01

SELECT 문

▷ 테이블이나 뷰에 있는 데이터를 선택(조회)할 때 사용

▷ 구문

SELECT * *혹은* 컬럼
FROM [스키마.]테이블명 *혹은* [스키마.]뷰명
WHERE 조건
ORDER BY 컬럼;

● SELECT : 선택하고자 하는 컬럼명, 모든 컬럼을 조회하고 싶다면 *

● FROM : 선택할 테이블이나 뷰 명

● WHERE : 선택 조건, 여러 조건 기술 시에는 AND, OR로 연결

● ORDER BY : 조회 데이터 정렬 시, 정렬하고자 하는 컬럼명 기술

정렬하기 ORDER BY

출력된 데이터를 알아보기 쉽게 하기 위해서 특정 컬럼을 기준으로 정렬을 할 수 있는데 이때 사용하는 것이 ORDER BY 절

정렬 방법

- 기준 컬럼 뒤에 ASC를 붙이면(또는 생략시) 오름차순
- 기준 컬럼 뒤에 DESC를 붙이면 내림차순
- 숫자뿐 아니라 문자나 날짜도 같은 방법으로 정렬이 가능

정렬기준

- ORDER BY 뒤에 오는 컬럼
- 2개 이상 기입 시 순서대로
- 숫자 입력 시 컬럼의 순번

01

SELECT 문

▷ 뷰를 정의하는 서브 쿼리문에 WHERE절을 추가하여 기본 테이블 중 특정 조건에 만족하는 로우(행)만으로 구성된 뷰를 생성할 수 있다.

▷ 구문

CREATE OR REPLACE VIEW 뷰명 **AS**
SELECT * *혹은* 컬럼
FROM [스키마.]테이블명 *혹은* [스키마.]뷰명
WHERE 조건
WITH CHECK OPTION; [**WITH READ ONLY;**]

● **WITH CHECK OPTION** : 조건에 사용 되어진 컬럼 값은 뷰를 통해서는 변경이 불가능. 뷰를 통해서 접근가능한 데이터만 입력 가능.

● **WITH READ ONLY** : 기본 테이블의 어떤 컬럼에 대해서도 뷰를 통한 내용 수정을 불가능하게 만드는 옵션

02

INSERT 문

▷ 새로 데이터를 입력해 넣을 때 사용하는 문장

▶ 기본형태

▷ 구문

INSERT INTO [스키마.]테이블명 (컬럼1, 컬럼2, ...)
VALUES (값1, 값2, ...);

● 테이블명 다음 컬럼 순서, 타입이 VALUES 다음 괄호 안의 값 순서와 타입이 일치해야 함

02

INSERT 문

▶ 컬럼명 기술 생략

▷ 구문

INSERT INTO [스키마.]테이블명
VALUES (값1, 값2, ...);

- 테이블명 다음에 컬럼을 명시하지 않았으므로 테이블에 있는 모든 컬럼에 값을 넣는다는 의미
- VALUES 다음 값은 테이블에 있는 모든 컬럼 순서에 맞춰 넣어야 한다
- 컬럼 순서는 테이블 생성 시 명시한 컬럼 순서

02

INSERT 문

INSERT ~ SELECT 형태

구문

INSERT INTO [스키마.]테이블명 (컬럼1, 컬럼2, ...)
SELECT 문;

- VALUES 절과 함께 값을 일일이 명시하는 대신 SELECT 문을 사용
- 테이블명 다음 컬럼 순서와 SELECT 다음의 컬럼 순서, 타입이 맞아야 함
- 테이블명 다음 컬럼 리스트를 생략할 경우 이 테이블의 모든 컬럼에 값을 넣는다는 의미

03

UPDATE 문

▷ 테이블에 있는 기존 데이터를 수정하는 문장

▷ 구문

```
UPDATE [스키마.]테이블명
SET 컬럼1 = 변경값1,
      컬럼2 = 변경값2,
....
WHERE 조건;
```

● SET : 변경하고자 하는 컬럼과 그 값을 명시, 여러 컬럼을 갱신할 때는 콤마로 분리

● WHERE : 데이터를 갱신하는 조건, 이 조건에 맞는 데이터만 변경됨. WHERE 조건을 생략하면 테이블에 있는 모든 데이터가 변경된다.

04

MERGE 문

- 조건을 비교해 조건에 맞는 데이터가 없으면 INSERT, 있으면 UPDATE를 수행하는 문장
- 구문

```
MERGE INTO [스키마.]테이블명
    USING ( update나 insert 될 데이터 원천 )
        ON ( update될 조건 )

WHEN MATCHED THEN
        SET컬럼1 = 값1, 컬럼2 = 값2, ...
WHERE update조건
        DELETE WHERE update_delete조건

WHEN NOT MATCHED THEN
        INSERT ( 컬럼1, 컬럼2, ...) VALUES (값1, 값2,...)
        WHERE insert조건;
```

04

MERGE 문

- **MERGE INTO** : update나 insert 할 대상 테이블을 명시
- **USING** : 보통 다른 테이블이나 서브쿼리 형태로 변경할 데이터가 온다.
- **ON** : 변경 대상 테이블과 USING절에 명시한 테이블이나 서브쿼리와의 조인조건,
  이 조건에 맞는 데이터만 UPDATE 된다.
- **WHEN MATCHED THEN SET ~** : update 할 컬럼과 값 명시
- **WHERE** update조건 : 추가적으로 update될 조건을 명시, 생략 가능
- **DELETE WHERE** update_delete조건 : UPDATE될 값을 평가해서 조건에 맞는 데이터를 삭제
- **WHEN NOT MATCHED THEN ~ :** ON절에 명시한 조건에 일치하지 않을 경우,
  INSERT 할 컬럼과 값 명시

04

MERGE 문 Ex

- Ex3-3 신규테이블 생성
- Sales 테이블에서 2000년 10월 부터 2000년 12월까지 매출을 달성한 사원 번호(employee 테이블)를 입력.(INSERT 사용)
- employee 테이블에서 manager_id 가 146번인 사원을 찾아 ex3-3테이블의 사번과 일치하면 bonus_amt에 자신의 salary의 1%를 보너스로 갱신
- manager_id 가 146번인 사원 중에서 ex3-3테이블의 사번과 일치하지 않는 사원은 salary가 8000미만이면 신규 데이터로 입력. bonus_amt는 Salary의 0.1%

05

DELETE 문

- 테이블에 있는 데이터를 삭제하는 문장
- 구문

① 일반 구문
DELETE [FROM] [스키마.]테이블명
WHERE delete조건;

② 특정 파티션만 삭제할 경우의 구문
DELETE [FROM] [스키마.]테이블명 **PARTITION** (파티션명)
WHERE delete조건;

트랜젝션(Transaction)

- 오라클의 트랜젝션은 데이터의 일관성을 위해서 ALL-or Nothing 방식으로 처리한다.
- 즉, 여러 개의 명령어중 하나만 잘못되어도 모든 명령을 취소시켜서 데이터의 일관성을 유지한다.

- 트랜젝션 관리를 위해 제공하는 명령어는 두가지인데.

COMMIT와 ROLLBACK이다.

트랜젝션(Transaction) - COMMIT

- COMMIT 명령은 모든 작업을 정상 처리 완료하고 처리의 모든 과정을 확정하는 명령이다.

- 모든 트랜젝션의 처리과정을 데이터베이스에 반영하고 변경된 모든 내용을 영구 저장을 한다.

- COMMIT 명령어를 수행하면 하나의 트랜젝션 과정을 종료한다.

트랜젝션(Transaction) - ROLLBACK

- ROLLBACK 명령은 작업중 문제가 발생해서 트랜젝션 처리 과정에 발생한 변경 내용을 취소하는 명령이다.

- ROLLBACK은 트랜젝션으로 인한 하나의 묶음 처리가 시작되기 이전 상태로 되돌린다.

06

COMMIT, ROLLBACK, TRUNCATE

COMMIT

- 변경한 데이터를 데이터베이스에 최종적으로 반영
- 구문

COMMIT [WORK] [**TO SAVEPOINT** 세이브포인트명] ;

ROLLBACK

- 변경한 데이터를 변경 전 상태로 되돌림
- 구문

ROLLBACK [WORK] [**TO SAVEPOINT** 세이브포인트명] ;

06

COMMIT, ROLLBACK, TRUNCATE

TRUNCATE

- DELETE 처럼 테이블 데이터를 삭제
- 실행 시 영구적으로 데이터가 삭제되는 DDL 문으로 WHERE 조건은 붙을 수 없다.
- 구문

  TRUNCATE TABLE [스키마명.]테이블명;

테이블 변경 - UNUSED

- 컬럼 삭제 : 실제 테이블에서 컬럼을 삭제하는 것이 아닌 사용하지 않는다고 명령을 내릴 수 있다

- 실제 삭제하는 것보다 빠르게 삭제한 것과 같은 효과를 낼 수 있다
  - 삭제의 효율성을 위한 것으로 복원이 불가능하다.

```
ALTER TABLE dept20
SET UNUSED (BIRTH);
```

  - 숨겨진 UNUSED를 테이블로부터 완전히 삭제하는 명령은 다음과 같다

```
ALTER TABLE dept20
DROP UNUSED COLUMNS
```

07

의사컬럼

- 테이블의 컬럼처럼 동작하지만 실제로 테이블에 저장되지는 않는 컬럼
- **CONNECT_BY_ISCYCLE, CONNECT_BY_ISLEAF, LEVEL**

  계층형 쿼리에서 사용하는 의사컬럼
- **NEXTVAL, CURRVAL**

  시퀀스에서 사용하는 의사컬럼
- **ROWNUM,ROWID**

  ROWNM은 쿼리에서 반환되는 각 로우에 대한 순서값

  ROWID는 테이블에 저장된 각 로우가 저장된 주소값

08

연산자

- 수식연산자 : +, -, *, /
- 문자연산자 : ||
- 논리연산자 : >, <, >=, <=, =, <>, !=, ^=
- 집합연산자 : UNION, UNION ALL, INTERSECT, MINUS
- 계층형 쿼리 연산자 : PRIOR, CONNECT_BY_ROOT

09

표현식

▷ 한 개 이상의 값과 연산자 그리고 SQL 함수 등이 결합된 식

▶ CASE 표현식

▷ 구문

```
CASE WHEN 조건1 THEN 값1
     WHEN 조건2 THEN 값2
      ...
     ELSE 기타값
END
```

조건식

- 조건 혹은 조건식(Condition)은 한 개 이상의 표현식과 논리 연산자가 결합된 식
- TRUE, FALSE, UNKNOWN 3가지 타입을 반환

비교 조건식

- 논리 연산자나 ANY, SOME, ALL 키워드로 비교하는 조건식

논리 조건식

- AND, OR, NOT을 사용하는 조건식

NULL 조건식

- 특정 값이 NULL인지 여부를 체크하는 조건식
- IS NULL, IS NOT NULL

조건식

BETWEEN AND 조건식

- 범위에 해당되는 값을 찾을 때 사용

IN 조건식

- 조건절에 명시한 값이 포함된 건을 반환, ANY와 비슷

EXISTS 조건식

- IN과 비슷하지만 후행 조건절로 값의 리스트가 아닌 서브쿼리만 올 수 있다
- 또한 서브쿼리 내에서 조인조건이 있어야 한다

LIKE 조건식

- 문자열의 패턴을 검색할때 사용.

파일명 : 이름_chap03_ex1.txt

Quiz 1. ex3_6란 테이블을 만들고, 사원테이블(employees)에서 관리자사번이 124번이고 급여가 2000에서 3000 사이에 있는 사원의 사번, 사원명, 급여, 관리자사번을 입력하는 INSERT문을 작성해보자.

Quiz 2. 다음 문장을 실행해보자.

```
DELETE ex3_3;

INSERT INTO ex3_3 (employee_id)
SELECT e.employee_id
  FROM employees e, sales s
 WHERE e.employee_id = s.employee_id
   AND s.SALES_MONTH BETWEEN '200010' AND '200012'
 GROUP BY e.employee_id;

COMMIT;
```

관리자사번(manager_id)이 145번인 사원을 찾아 위 테이블에 있는 사원의 사번과 일치하면 보너스 금액(bonus_amt)에 자신의 급여의 1%를 보너스로 갱신하고, ex3_3 테이블에 있는 사원의 사번과 일치하지 않는 사원을 신규 입력 (이때 보너스 금액은 급여의 0.5%로 한다) 하는 MERGE 문을 작성해 보자.

Quiz 3. 사원테이블(employees)에서 커미션(commission_pct) 값이 없는 사원의 사번과 사원명을 추출하는 쿼리를 작성해보자.

Quiz 4. 아래의 쿼리를 논리연산자로 변환해보자.

```
SELECT employee_id, salary
  FROM employees
WHERE salary BETWEEN 2000 AND 2500
ORDER BY employee_id;
```

Quiz 5. 다음의 두 쿼리를 ANY, ALL을 사용해서 동일한 결과를 추출하도록 변경해보자.

```
SELECT employee_id, salary
  FROM employees
WHERE salary IN (2000, 3000, 4000)
ORDER BY employee_id;
```

```
SELECT employee_id, salary
  FROM employees
WHERE salary NOT IN (2000, 3000, 4000)
ORDER BY employee_id;
```

파일명 : 이름_chap03_ex2.txt (ex_user로 접속, EMPLOYEE테이블)

1. Employee테이블의 구조만 복사하여 EMP_INSERT란 빈 테이블을 만드세요.

2. 임의의 데이터를 EMP_INSERT테이블에 추가하되 SYSDATE를 이용해서 입사일을 오늘로 입력하세요

3. EMP_INSERT 테이블에 옆사람을 추가하되 SYSDATE를 이용해서 입사일을 오늘로 입력하세요

4. Employee테이블의 구조와 내용을 복사하여 EMP_COPY란 이름의 테이블을 만드세요.

5. 사원번호가 7788인 사원의 부서번호를 10으로 수정하세요.

6. Department 테이블의 구조와 내용을 복사하여 DEPT_COPY란 이름의 테이블을 만드세요

7. DEPT_COPY테이블에서 부서명이 RESEARCH인 부서를 제거하세요.

8. DEPT_COPY테이블에서 부서번호가 10이거나 40인 부서를 제거하세요.