 제약조건에 대해 변경하는 것 까지

배웠다.

1) 제약조건의 종류와 특징

NOT NULL : 이 컬럼이 걸리게 되면 이 컬럼에 null 이란 걸 일체 추가할 수 없게 된다.

UNIQUE : 이 컬럼이 걸리게 되면 유일한 값들로만 추가가 가능하게 된다. 허나 이는 이름이나 주민번호 같은 컬럼에 쓰지 말길 바란다.(주민번호가 일치하는 사람도 더러 존재한 경우도 있었다.)

PRIMARY KEY : 이 조건은 NOT NULL과 UNIQUE의 개념을 합쳤다고 생각을 하면 쉽다. 그러나 실질적으로 테이블계의 회장, 사장님 역할을 하듯이 테이블의 기본키 역할을 함으로서 테이블 내의 세부정보를 구별하기 쉽게 해주는 역할이라 보면 된다.

FOREIGN KEY : 다른 테이블의 컬럼을 참조하게 도와주는 외교부와 같은 역할이라고 생각하면 된다. 이는 외래키로서, 다른 테이블의 기본키를 통해서 참조를 하도록 도와준다.

CHECK : 설정된 범위 값만 입력을 허용하게끔 하는 조건이다.

2) Create 문으로 제약조건을 추가하기

우선 제약조건을 걸는 방법은 2가지로 나뉘게 된다.
1. 제약조건을 거는 동시에 테이블에 저장하는 법
2. 테이블 구조 완성 뒤 제약조건을 추가하는 방법

쉽게 설명하기 위해서 다음과 같은 테이블이 있다고 가정을 하자.
테이블 이름 : Emp_Table

컬럼 이름	ID	Name	loc_ID
참조 테이블	.	.	Location
참조 컬럼	.	.	Location_id
제약 조건	Primary Key	Not Null	Foreign Key
데이터 종류	Number (5)	Varchar2 (30)	Number (5)

이 조건을 통해서 Emp_Table을 만들어보는 문장을 작성하겠다.

2-1)
```
Create table EMP_TABLE
(ID Number(5) Primary Key // 1
, Name Varchar2(30) Constraint Emp_Tab_NN Not Null // 2
, Loc_ID Number(5) Constraint Emp_Tab_FK Foreign Key References Location(Location_ID)); // 3
```

1 : 제약 조건의 이름을 따로 정의하지 않더라도 문장 작성하면 오류는 안 나지만, 차후에 제약 조건을 삭제하는 경우에는 큰 어려움이 있음을 알린다.

2 : Constraint 과 Not Null 사이에 제약조건의 이름이 새로 정의되었음을 알 수 있다. 이처럼 작성하게 되면 차후에 제약조건을 터치하거나 삭제하는 경우에 주로 쓰이게 되니 꼭 이름을 설정할 필요가 있다.

3 : Foreingn Key와 Check 제약조건에 대해서는 정의하는데 있어서 다른 제약조건들과는 다르다. 어떤 구조인지 아래를 살펴보자.

Constraint (제약조건_이름) Foreign Key
References 참조테이블(참조테이블_컬럼)

Constraint (제약조건_이름) Check(컬럼조건)
Foreign Key은 뒤에 무조건 References가 붙어야 된다. Foreign Key 안에 References가 없다는 것은 쉽게 이야기 하면 고기나 계란 없는 햄버거나 마찬가지다. 또한 Foreign Key가 가리키는 컬럼이 Unique, Primary Key가 아닌 경우에는 추가하기 힘드니 참조하자.
물론 Check에도 조건을 넣어줘야 되니 꼭 써두도록 하자!

2-2)
```
Create Table EMP_TABLE
(ID Number(5), Name Varchar2(30), Loc_ID Number(5),
Constraint EMP_TAB_PK Primary Key(ID),
Constraint EMP_TAB_NN Not Null(Name),
Constraint EMP_TAB_FK Foreign Key(Loc_ID)
References Location(Location_ID));
```

뒤에도 제약조건들이 추가가 될 수 있다만, 주의할 점은 동시에 형성하게 되면 Primary Key, Not Null, Foreign Key 뒤에 (컬럼이름)을 작성하지 않아도 되지만, 뒤에 붙는 만큼 그만한 대가라고 생각하면 된다.

3) Foreign Key 뒤 On Delete

Foreign Key 구조가 이것만이 아니다. 이는 차후에 튜플들을 관리하는데 있어서 꼭 알고 있어야 하는 개념이다. 가령 예를 들어 위의 문장 중에서 2-2의 마지막 제약조건을 재작성해보자.
```
Constraint EMP_TAB_FK Foreign Key(Loc_ID)
References Location(Location_ID) On Set Null
```
On Set Null은 Location_ID가 가령 320인 정보들이 없어지게 된다면 EMP_Table의 Loc_ID가 320인 튜플은 이 값에 대해서 Null로 지정이 되게끔 하는 문장이다. 허나 아래의 개념은 무시무시한 개념이다.

```
Constraint EMP_TAB_FK Foreign Key(Loc_ID)
References Location(Location_ID) On Delete Cascade
```
On Delete Cascade는 쉽게 이야기해서 연쇄 반응이나 한 사람이 잘 못되면 연대 책임을 지는 경우와 같다고 생각하면 된다. On delete Cascade를 작성하게 된다면 Location_ID가 320인 사원들의 정보는 일단 없애버리게끔 한다고 생각을 하면 된다.
(추가 개념) On delete 뭐시기 말고 deferrable initially deferred를 작성하게 된다면 트랜젝션이 완료하는 시점에 검사를 하도록 작성해주는 문장이니 알아두도록 하자.
```
Constraint EMP_TAB_FK Foreign Key(Loc_ID)
```

References Location(Location_ID) deferrable initially deferred

4) Primary Key

Primary Key에 대해서 더욱 알아보고 싶은 점이 있다. 이는 바로 기본키가 2개 이상인 경우에는 어떠한지에 대해서이다. 가령 예를 들어 다음과 같은 테이블로서 학생을 구분하는 테이블을 형성한다고 가정을 하자.

DeptID	StuID	Name	Birthdate
100	10	최종훈	1918/1/2
100	11	김재우	1911/10/1

이 색칠된 부분을 기본키로 구분을 하게 되는 테이블은 여태껏 생각을 못 해왔을 것이다. 허나 이를 체계적으로 관리하기 위해서는 2가지 방법으로 재작성을 하면 된다.

4-1) 컬럼과 함께 정의하기
Create table StuPlus
(DeptID Number(5) Constraint Stu_PK Primary Key, StuID Number(5) Constraint Stu_UK Unique, Name Varchar2(20), Birthdate Date Default To_Date('1990/1/1', 'YYYY/MM/DD'));
각 컬럼에서 제약 조건을 추가하는데 있어서 2개의 컬럼을 설정하는 방법에 대해서는 없다. 왜냐면 제약 조건은 한 컬럼에 하나씩 추가되는 개념이기 때문이다. 또한 무엇보다 중요한 점은 Primary Key(기본키)는 테이블에 추가하는 경우 한 테이블에 1개씩으로만 가지게끔 되어 있다. 같은 하늘 아래에 태양이 2개일 수가 없다는 의미랑 비스무리하다고 생각을 하면 된다. 그래서 4-1에서 제시한 방법으로서는 구현을 할 수 없기 때문에 4-2와 같은 방법으로 작성을 해보겠다.

4-2) 컬럼 뒤에 정의하기
Create table StuPlus
(DeptID Number(5), StuID Number(5), name Varchar2(20), Birthdate Date, Constraint Stu_ID_PK Primary Key(deptID, stuID));
마지막 문장에 제약 조건을 추가하는 방법으로서는 제약조건에서 컬럼이 2개이든 하나이든 간에 추가하는데 있어서 큰 어려움 없이 추가할 수 있다고 생각하면 된다. 허나 주의해야 하는 점으로서는 Oracle SQL, MySQL에서는 허용이 되지만, 참고로 **다른 체계의 SQL(오라클 이외)에서는 이를 구현 할 수 없다.**
(+) 물론 Constraint Stu_ID_UK Unique(deptID, stuID)으로도 구현이 가능하다는 점을 알아두도록 하자.

5) Alter Table문에 Constraint 적용

방금 4번에서 참조되었던 StuPlus 테이블을 참조하면서 아무런 제약 조건들이 안 들어 있다고 가정을 하고 생각을 해보면서 풀어보도록 하자.

5-1) name 컬럼에 UNIQUE 제약 조건 추가하기
Alter Table StuPlus
Add Constraint Stu_UK Unique(Name);
이처럼 작성을 하게 된다면 StuPlus 테이블에 Unique라는 제약조건을 추가를 할 수 있다. 허나 주의해야 할 점은 중복된 값이 존재하는 경우에는 이를 적용할 수 없으니 참조하도록 하자.

5-2) StuID 컬럼에 Primary Key 제약 조건 추가하기
Alter Table StuPlus
Add Constraint Stu_PK Primary Key(StuID);
Primary Key 라는 개념이 Not Null, Unique 2개의 개념이 모인 거라는 것은 이미 알고 있을 것이다. 그러나 중요한 점은 만일 중복된 값이 있거나 Null이 있는 경우에는 오류가 걸리니 null를 없애고 알아서 조정을 하고 제약 조건을 추가하길 바란다.

5-3) Name 컬럼에 Not Null 제약 조건 추가하기
여기서 주의해야 할 점은 Not Null 제약 조건을 추가하기 위해서는 위와 같은 문장으로 추가해서는 안 된다. 그래서 Null를 Not Null로 변경하는 것이기 때문에 MODIFY를 이용하도록 한다.
Alter Table StuPlus
Modify(Name Constraint Stu_NN Not NULL);
이처럼 재작성을 하게 되면 Name에 Null이 안 들어 있는 이상 제약 조건이 추가하게 된다.

5-4) drop을 통해서 제약 조건을 삭제하기
Alter Table StuPlus
drop Constraint Stu_NN;
이를 이용해서 작성을 하게 되면 drop Constraint를 통해서 제약 조건 이름을 입력하게 된다면 제약 조건이 삭제하게끔 도와준다. 제약 조건을 삭제하게 된다면 값을 추가하거나 변경하는데 있어서 무결성 제약 조건이 없어지게끔 해주니 참조하도록 하자. 또한 참조로 Modify로 constraint를 변경하는 것은 애석하게도 불가능하니 Add, Drop을 이용하도록 하자.