

[주의] 이 자료는 참고자료를 바탕으로 작성되었으며 본 수업 이외의 곳에서 활용 또는 배포를 금합니다.

Oracle DB 프로그래밍

(NCS기반 DB구현과 SQL활용)

(버전 v0.7)



목차

A 훈련 준비	8
A.1 학습 진도표	8
A.2 실습 환경	8
A.3 참고 자료	8
1 데이터베이스 개념과 오라클 설치.....	9
1.1 데이터베이스와 데이터베이스 관리시스템.....	9
1.1.1 파일 시스템의 문제점.....	9
1.1.2 데이터베이스의 개념.....	9
1.1.3 데이터베이스의 특징.....	10
1.1.4 데이터베이스 시스템의 구성.....	10
1.1.5 관계형 데이터베이스 관리 시스템.....	11
1.2 오라클 설치 및 설정	12
1.2.1 오라클 설치	12
1.2.2 개발 도구 소개	13
1.2.3 샘플 스키마	19
1.2.4 과제 스키마	20
1.2.5 추가 스키마 (마당서점).....	21
1.3 SQL과 SQL*Plus의 개념	22
1.3.1 사용 용도에 따른 SQL.....	22
1.3.2 가장 기본이 되는 SELECT문.....	23
1.3.3 DML(Data Manipulation Language).....	23
1.3.4 TCL(Transaction Control Language).....	24
1.3.5 DDL(Data Definition Language).....	24
1.3.6 DCL(Data Control Language).....	24
1.3.7 SQL*Plus란?	25
1.4 SQL*Plus 로그인	25
1.4.1 Command 환경에서 SQL*Plus 로그인.....	25
1.4.2 시스템 권한을 갖는 데이터베이스 관리자.....	25
1.4.3 SQL*Plus 로그인에 실패할 경우 해결 방법.....	26
2 SQL의 기본	27
2.1 데이터 덱서너리 TAB	27
2.2 테이블 구조를 살펴보기 위한 DESC.....	27
2.3 오라클의 데이터형	27
2.3.1 NUMBER	27
2.3.2 DATE	28
2.3.3 CHAR	28
2.3.4 VARCHAR2	28
[꿀팁] Boolean 타입	28
2.4 데이터를 조회하기 위한 SELECT문.....	29
2.5 산출 연산자	29
2.6 NULL도 데이터!	29
2.7 컬럼 이름에 별칭 지정하기	30
2.7.1. AS로 컬럼에 별칭 부여하기.....	30
2.7.2. ""로 별칭 부여하기.....	30
2.7.3. 별칭으로 한글 사용하기.....	30
2.8 Concatenation 연산자의 정의와 사용.....	31
2.9 DISTINCT 키워드	31
3 SQL*Plus 명령어	32
3.1 SQL*Plus 명령어의 개념	32
3.2 SQL*Plus 편집 명령	32
3.3 SQL*Plus 파일 명령어	32

3.3.1	파일에 내용을 메모장에서 편집하게 하는 EDIT(ED)	33
3.3.2	DOS 프롬프트로 나가게 하는 HOST	33
3.3.3	사용자가 현재 수행 중인 쿼리문을 저장하는 SAVE	33
3.3.4	SQL 파일에 저장된 명령어를 실행하는 @	33
3.3.5	갈무리 기능을 하는 SPOOL	34
3.3.6	저장한 SQL 명령어를 가져오는 GET	34
3.4	시스템 변수 조작을 위한 SET 명령어	34
3.4.1	컬럼 제목의 출력 여부를 결정하는 HEADING(HEA) 변수	34
3.4.2	라인 당 출력할 문자의 수를 결정하는 LINESIZE 변수	34
3.4.3	페이지 당 출력할 라인의 수를 결정하는 PAGESIZE 변수	34
3.4.4	컬럼에 저장된 데이터의 출력 형식 변경을 위한 COLUMN FORMAT	35
4	SELECT로 특정 데이터를 추출하기	36
4.1	WHERE 조건과 비교 연산자	36
4.1.1	비교 연산자	36
4.1.2	문자 데이터 조회	36
4.1.3	날짜 데이터 조회	36
4.2	논리 연산자	37
4.2.1	AND 연산자	37
4.2.2	OR 연산자	37
4.2.3	NOT 연산자	37
4.3	BETWEEN AND 연산자	37
4.4	IN 연산자	38
4.5	LIKE 연산자와 와일드카드	38
4.5.1	와일드카드(%) 사용하기	38
4.5.2	와일드카드(_) 사용하기	39
4.5.3	NOT LIKE 연산자	39
4.6	NULL을 위한 연산자	39
4.7	정렬을 위한 ORDER BY절	40
4.7.1	오름차순 정렬을 위한 ASC	40
4.7.2	내림차순 정렬을 위한 DESC	40
5	SQL 주요 함수	42
5.1	DUAL 테이블과 SQL 함수 분류	42
5.2	숫자 함수	42
5.2.1	절댓값 구하는 ABS 함수	42
5.2.2	소수점 아래를 버리는 FLOOR 함수	42
5.2.3	특정 자릿수에서 반올림하는 ROUND 함수	43
5.2.4	특정 자릿수에서 잘라내는 TRUNC 함수	43
5.2.5	나머지 구하는 MOD 함수	43
5.3	문자 처리 함수	43
5.3.1	UPPER 함수	43
5.3.2	LOWER 함수	43
5.3.3	INITCAP 함수	44
5.3.4	LENGTH 함수	44
5.3.5	LENGTHB 함수	44
5.3.6	SUBSTR, SUBSTRB 함수	44
5.3.7	INSTR 함수	45
5.3.8	INSTRB 함수	45
5.3.9	LPAD/RPAD 함수	45
5.3.10	LTRIM와 RTRIM 함수	45
5.3.11	TRIM 함수	45
5.3.12	REPLACE 함수	46
5.4	날짜 함수	46

5.4.1	현재 날짜를 반환하는 SYSDATE 함수	46
5.4.2	날짜 연산	46
5.4.3	특정 기준으로 반올림하는 ROUND 함수	46
5.4.4	특정 기준으로 버리는 TRUNC 함수	47
5.4.5	두 날짜 사이의 날수를 구하는 MONTHS_BETWEEN 함수	47
5.4.6	개월 수를 더하는 ADD_MONTHS 함수	47
5.4.7	해당 요일의 가장 가까운 날짜를 반환하는 NEXT_DAY 함수	47
5.4.8	해당 달의 마지막 날짜를 반환하는 LAST_DAY 함수	48
5.5	형 변환 함수	48
5.5.1	문자형으로 변환하는 TO_CHAR 함수	48
5.5.2	날짜형으로 변환하는 TO_DATE 함수	49
5.5.3	숫자형으로 변환하는 TO_NUMBER 함수	49
5.6	NULL을 다른 값으로 변환하는 NVL 함수	49
5.7	선택을 위한 DECODE 함수	49
5.8	조건에 따라 서로 다른 처리가 가능한 CASE 함수	50
	[과제] 과제-05-01.TXT	50
	[과제] 과제-05-02.TXT	51
6.	그룹 쿼리와 집합 연산자	52
6.1.	기본 집계 함수	52
6.1.1.	합계를 구하는 SUM 함수	52
6.1.2.	평균 구하는 AVG 함수	52
6.1.3.	최대값 구하는 MAX, 최소값 구하는 MIN 함수	52
6.1.4.	그룹 함수와 단순 컬럼	52
6.1.5.	로우 개수 구하는 COUNT 함수	53
6.2.	GROUP BY절	53
6.3.	HAVING 조건	54
6.4.	집합 연산자	54
6.4.1	UNION	54
6.4.2	UNION ALL	55
6.4.3	INTERSECT	55
6.4.4	MINUS	55
6.4.5	집합 연산자 제한사항	56
7	조인	58
7.1	조인의 필요성	58
7.2	Cross Join	58
7.3	Equi Join	58
7.4	Non-Equi Join	59
7.5	Self Join	60
7.6	Outer Join	60
7.7	ANSI Join	61
7.7.1	ANSI Cross Join	61
7.7.2	ANSI Inner Join	61
7.7.3	ANSI Outer Join	61
8	서브 쿼리	63
8.1	서브 쿼리의 기본 개념	63
8.2	단일행 서브 쿼리	63
8.3	다중행 서브 쿼리	63
8.3.1	IN 연산자	64
8.3.2	ALL 연산자	64
8.3.3	ANY 연산자	64
8.3.4	EXISTS 연산자	64
9	테이블 구조 생성, 변경 및 삭제하는 DDL	67

9.1 테이블 구조를 정의하는 CREATE TABLE.....	67
9.1.1 데이터형	67
9.1.2 식별자 명명 규칙	67
9.1.3 테이블의 구조만 복사하기.....	69
9.2 테이블 구조를 변경하는 ALTER TABLE.....	69
9.2.1 새로운 컬럼 추가하기.....	69
9.2.2 기존 컬럼 속성 변경하기.....	70
9.2.3 기존 컬럼 삭제하기.....	70
9.2.4 SET UNUSED 옵션 적용하기.....	70
9.3 테이블 구조를 삭제하는 DROP TABLE.....	71
9.4 테이블의 모든 로우를 제거하는 TRUNCATE.....	71
9.5 테이블명을 변경하는 RENAME	71
9.6 데이터 디렉터리와 데이터 디렉터리 뷰.....	72
9.6.1 USER_ 데이터 디렉터리.....	72
9.6.2 ALL_ 데이터 디렉터리.....	72
9.6.3 DBA_ 데이터 디렉터리 뷰.....	72
10 테이블의 내용 추가, 수정, 삭제하는 DML.....	74
10.1 테이블에 새로운 행을 추가하는 INSERT문.....	74
10.1.1 INSERT 구문의 오류 발생 예.....	74
10.1.2 컬러명을 생략한 INSERT 구문.....	75
10.1.3 NULL 값을 삽입하는 다양한 방법.....	75
10.1.4 서브 쿼리로 데이터 삽입하기.....	76
10.2 다중 테이블에 다중행 입력하기.....	76
10.3 테이블의 내용을 수정하기 위한 UPDATE문.....	77
10.3.1 테이블의 모든 행 변경.....	77
10.3.2 테이블의 특정 행만 변경.....	78
10.3.3 테이블에서 2개 이상의 컬럼값 변경.....	78
10.3.4 서브 쿼리를 이용한 데이터 수정하기.....	79
10.4 테이블의 불필요한 행을 삭제하기 위한 DELETE문.....	79
10.4.1 DELETE문을 이용한 행 삭제.....	79
10.4.2 조건을 제시하여 특정 행만 삭제하는 법.....	79
10.4.3 서브 쿼리를 이용한 데이터 삭제.....	79
10.5 테이블을 합병하는 MERGE	80
11. 트랜잭션 관리	82
11.1. 트랜잭션	82
11.2. COMMIT과 ROLLBACK	82
11.2.1. COMMIT과 ROLLBACK의 개념.....	82
11.3. 자동 커밋	84
11.4. 트랜잭션을 작게 분할하는 SAVEPOINT.....	84
12. 데이터 읽기 일관성과 락	86
12.1. 데이터 읽기의 일관성과 락.....	86
12.2. 데드 락	87
12.3. SET UNUSED	87
12.4. DDL 명령의 롤백	88
12.5. TRUNCATE와 DELETE의 차이	88
13. 데이터 무결성을 위한 제약 조건.....	90
13.1. 무결성 제약 조건의 개념과 종류.....	90
13.1.1. 데이터 무결성 제약 조건이란?.....	90
13.1.2. 무결한 데이터의 5가지 제약 조건.....	90
13.1.3. 제약 조건 확인하기.....	90
13.1.4. 제약 조건 살피기.....	90
13.2. 필수 입력을 위한 NOT NULL 제약 조건.....	91

13.3.	유일한 값만 허용하는 UNIQUE 제약 조건	91
13.4.	컬럼 레벨로 제약 조건명을 명시하여 제약 조건 설정하기	92
13.5.	데이터 구분을 위한 PRIMARY KEY 제약 조건	92
13.6.	참조 무결성을 위한 FOREIGN KEY 제약 조건	93
13.7.	CHECK와 DEFAULT의 제약 조건	96
13.8.	테이블 레벨 방식으로 제약 조건 지정하기	96
13.9.	제약 조건 변경하기	97
13.9.1.	제약 조건 추가하기	97
13.9.2.	MODIFY로 NOT NULL 제약 조건 추가하기	97
13.9.3.	제약 조건 제거하기	98
13.10.	제약 조건의 비활성화와 CASCADE	98
13.10.1.	제약 조건에 의해 작업이 어려운 경우	98
13.10.2.	제약 조건의 비활성화	98
13.10.3.	제약 조건의 활성화	98
13.10.4.	CASCADE 옵션	99
	[과제] 과제-13-01.TXT	99
14.	가상 테이블인 뷰	101
14.1.	뷰의 기본 다루기	101
14.1.1.	뷰의 기본 테이블	101
14.1.2.	뷰 정의하기	101
14.2.	뷰 고급 다루기	102
14.2.1.	뷰를 사용하는 이유	102
14.2.2.	단순뷰와 복합뷰	102
14.2.3.	뷰 삭제 알아보기	103
14.3.	뷰 생성에 사용되는 다양한 옵션	103
14.3.1.	뷰 수정을 위한 OR REPLACE 옵션	103
14.3.2.	기본 테이블 없이 뷰를 생성하기 위한 FORCE 옵션	103
14.3.3.	조건 컬럼값을 변경하지 못하게 하는 WITH CHECK OPTION	104
14.3.4.	기본 테이블 변경을 막는 WITH READ ONLY 옵션 WITH CHECK OPTION 비교	105
14.4.	뷰 활용하여 Top-N 구하기	105
14.4.1.	ROWNUM 컬럼 성격 파악하기	105
14.4.2.	인라인 뷰로 구하는 TOP-N의 개념	105
15.	시퀀스	107
15.1.	시퀀스의 개념 이해와 시퀀스 생성	107
15.2.	시퀀스 관련 데이터 디렉터리	107
15.3.	시퀀스 값을 알아보는 CURRVAL과 NEXTVAL	108
15.4.	시퀀스 실무에 적용하기	108
15.5.	시퀀스 제거와 수정	108
15.5.1.	시퀀스를 삭제하는 방법	108
15.5.2.	시퀀스를 수정하는 방법	109
	[과제] 과제-15-01.TXT	109
16.	인덱스	110
16.1.	인덱스의 개요	110
16.1.1.	인덱스란?	110
16.1.2.	인덱스 정보 조회	110
16.1.3.	조회 속도 비교하기	110
16.1.4.	인덱스 생성 및 제거하기	111
16.1.5.	인덱스를 사용해야 하는 경우 판단하기	111
16.2.	인덱스의 물리적인 구조와 재생성	112
16.2.1.	B-트리 인덱스 구조	112
16.2.2.	B-트리 인덱스의 추가, 삭제	112
16.3.	인덱스의 종류 살피기	113

16.3.1	고유와 비고유 인덱스.....	113
16.3.2	결합 인덱스 정의하기.....	113
16.3.3	함수 기반 인덱스 정의하기.....	114
17	사용자 관리	115
17.1	데이터베이스 보안을 위한 권한.....	115
17.1.1	보안을 위한 데이터베이스 관리자.....	115
17.1.2	권한의 역할과 종류.....	115
17.2	사용자 생성하기	115
17.3	권한을 부여하는 GRANT 명령어.....	116
17.3.1	테이블 스페이스 확인하기.....	117
17.3.2	WITH ADMIN OPTION.....	118
17.4	객체 권한	118
17.4.1	객체와 권한 설정.....	118
17.4.2	다른 유저의 객체 접근하기.....	118
17.4.3	스키마 알아보기	119
17.4.4	사용자에게 부여된 권한 조회.....	119
17.4.5	사용자에게서 권한을 뺏기 위한 REVOKE 명령어.....	119
17.4.6	WITH GRANT OPTION.....	120
	[과제] 과제-17-01.TXT	120
18	데이터베이스 롤 권한 제어	121
18.1	롤의 정의와 종류	121
18.1.1	롤의 정의	121
18.1.2	사전에 정의된 롤의 종류.....	121
18.1.3	롤 관련 데이터 디렉터리.....	122
18.2	사용자 롤 정의	122
18.3	롤 회수하기	123
18.4	롤의 장점	123
19	동의어	125
19.1	동의어의 개념과 종류	125
19.2	동의어 생성 및 제거	125
20	PL/SQL 기초	127
20.1	PL/SQL의 구조	127
20.2	변수 선언과 대입문	127
20.2.1	대입문으로 변수에 값 지정하기.....	128
20.2.2	스칼라 변수/레퍼런스 변수.....	129
20.2.3	PL/SQL에서 SELECT INTO문.....	129
20.2.4	PL/SQL 테이블 TYPE.....	130
20.2.5	PL/SQL RECORD TYPE.....	132
20.3	PL/SQL의 제어문	133
20.3.1	IF~THEN~END IF	133
20.3.2	IF~THEN~ELSE~END IF.....	135
20.3.3	IF~THEN~ELSEIF~ELSE~END IF.....	136
20.3.4	BASIC LOOP문	137
20.3.5	FOR LOOP문	138
20.3.6	WHILE LOOP문	139

A 훈련 준비

A.1 학습 진도표

학습 목표	학습 내용	학습 소요시간(일)
데이터베이스 구현	1. 데이터베이스 개념과 오라클 설치	
	2. SQL의 기본	
	3. SQL*PLUS 명령어	
	4. SELECT로 특정 데이터 추출하기	
	5. SQL 주요 함수	
	6. 그룹 함수	
	7. 조인	
	8. 서브 쿼리	
	9. 테이블 구조 생성, 변경 및 삭제하는 DDL	
	10. 테이블의 내용을 추가, 수정, 삭제하는 DML	
SQL 활용	11. 트랜잭션 관리	
	12. 데이터 읽기의 일관성과 락	
	13. 데이터 무결성을 위한 제약조건	
	14. 가상 테이블인 뷰	
	15. 시퀀스	
	16. 인덱스	
	17. 사용자 관리	
	18. 데이터베이스 롤 권한 제어	
	19. 동의어	
	20. PL/SQL 기초	
	21. 저장 프로시저, 저장 함수, 커서, 트리거	
	22. 패키지	
	23. 데이터 모델링	
총계		

A.2 실습 환경

- Oracle Database 11g Express Edition: OracleXE112_Win64.zip
- Java (JDK) for Developers: jdk-8u111-windows-x64.exe
- Eclipse IDE: eclipse-jee-neon-1a-win32-x86_64.zip

A.3 참고 자료

- 오라클 SQL과 PL/SQL을 다루는 기술 (길벗 출판, 홍형경 지음)
- 오라클로 배우는 데이터베이스 개론과 실습 (한빛아카데미 출판, 박우장외 지음)

1 데이터베이스 개념과 오라클 설치

- 유용한 데이터의 집합
- 데이터베이스에 저장된 정보는 검색, 수정, 삭제에 용이

1.1 데이터베이스와 데이터베이스 관리시스템

1.1.1 파일 시스템의 문제점

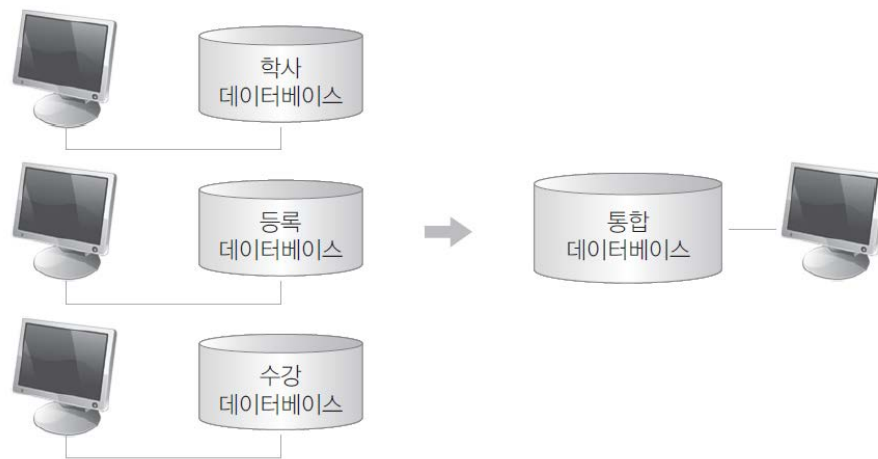
- 파일시스템에 다음과 같은 문제점이 제기되면서 새로운 시스템(DBMS)에 대한 요구가 제기되었다.

구분	파일 시스템	DBMS
데이터 중복	데이터를 파일 단위로 저장하므로 중복 가능	DBMS를 이용하여 데이터를 공유하기 때문에 중복 가능성 낮음
데이터 일관성	데이터의 중복 저장으로 일관성이 결여됨	중복 제거로 데이터의 일관성이 유지됨
데이터 독립성	데이터 정의와 프로그램의 독립성 유지 불가능	데이터 정의와 프로그램의 독립성 유지 가능
관리 기능	보통	데이터 복구, 보안, 동시성 제어, 데이터 관리 기능 등을 수행
프로그램 개발 생산성	나쁨	짧은 시간에 큰 프로그램을 개발할 수 있음
기타 장점	보통	데이터 무결성 유지, 데이터 표준 준수 용이

[표] 파일 시스템과 DBMS의 비교

1.1.2 데이터베이스의 개념

- 데이터베이스의 개념은 다음과 같이 네 가지로 설명할 수 있다.
 - 통합된 데이터(integrated data)
 - 데이터를 통합하는 개념으로, 각자 사용하던 데이터의 중복을 최소화하여 중복으로 인한 데이터 불일치 현상을 제거
 - 저장된 데이터(stored data)
 - 문서로 보관된 데이터가 아니라 디스크, 테이프 같은 컴퓨터 저장장치에 저장된 데이터를 의미
 - 운영 데이터(operational data)
 - 조직의 목적을 위해 사용되는 데이터를 의미한다. 즉 업무를 위한 검색을 할 목적으로 저장된 데이터
 - 공용 데이터(shared data)
 - 한 사람 또는 한 업무를 위해 사용되는 데이터가 아니라 공동으로 사용되는 데이터를 의미

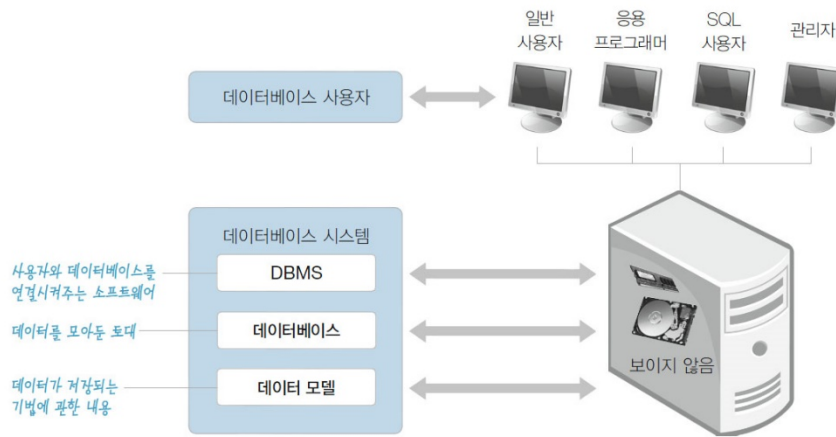


1.1.3 데이터베이스의 특징

- 데이터베이스의 특징은 다음과 같이 네 가지로 설명할 수 있다.
 - 실시간 접근성(real time accessibility)
 - 데이터베이스는 실시간으로 서비스된다. 사용자가 데이터를 요청하면 몇 시간이나 몇 일 뒤에 결과를 전송하는 것이 아니라 수 초 내에 결과를 서비스한다.
 - 지속적인 변화(continuous change)
 - 데이터베이스에 저장된 내용은 어느 한 순간의 상태를 나타내지만, 데이터 값은 시간에 따라 항상 바뀐다. 데이터베이스는 삽입(insert), 삭제(delete), 수정(update) 등의 작업을 통하여 바뀐 데이터 값을 저장한다.
 - 동시 공유(concurrent sharing)
 - 데이터베이스는 서로 다른 업무 또는 여러 사용자에게 동시에 공유된다. 동시(concurrent)는 병행이라고도 하며, 데이터베이스에 접근하는 프로그램이 여러 개 있다는 의미다.
 - 내용에 따른 참조(reference by content)
 - 데이터베이스에 저장된 데이터는 데이터의 물리적인 위치가 아니라 데이터 값에 따라 참조된다

1.1.4 데이터베이스 시스템의 구성

- 데이터베이스 시스템은 각 조직에서 사용하던 데이터를 모아서 통합하고 서로 공유하여 생기는 장점을 이용하는 시스템이다.
- 데이터베이스(DataBase)
 - 기업이 지속적으로 유지 관리해야 하는 데이터의 집합
- 데이터베이스 관리 시스템(DataBase Management System)
 - 방대한 양의 데이터를 편리하게 저장하고 효율적으로 관리하고 검색할 수 있는 환경을 제공해주는 시스템 소프트웨어
 - 데이터를 공유하여 정보의 체계적인 활용을 가능하게 한다.
 - 응용 프로그램과 데이터베이스의 중재자로서 모든 응용 프로그램들이 데이터베이스를 공유할 수 있게끔 관리해 주는 소프트웨어 시스템이다.
- 데이터 모델
 - 데이터가 저장되는 기법에 관한 내용, 예) 가전제품에 붙여진 모델 번호가 제품의 특성을 나타내듯이 데이터 모델은 데이터가 저장되는 스타일을 나타낸다.



[그림] 데이터베이스 시스템의 구성요소

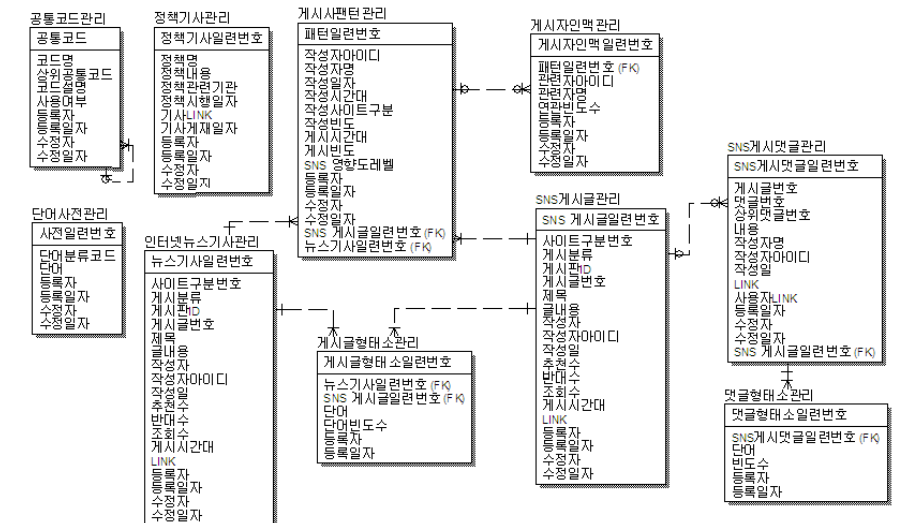
1.1.5 관계형 데이터베이스 관리 시스템

- 관계형 데이터베이스 관리시스템(RDBMS: Relational DataBase Management System)은 가장 일반적인 형태의 DBMS 이다.
- 관계형 데이터베이스 정보를 테이블 형태로 저장한다. 테이블은 2차원 형태의 표처럼 볼 수 있도록 로우(ROW:행)와 칼럼(COLUMN:열)으로 구성한다.

테이블 이름 : DEPT

칼럼 이름	DEPTNO	DNAME	LOC
로우	10	ACCOUNTING	NEW YORK
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

■ ERD(Entity Relationship Diagram)



- RDBMS의 종류: Oracle, MS SQL Server, DB2, Sybase, Infomix, MySQL, MS Access, Tibero,

Postgres, MongoDB, MariaDB, NoSQL 등이다.

[꿀팁] DB-Engines Ranking (<https://db-engines.com/en/ranking>)

348 systems in ranking, November 2018

Rank			DBMS	Database Model	Score		
Nov 2018	Oct 2018	Nov 2017			Nov 2018	Oct 2018	Nov 2017
1.	1.	1.	Oracle +	Relational DBMS	1301.11	-18.16	-58.94
2.	2.	2.	MySQL +	Relational DBMS	1159.89	-18.22	-162.14
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1051.55	-6.78	-163.53
4.	4.	4.	PostgreSQL +	Relational DBMS	440.24	+20.85	+60.33
5.	5.	5.	MongoDB +	Document store	369.48	+6.30	+39.01
6.	6.	6.	IBM Db2 +	Relational DBMS	179.87	+0.19	-14.19
7.	7.	9.	Redis +	Key-value store	144.17	-1.12	+22.99
8.	8.	10.	Elasticsearch +	Search engine	143.46	+1.13	+24.05
9.	9.	7.	Microsoft Access	Relational DBMS	138.44	+1.64	+5.12
10.	11.	11.	SQLite +	Relational DBMS	122.71	+5.96	+9.95
11.	10.	8.	Cassandra +	Wide column store	121.74	-1.64	-2.47
12.	13.	15.	Splunk	Search engine	80.37	+3.48	+15.50
13.	12.	12.	Teradata +	Relational DBMS	79.31	+0.67	+1.07
14.	14.	18.	MariaDB +	Relational DBMS	73.25	+0.12	+17.96
15.	16.	19.	Hive +	Relational DBMS	64.57	+3.47	+11.32
16.	15.	13.	Solr	Search engine	60.87	-0.44	-8.28
17.	17.	16.	HBase +	Wide column store	60.41	-0.26	-3.15
18.	18.	14.	SAP Adaptive Server +	Relational DBMS	56.57	-2.00	-10.47

1.2 오라클 설치 및 설정

1.2.1 오라클 설치

- OTN 가입 : <http://www.oracle.com/kr>
- Oracle Database XE 11g Release 2 다운로드
 - <http://www.oracle.com/technology/software/products/database/index.html>
- Oracle Database XE 11g Release 2 설치
 - 전역 데이터베이스 이름(SID): **XE**
 - 데이터베이스 암호: "sys"로 설정한다.
 - 설치 디렉토리: **C:\prod\oraclexe**
- 주요 오라클 서비스 (Microsoft Windows 버전인 경우)
 - OracleServiceXE : OracleService + SID명 형태로 구성된 서비스로 설치한 오라클 기본 서비스이며 오라클 사용 시 반드시 맨 먼저 시작되어야 한다.
 - OracleXETNSListener : 리스너 서비스로 이것 역시 반드시 시작되어야 한다. 리스너란 네트워크를 통해 클라이언트(오라클을 사용하려는 사용자)와 오라클 서버와의 연결을 담당하는 관리 프로그램이다.

[꿀팁] 오라클 네트워크 접속 설정

```
// 1. d:\oracle\app\product\11.2.0\dbhome_1\NETWORK\ADMIN\listener.ora 파일 설정
# listener.ora Network Configuration File: D:\oracle\app\product\11.2.0\dbhome_1\network\admin\listener.ora
# Generated by Oracle configuration tools.

SID_LIST_LISTENER =
(SID_LIST =
(SID_DESC =
(SID_NAME = CLRExtProc)
(ORACLE_HOME = D:\oracle\app\product\11.2.0\dbhome_1)
(PROGRAM = extproc)
```

```

        (ENVS = "EXTPROC_DLLS=ONLY:D:\oracle\app\product\11.2.0\dbhome_1\bin\oraclr11.dll")
    )
)

LISTENER =
(DESCRIPTION_LIST =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    )
)

ADR_BASE_LISTENER = D:\oracle\app

// 2. d:\oracle\app\product\11.2.0\dbhome_1\NETWORK\ADMIN\tnsnames.ora 파일 설정
# tnsnames.ora Network Configuration File: D:\oracle\app\product\11.2.0\dbhome_1\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

LISTENER_MYORACLE =
(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))

ORACLR_CONNECTION_DATA =
(DESCRIPTION =
    (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
    (CONNECT_DATA =
        (SID = CLRExtProc)
        (PRESENTATION = RO)
    )
)

MYORACLE =
(DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = myoracle)
    )
)

ORCL =
(DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = myoracle)
    )
)

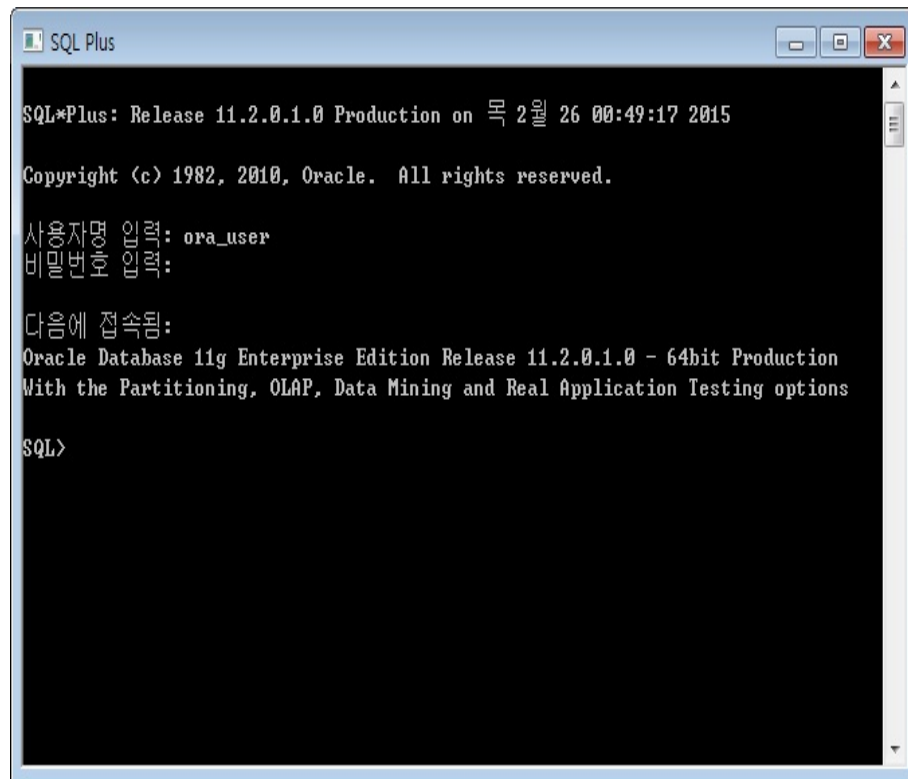
```

1.2.2 개발 도구 소개

- 오라클 설치 후 SQL이나 PL/SQL을 실행하고 결과를 보려면 개발도구가 필요함
- 대표적인 개발 도구로는 SQL Plus, 토드, 오렌지, PL/SQL Developer, SQL Developer 등이 있음

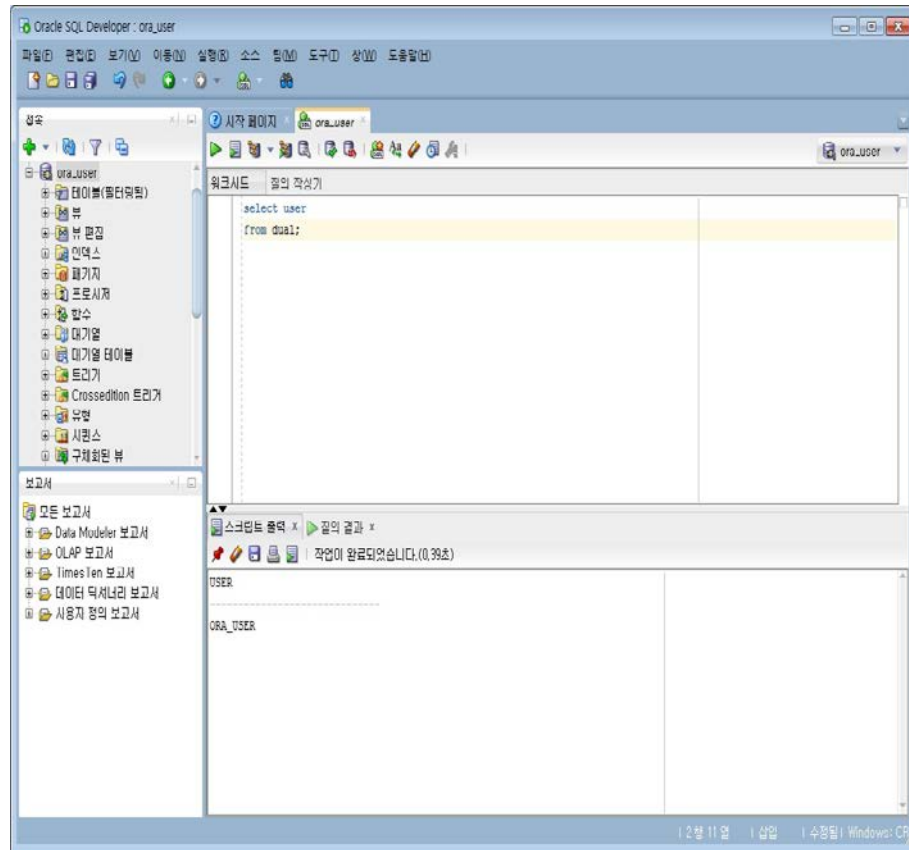
(1) SQLPLUS

- 오라클에 내장된 톨로 윈도우의 명령창 화면과 유사
- SQL, PL/SQL 뿐만 아니라 오라클 내부적인 명령어도 사용 가능



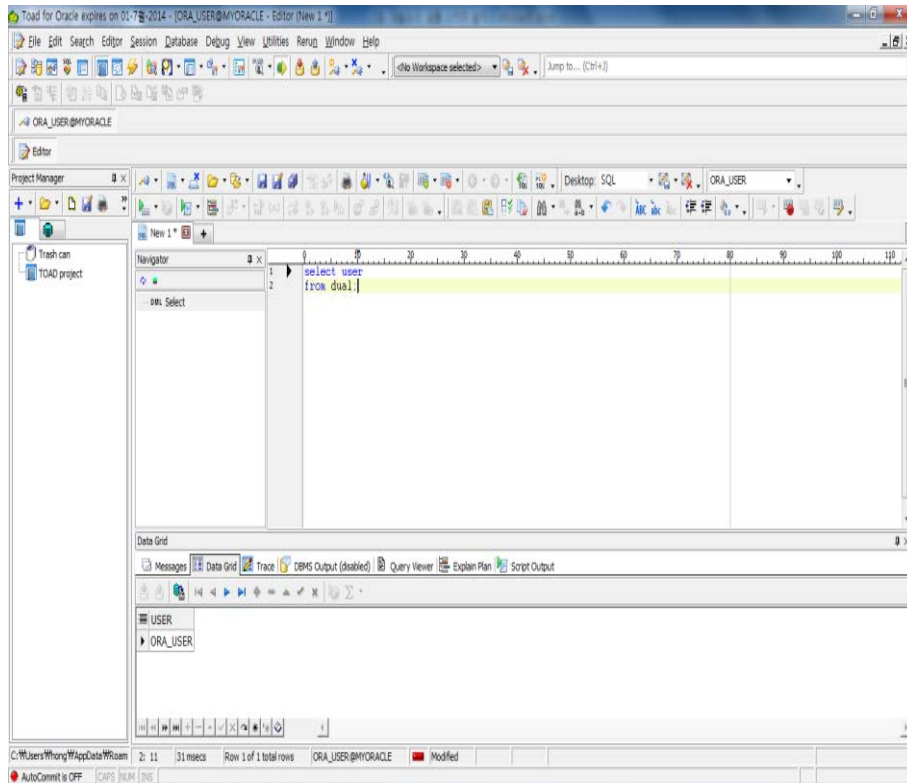
(2) SQL Developer

- 오라클에서 제공하는 무료 개발도구
- 다운로드: <http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>



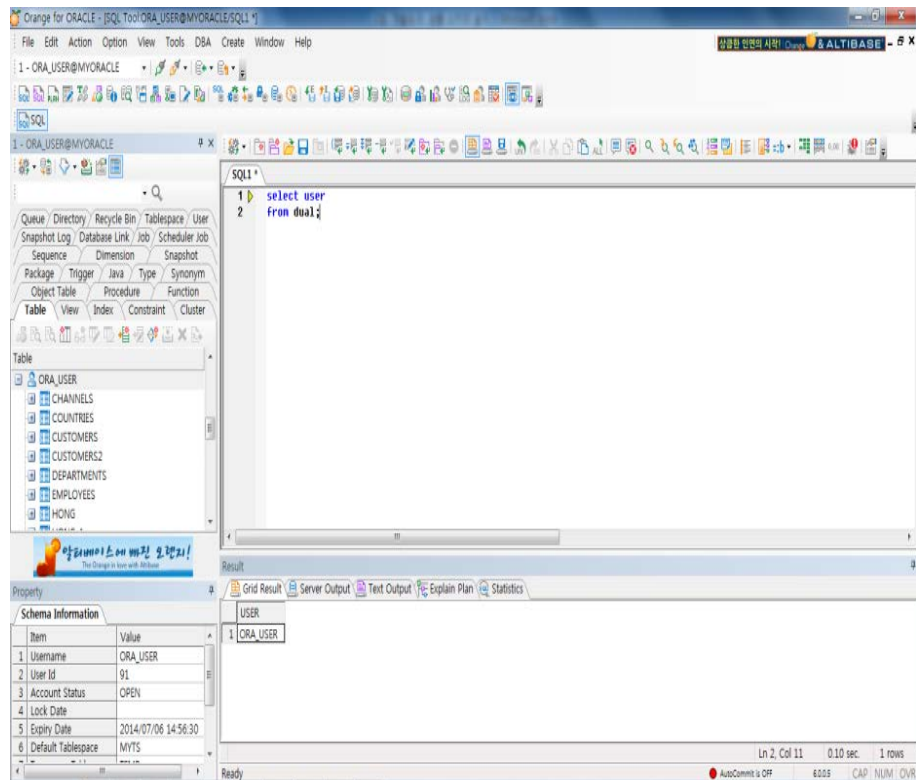
(3) Toad

- 전 세계적으로 개발자들 사이에 가장 많이 사용됨
- 상용제품이나 프리웨어 버전도 존재
- <http://www.toadworld.com>



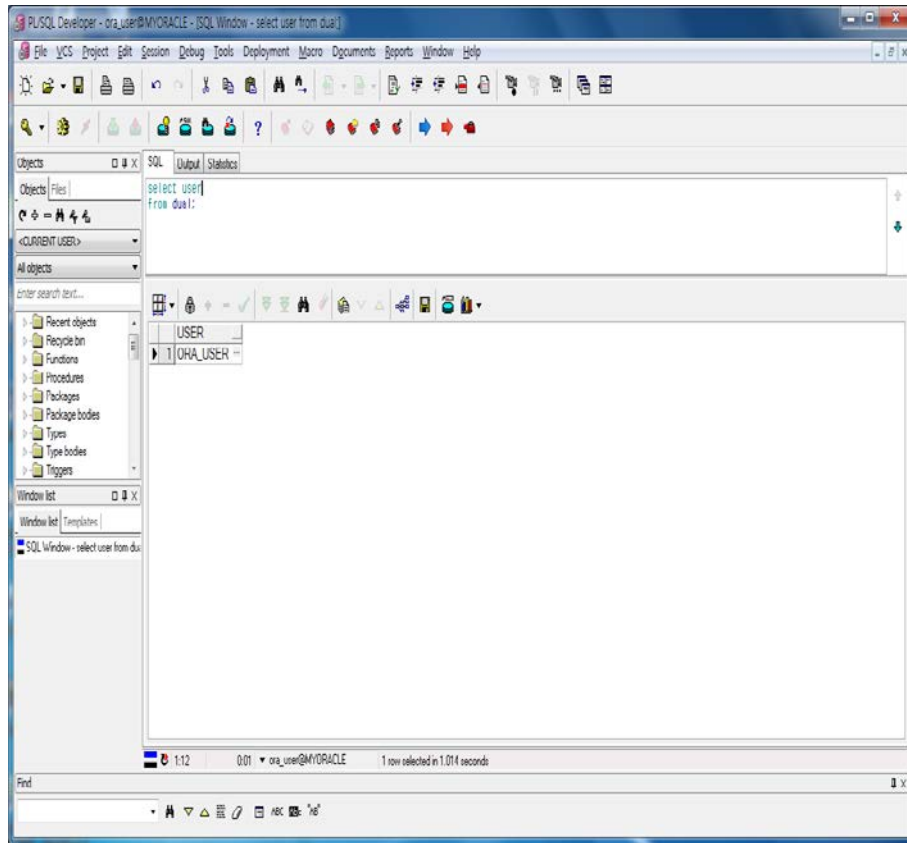
(4) Orange

- 국산 제품으로 화면은 토드와 비슷하고 점차 사용자가 늘고 있다
- 상용제품으로 프리웨어 버전은 없고 기간제한의 Trial 버전은 존재
- <http://www.warevalley.com/>



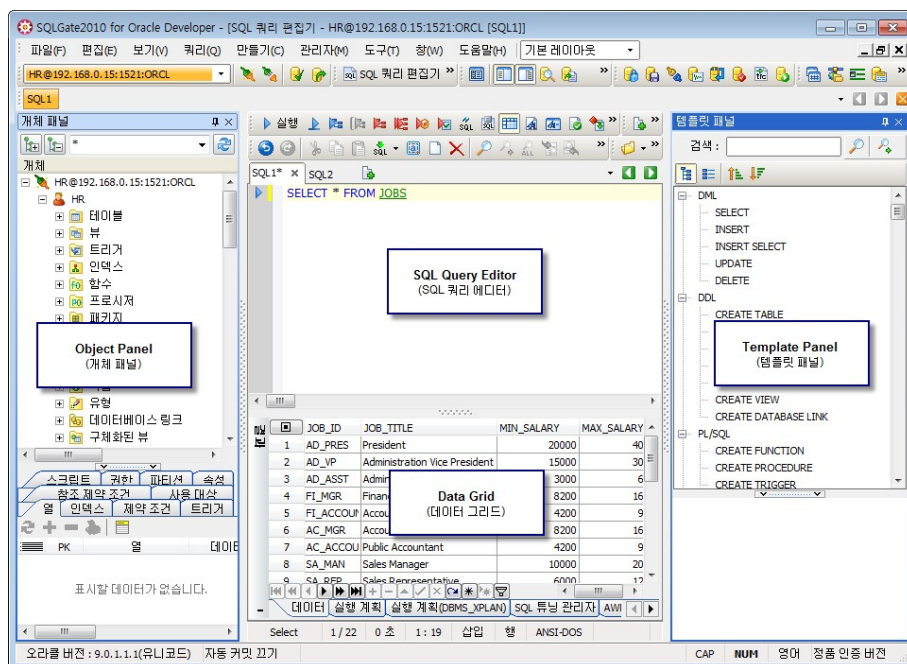
(5) PL/SQL Developer

- 토드나 오렌지와 GUI 면에서는 차이가 있지만, 나름 개발하기에 편리한 툴
- 상용제품으로 프리웨어 버전은 없고 기간제한의 Trial 버전은 존재
- <http://www.allroundautomations.com>



(6) SQLGate for Oracle Developer

- 테이블 설계 및 정의서 작성에 용이하다.
- 다운로드: <https://www.sqlgate.com/>



1.2.3 샘플 스키마

(1) SCOTT 계정 활성화

```
C:\TEMP> sqlplus system/sys
SQL> CREATE USER SCOTT IDENTIFIED BY TIGER;
SQL> GRANT CONNECT, RESOURCE TO SCOTT;
SQL> @SCOTT.SQL
SQL> CONN SCOTT/TIGER
SQL> SHOW USER
SQL> SELECT COUNT(*) FROM TAB;
SQL> QUIT;
```

(2) 오라클 접속

```
C:\TEMP> SQLPLUS SCOTT/TIGER
```

(3) 문제해결

■ 계정이 잠금되었을때

```
SQL> CONN SCOTT/TIGER
ERROR:
ORA-28000: 계정이 잠금되었습니다
경고: 이제는 ORACLE에 연결되어 있지 않습니다.

SQL> ALTER USER SCOTT ACCOUNT UNLOCK;
사용자가 변경되었습니다.

SQL> CONN SCOTT/TIGER
연결되었습니다.
SQL>
```

■ DBA 및 USER 비밀번호 까먹었을때

```
SQLPLUS /"AS SYSDBA"
접속 후
ALTER USER ID IDENTIFIED BY PW;
EX) ALTER USER SYS IDENTIFIED BY ORA123;

CF.
오라클 SYS, SYSTEM 암호 까먹었을때
명령 프롬프트에서 다음을 실행합니다.

C:>SQLPLUS /"AS SYSDBA"
SQL> SHOW USER
USER IS "SYS"

암호를 원하는 대로 설정합니다.

SQL> ALTER USER SYS IDENTIFIED BY 암호;
SQL> ALTER USER SYSTEM IDENTIFIED BY 암호;
```

1.2.4 과제 스키마

(1) 과제 스키마 설치

- ch01_script.sql, expall.dmp, expcust.dmp, export_sales.sql 파일을 다운로드 받아 'C:\Temp\oracle\01\' 폴더 생성 후 붙여 놓는다.
- 명령 창을 열어 'C:\Temp\oracle\01\' 폴더로 이동한다.

```
C:\temp\oracle\01> sqlplus system/sys
SQL> @ch01_script.sql
SQL> CONN ORA_USER/HONG
연결되었습니다.
SQL> SHOW USER
USER은 "ORA_USER"입니다
SQL> EXIT
C:\temp\oracle\01> IMP ORA_USER/HONG FILE=EXPALL.DMP LOG=EMPALL.LOG IGNORE=Y GRANTS=Y ROWS=Y INDEXES=Y FULL=Y

Import: Release 11.2.0.1.0 - Production on 일 6월 11 00:38:37 2017
...
임포트가 경고 없이 정상적으로 종료되었습니다.

C:\TEMP> IMP ORA_USER/HONG FILE=EXPCUST.DMP LOG=EMPALL.LOG IGNORE=Y GRANTS=Y ROWS=Y INDEXES=Y FULL=Y

Import: Release 11.2.0.1.0 - Production on 일 6월 11 00:38:54 2017
...
임포트가 경고 없이 정상적으로 종료되었습니다.

SQL> @export_sales.sql
```

(2) 과제 스키마 설치 확인

- SQL Plus나 SQL Developer에 접속 해 다음 명령어 실행

```
C:\temp\oracle\01> SQLPLUS ORA_USER/HONG

SQL*Plus: Release 11.2.0.1.0 Production on 일 6월 11 00:42:50 2017

Copyright (c) 1982, 2010, Oracle. All rights reserved.

다음에 접속됨:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SELECT TABLE_NAME FROM USER_TABLES;

TABLE_NAME
-----
SALES
PRODUCTS
KOR_LOAN_STATUS
JOBS
JOB_HISTORY
EMPLOYEES
DEPARTMENTS
CUSTOMERS
```

COUNTRIES
CHANNELS

10 개의 행이 선택되었습니다.

SQL>

(3) 과제 스키마 설명

- employees : 사원테이블 (사원번호, 사원명, 부서번호 등)
- departments : 부서테이블 (부서번호, 부서명 등)
- jobs : job 테이블 (job번호, 명칭 등)
- job_history : job_history 테이블 (job번호, 사원번호, 부서번호 등)
- countries : 국가 테이블 (국가번호, 국가코드, 국가명 등)
- customers : 고객 테이블 (고객번호, 고객명, 국가번호 등)
- channels : 판매채널 테이블(채널번호, 채널명 등)
- products : 제품 테이블 (제품번호, 제품명 등)
- sales : 판매 테이블(제품번호, 고객번호, 채널번호, 사원번호 등)

1.2.5 추가 스키마 (마당서점)

```
-- 이름: DEMO_MADANG.SQL
-- 설명
-- MADANG 스키마를 생성하고 MADANG 서점 실습테이블과 데이터를 입력한다.
-- 본스크립트는 SYSTEM 계정에서 실행해야한다.
---
-- SQLPLUS 실행방법
-- SQL>@DEMO_MADANG.SQL
-- SQL DEVELOPER F5 스크립터 실행

D:\temp\sql>SQLPLUS SYSTEM/MANAGER
SQL> ED DEMO_MADANG.SQL
DROP USER MADANG CASCADE;

CREATE USER MADANG IDENTIFIED BY MADANG DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP PROFILE DEFAULT;

GRANT CONNECT, RESOURCE TO MADANG;
GRANT CREATE VIEW, CREATE SYNONYM TO MADANG;

ALTER USER MADANG ACCOUNT UNLOCK;

CONN MADANG/MADANG;

CREATE TABLE BOOK (
  BOOKID      NUMBER(2) PRIMARY KEY,
  BOOKNAME    VARCHAR2(40),
  PUBLISHER   VARCHAR2(40),
  PRICE       NUMBER(8)
);

CREATE TABLE CUSTOMER (
  CUSTID      NUMBER(2) PRIMARY KEY,
  NAME        VARCHAR2(40),
  ADDRESS     VARCHAR2(50),
  PHONE       VARCHAR2(20)
);
```

```

CREATE TABLE ORDERS (
  ORDERID NUMBER(2) PRIMARY KEY,
  CUSTID  NUMBER(2) REFERENCES CUSTOMER(CUSTID),
  BOOKID  NUMBER(2) REFERENCES BOOK(BOOKID),
  SALEPRICE NUMBER(8) ,
  ORDERDATE DATE
);
-- BOOK, CUSTOMER, ORDERS 데이터 생성
INSERT INTO BOOK VALUES(1, '축구의 역사', '굿스포츠', 7000);
INSERT INTO BOOK VALUES(2, '축구하는 여자', '나무수', 13000);
INSERT INTO BOOK VALUES(3, '축구의 이해', '대한미디어', 22000);
INSERT INTO BOOK VALUES(4, '골프 바이블', '대한미디어', 35000);
INSERT INTO BOOK VALUES(5, '피겨 교본', '굿스포츠', 8000);
INSERT INTO BOOK VALUES(6, '역도 단계별기술', '굿스포츠', 6000);
INSERT INTO BOOK VALUES(7, '야구의 추억', '이상미디어', 20000);
INSERT INTO BOOK VALUES(8, '야구를 부탁해', '이상미디어', 13000);
INSERT INTO BOOK VALUES(9, '올림픽 이야기', '삼성당', 7500);
INSERT INTO BOOK VALUES(10, 'OLYMPIC CHAMPIONS', 'PEARSON', 13000);

INSERT INTO CUSTOMER VALUES (1, '박지성', '영국 맨체스터', '000-5000-0001');
INSERT INTO CUSTOMER VALUES (2, '김연아', '대한민국 서울', '000-6000-0001');
INSERT INTO CUSTOMER VALUES (3, '장미란', '대한민국 강원도', '000-7000-0001');
INSERT INTO CUSTOMER VALUES (4, '추신수', '미국 클리블랜드', '000-8000-0001');
INSERT INTO CUSTOMER VALUES (5, '박세리', '대한민국 대전', NULL);

-- 주문(ORDERS) 테이블의 책값은 할인 판매를 가정함
INSERT INTO ORDERS VALUES (1, 1, 1, 6000, TO_DATE('2014-07-01','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (2, 1, 3, 21000, TO_DATE('2014-07-03','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (3, 2, 5, 8000, TO_DATE('2014-07-03','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (4, 3, 6, 6000, TO_DATE('2014-07-04','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (5, 4, 7, 20000, TO_DATE('2014-07-05','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (6, 1, 2, 12000, TO_DATE('2014-07-07','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (7, 4, 8, 13000, TO_DATE('2014-07-07','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (8, 3, 10, 12000, TO_DATE('2014-07-08','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (9, 2, 10, 7000, TO_DATE('2014-07-09','YYYY-MM-DD'));
INSERT INTO ORDERS VALUES (10, 3, 8, 13000, TO_DATE('2014-07-10','YYYY-MM-DD'));

-- 여기는 3장에서 사용되는 IMPORTED_BOOK 테이블

CREATE TABLE IMPORTED_BOOK (
  BOOKID      NUMBER ,
  BOOKNAME    VARCHAR(40),
  PUBLISHER   VARCHAR(40),
  PRICE       NUMBER(8)
);
INSERT INTO IMPORTED_BOOK VALUES(21, 'ZEN GOLF', 'PEARSON', 12000);
INSERT INTO IMPORTED_BOOK VALUES(22, 'SOCCER SKILLS', 'HUMAN KINETICS', 15000);

COMMIT;

SQL> @DEMO_MADANG.SQL

사용자가 삭제되었습니다.
...
SQL>

```

1.3 SQL과 SQL*Plus의 개념

1.3.1 사용 용도에 따른 SQL

- SQL(Structured Query Language) : 데이터베이스에 저장된 데이터를 조회, 입력, 수정 삭제하는 등의 조작이나, 테이블을 비롯한 다양한 객체(시퀀스, 인덱스 등)를 생성 및 제어하는 역할을 한다.
- SQL의 종류
 - 데이터 정의어(DDL) : 데이터베이스 관리자나 응용 프로그래머가 데이터베이스의 논리적 구조를 정의하기 위한 언어로서 데이터 사전(Data Dictionary)에 저장 된다.
 - 데이터 조작어(DML) : 데이터베이스에 저장된 데이터를 조작하기 위해 사용하는 언어로서 데이터 검색(Retrieval), 추가(Insert), 삭제>Delete), 갱신(Update) 작업 수행한다.
 - 데이터 제어어(DCL) : 데이터에 대한 접근 권한 부여 등의 데이터베이스 시스템의 트랜잭션을 관리하기 위한 목적으로 사용되는 언어이다.

유형	명령문
DQL:Data Query Language(질의어)	SELECT(데이터 검색시 사용)
DML:Data Manipulation Language (데이터 조작어) - 데이터 변경시 사용	INSERT(데이터 입력) UPDATE(데이터 수정) DELETE(데이터 삭제)
DDL:Data Definition Language (데이터 정의어) - 객체 생성 및 변경시 사용	CREATE(데이터베이스 생성) ALTER(데이터베이스 변경) DROP(데이터베이스 삭제) RENAME(데이터베이스 객체이름 변경) TRUNCATE(데이터베이스 저장 공간 삭제)
TCL:Transaction Control Language (트랜잭션 처리어)	COMMIT(트랜잭션의 정상적인 종료처리) ROLLBACK(트랜잭션 취소) SAVEPOINT(트랜잭션내에 임시 저장점 설정)
DCL:Data Control Language (데이터 제어어)	GRANT(데이터베이스에 대한 일련의 권한 부여) REVOKE(데이터베이스에 대한 일련의 권한 취소)

1.3.2 가장 기본이 되는 SELECT문

- SELECT문은 테이블에 저장된 데이터를 조회하는 데 사용되는 가장 기본적인 문법이고 가장 많이 쓰이는 문장이다.
 - SELECT : 데이터 검색시 사용

```
SELECT * FROM DEPT;
```

1.3.3 DML(Data Manipulation Language)

- DML은 데이터를 조작하는 역할을 한다. 새로운 데이터를 삽입하고, 기존의 데이터를 변경하고 삭제하는 것이 DML에 속한다.
 - INSERT : 데이터 입력
 - UPDATE : 데이터 수정
 - DELETE : 데이터 삭제

```
INSERT INTO DEPT VALUES(50, '총무부', '서울');
UPDATE DEPT SET LOC='부산' WHERE DNAME='총무부';
DELETE FROM DEPT WHERE DEPTNO=50;
```

1.3.4 TCL(Transaction Control Language)

- TCL은 데이터의 일관성을 유지하면서 안정적으로 데이터를 복구시키기 위해서 사용한다.
 - COMMIT : 변경된 내용을 영구 저장한다.
 - ROLLBACK : 변경되기 이전 상태로 되돌린다.
 - SAVEPOINT : 특정 위치까지를 저장 혹은 이전 상태로 되돌리 수 있도록 트랜잭션 중에 저장점을 만든다.

1.3.5 DDL(Data Definition Language)

- DDL은 데이터베이스 객체들을 생성 또는 변경, 제거할 때 사용한다. 객체란 테이블, 인덱스, 뷰, 트리거 등 SQL문을 수행하는 대상이 된다.
 - CREATE : 데이터베이스 생성
 - ALTER : 데이터베이스 변경
 - DROP : 데이터베이스 삭제
 - RENAME : 데이터베이스 객체이름 변경
 - TRUNCATE : 데이터베이스 저장 공간 삭제

```
-- CREATE
CREATE TABLE DEPT01 (
    DEPTNO NUMBER(4),
    DNAME VARCHAR2(10),
    LOC VARCHAR2(9)
);

-- ALTER
ALTER TABLE DEPT01
MODIFY(DNAME VARCHAR2(30));

-- RENAME
RENAME DEPT01 TO DEPT02;

-- TRUNCATE
TRUNCATE TABLE DEPT02;

-- DROP
DROP TABLE DEPT02;
DROP TABLE DEPT02 CASCADE CONSTRAINTS; -- CASCADE CONSTRAINTS 옵션은 종속된 제약조건을 삭제한다.
```

1.3.6 DCL(Data Control Language)

- DCL은 특정 사용자에게 권한을 부여하거나 제거하기 위해서 사용하는 명령어이다.
 - GRANT : 데이터베이스에 대한 일련의 권한 부여
 - REVOKE : 데이터베이스에 대한 일련의 권한 취소

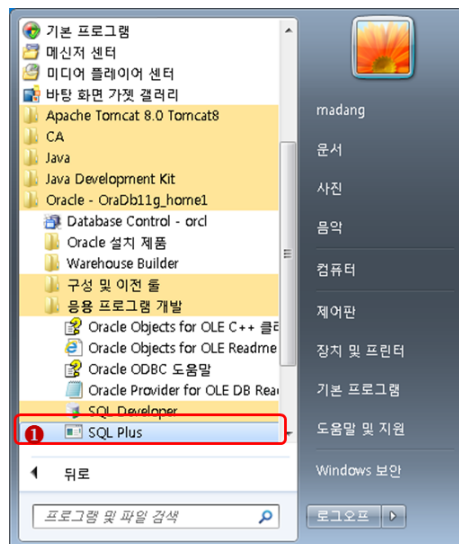
```
GRANT CREATE SESSION TO ORAUZER01;
REVOKE CREATE SESSION FROM ORAUZER01;
```


1.3.7 SQL*Plus란?

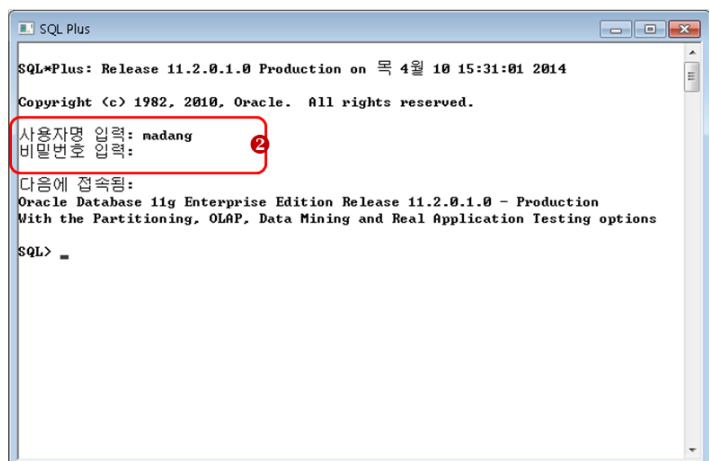
■ SQL*Plus란?

- SQL 명령문을 Command line으로 수행할 수 있는 도구이다.
- 칼럼이나 데이터의 출력 형식을 설정하거나 환경 설정하는 기능을 제공한다.

① SQL Plus 시작



② 쿼리창 열기



1.4 SQL*Plus 로그인

1.4.1 Command 환경에서 SQL*Plus 로그인

```
-- 형식
SQLPLUS 사용자/계정암호

-- 예
SQLPLUS SCOTT/TIGER
EXIT
```

1.4.2 시스템 권한을 갖는 데이터베이스 관리자

- 데이터베이스 사용자는 오라클 계정(Account)이라는 용어와 같은 의미로 사용된다.

사용자 계정	설명
SYS	오라클 SUPER 사용자 계정이며, 데이터베이스에서 발생하는 모든 문제들을 처리할 수 있는 권한을 가지고 있다.
SYSTEM	오라클 데이터베이스를 유지보수 및 관리할 때 사용하는 사용자 계정이다. SYS 사용자와 다른 점은 데이터베이스를 생성할 수 있는 권한이 없다.
SCOTT	처음 오라클을 사용하는 사용자의 실습을 위해 만들어 놓은 연습용 계정이다.
HR	오라클에 접근할 수 있도록 샘플로 만들어 놓은 사용자 계정이다.

1.4.3 SQL*Plus 로그인에 실패할 경우 해결 방법

(1) 사용자의 계정이 잠겨 있는 경우

```
SQLPLUS SYSTEM/MANAGER
ALTER USER SCOTT IDENTIFIED BY TIGER ACCOUNT UNLOCK;

-- CONNECT SCOTT/TIGER;
CONN SCOTT/TIGER;
```

(2) 데이터베이스가 기동되지 않는(not available) 경우

```
SQLPLUS SYS/MANAGER AS SYSDBA
STARTUP
```

2 SQL의 기본

2.1 데이터 덱서너리 TAB

- 오라클을 설치하면 제공되는 사용자인 SCOTT은 학습을 위해서 테이블들이 제공된다.
- TAB은 TABLE의 약자로서 SCOTT 사용자가 소유하고 있는 테이블의 정보를 알려주는 데이터 덱서너리이다.

```
SQLPLUS SCOTT/TIGER
SELECT * FROM TAB;
```

2.2 테이블 구조를 살펴보기 위한 DESC

- 테이블에서 데이터를 조회하기 위해서는 테이블의 구조를 알아야 한다. 테이블의 구조를 확인하기 위한 명령어로는 DESCRIBE가 있다.
- DESC 명령어는 테이블의 컬럼 이름, 데이터 형, 길이와 NULL 허용 유무 등과 같은 특정 테이블의 정보를 알려준다.
- NOT NULL 제약 조건
 - NULL: 컬럼에 어떠한 값도 정해지지 않을 때 갖는 값이다. NULL은 할당받지 않은 값, 모르는 값, 정해지지 않을 값을 의미하여 0(숫자의 한 자리)이나, 스페이스(문자의 한 자리)와는 다른 값이다.
 - NOT NULL: 이 조건이 설정된 컬럼은 반드시 NULL값이 아닌 확실한 정보가 필수적으로 입력되어야 한다.
- 기본 키(Primary key) 제약 조건
 - 유일한 값을 저장하기 위한 조건으로 DESC 명령으로 확인할 수 없다.
- 외래 키(Foreign key) 제약 조건
 - 관계형 데이터베이스는 테이블 사이의 관계를 설정할 수 있다.
 - 다른 테이블의 컬럼을 참조하도록 연결하여 제약을 걸어 두기 위해서 사용한다.

```
-- DESC[RIBE] 테이블명
DESC DEPT
DESC EMP
```

2.3 오라클의 데이터형

2.3.1 NUMBER

- NUMBER 데이터 형은 숫자 데이터를 저장하기 위해서 제공된다.
- precision은 소수점을 포함한 전체 자리수를 의미하며 scale은 소수점 이하 자리수를 지정한다.
- scale을 생략한 채 precision만 지정하면 소수점 이하는 반올림되어 정수 값만 저장된다.
- precision과 scale을 모두 생략하면 입력한 데이터 값만큼 공간이 할당된다.

```
-- 형식
NUMBER(PRECISION, SCALE)
```

```
-- 예
NUMBER(4) -- 정수로 최대 4자리
NUMBER(8,2) -- 전체 8자리, 소수 이하 2자리의 실수
```

2.3.2 DATE

- DATE는 세기, 년, 월, 일, 시간, 분, 초의 날짜 및 시간 데이터를 저장하기 위한 데이터 형이다.
- 기본 날짜 형식은 "YY/MM/DD" 형식으로 "년/월/일"로 출력된다. 2005년 12월 14일은 "05/12/14"로 출력된다.

2.3.3 CHAR

- 문자 데이터를 저장하기 위한 자료형으로 CHAR가 있다. CHAR는 고정 길이 문자 데이터를 저장한다.
- 입력된 자료의 길이와는 상관없이 정해진 길이만큼 저장 영역 차지하며 최소 크기는 1이다.
- char(20)이라고 설정하고 'seoul'이라는 데이터를 저장하였다면 ...

address																			
s	e	o	u	l															

- CHAR는 주어진 크기만큼 저장공간이 할당되므로 편차가 심한 데이터를 입력할 경우 위의 예와 같이 저장공간의 낭비를 초래한다.

2.3.4 VARCHAR2

- VARCHAR2 데이터 형은 가변적인 길이의 문자열을 저장하기 위해서 제공한다.
- address 란 컬럼의 데이터형을 VARCHAR2(20)이라고 설정하고, 'seoul' 이란 데이터를 저장하였다면

address				
s	e	o	u	l

- VARCHAR2는 저장되는 데이터에 의해서 저장공간이 할당되므로 메모리 낭비를 줄일 수 있습니다.

[꿀팁] Boolean 타입

- 오라클 데이터베이스에서는 불리형(boolean type)을 지원하지 않으므로 아래와 같은 방법을 사용할 수 있다.

```
-- 컬럼에 체크 제약조건(check constraint)을 사용한다.
CREATE TABLE ITEMS (
  ITEM_NO NUMBER(3),
  ISACTIVE CHAR(1) CHECK (ISACTIVE IN ('Y','N'))
);

INSERT INTO ITEMS VALUES (101, 'Y');
```

2.4 데이터를 조회하기 위한 SELECT문

- SELECT 문은 데이터를 조회하기 위한 SQL 명령어이다.

```
-- 형식
SELECT [DISTINCT] {*, COLUMN[ALIAS], . . .}
FROM TABLE_NAME;

-- 예
SELECT * FROM DEPT;
SELECT * FROM EMP;
DESC EMP
-- EMP 테이블의 구조 살펴보기
SET LINESIZE 100
-- 라인당 출력될 문자 수 변경
SELECT EMPNO, ENAME FROM EMP;
```

2.5 산술 연산자

- SQL은 다른 프로그래밍 언어와 같이 산술 연산자를 사용할 수 있다.

종류	예
+	SELECT SAL + COMM FROM EMP;
-	SELECT SAL - 100 FROM EMP;
*	SELECT SAL * 12 FROM EMP;
/	SELECT SAL / 2 FROM EMP;

```
SELECT ENAME, SAL, SAL*12
FROM EMP;
```

2.6 NULL도 데이터!

- 오라클에서의 널은 매우 중요한 데이터이다. 왜냐하면 오라클에서는 컬럼에 널값이 저장되는 것을 허용하는데 널 값을 제대로 이해하지 못한 채 쿼리문을 사용하면 원하지 않는 결과를 얻을 수 있기 때문이다.
- 널의 정의
 - 0(zero)도 아니고 빈 공간도 아니다.
 - 미확정(해당 사항 없음), 알 수 없는(unknown) 값을 의미한다.
 - 어떤 값인지 알 수 없지만 어떤 값이 존재하고 있다.
 - ? 혹은 ∞의 의미이므로 연산, 할당, 비교가 불가능하다.

$$\begin{aligned} 100 + ? &= ? \\ 100 + \infty &= \infty \end{aligned}$$

```
SELECT ENAME, SAL, JOB, COMM, SAL*12, SAL*12+COMM
```

```
FROM EMP;
```

```
-- NVL(Null Value) 함수는 NULL을 0 또는 다른 값으로 변환하기 위해서 사용한다.  
SELECT ENAME, SAL, JOB, COMM, SAL*12, SAL*12+NVL(COMM, 0)  
FROM EMP;
```

2.7 컬럼 이름에 별칭 지정하기

- SQL에서 쿼리문의 결과가 출력될 때, 컬럼 이름이 컬럼에 대한 헤딩(heading)으로 출력된다.

2.7.1. AS로 컬럼에 별칭 부여하기

- 컬럼 이름 대신 별칭을 출력하고자 하면 컬럼을 기술한 바로 뒤에 AS 라는 키워드를 쓴 후 별칭을 기술한다.
- AS 키워드는 생략이 가능하다.

```
SELECT ENAME, SAL*12+NVL(COMM, 0) AS ANNSAL  
FROM EMP;
```

```
-- AS 키워드는 생략이 가능하다.  
SELECT ENAME, SAL*12+NVL(COMM, 0) ANNSAL  
FROM EMP;
```

2.7.2. ""로 별칭 부여하기

- 위 예를 살펴보면 별칭을 부여 할 때에는 대소문자를 섞어서 기술하였는데 출력 결과를 보면 일괄적으로 대문자로 출력된 것을 확인할 수 있다.
- 대소문자를 구별하고 싶으면 " "을 사용한다.
- " "을 사용하여 별칭을 부여할 경우에는 별칭에 공백문자나 \$,_, #등 특수 문자를 포함시킬 수 있다.

```
SELECT ENAME, SAL*12+NVL(COMM, 0) "A N N S A L"  
FROM EMP;
```

2.7.3. 별칭으로 한글 사용하기

- 오라클에서 한글을 지원하므로 별칭이 아닌 테이블을 생성할 때 컬럼을 설정하면서 컬럼 이름도 한글로 부여할 수 있다.

```
SELECT ENAME, SAL*12+NVL(COMM, 0) "연봉"  
FROM EMP;
```

2.8 Concatenation 연산자의 정의와 사용

- Concatenation 의 사전적인 의미는 연결이다.
- 여러 개의 컬럼을 연결할 때 사용하는데 Concatenation 연산자로 "||" 수직바를 사용한다.

```
SELECT ENAME || ' IS A ' || JOB
FROM EMP;
```

2.9 DISTINCT 키워드

- 동일한 데이터 값들이 중복되지 않도록, 즉 한 번씩만 출력되도록 하기 위해서 사용한다.

```
SELECT DEPTNO
FROM EMP;

SELECT DISTINCT DEPTNO
FROM EMP;
```

[과제] 과제-02-01.TXT

1. emp 테이블을 구성하는 각 컬럼의 데이터 형태에 대해서 설명하세요.

```
SQL> DESC EMP
이름                                널?       유형
-----
EMPNO                                NOT NULL  NUMBER(4)
ENAME                                VCHAR2(10)
JOB                                  VCHAR2(9)
MGR                                  NUMBER(4)
HIREDATE                             DATE
SAL                                  NUMBER(7,2)
COMM                                  NUMBER(7,2)
DEPTNO                               NUMBER(2)

SQL>
```

컬럼 이름	자료형 설명
EMPNO	정수로 최대 4자리
ENAME	가변형 문자열로 최대 10자리
JOB	
MGR	
HIREDATE	
SAL	
COMM	
DEPTNO	

3 SQL*Plus 명령어

3.1 SQL*Plus 명령어의 개념

- SQL*Plus는 SQL문을 실행시키고 그 결과를 볼 수 있도록 오라클에서 제공하는 툴이다.
- SQL은 데이터베이스에서 자료를 검색하고 수정하고 삭제하는 등을 위한 데이터베이스 언어인 반면, SQL*Plus 명령어는 툴에서 출력 형식을 지정하는 등 환경을 설정한다.
- SQL과 SQL*Plus 명령어의 차이점

SQL 문	SQL*Plus 명령문
관계형 데이터베이스의 ANSI 표준 언어	SQL 문을 실행 시킬 수 있는 오라클의 툴
여러 줄 실행	한줄 실행
종결문자(;) 필요	종결문자 불요
연결문자 불요	연결문자(-) 필요
키워드 단축 불가	키워드 단축 가능
버퍼에 마지막 명령문 저장	버퍼 저장 안함

- SQL*Plus 명령어

명령어	기능
LIST, RUN, @, /	편집 명령
SAVE, GET, EDIT, SPOOL	파일 명령
HOST, EXIT	데이터베이스 접속 및 종료
LINE, PAGE	출력 형식

3.2 SQL*Plus 편집 명령

명령어(약어)	설명
LIST (L)	버퍼에 저장된 모든 SQL 문 또는 검색한 라인의 SQL 문을 나타낸다.
/	SQL 문을 보여주지 않고 바로 실행한다.
RUN (R)	버퍼에 저장된 SQL 문을 보여주고 실행한다.

```
SQL> SELECT DISTINCT DEPTNO
FROM EMP;
SQL> L
SQL> /
SQL> R
```

3.3 SQL*Plus 파일 명령어

명령어(약어)	설 명
EDIT (ED)	파일의 내용을 vi(유닉스)나 notepad(윈도우즈)와 같은 에디터로 읽어 편집할 수 있도록 한다.
HOST	오라클을 종료하지 않고 OS 명령을 수행할 수 있도록 OS 환경으로 잠시 빠져 나갈 수 있도록 한다. OS Prompt 상에서 Exit 하면 다시 오라클 환경으로 돌아온다.
SAVE	SQL 버퍼 내의 현재 내용을 실제 파일로 저장한다.
@	SQL 파일에 저장된 내용을 실행한다.
SPOOL	오라클 화면을 갈무리하여 파일로 저장한다.
GET	파일의 내용을 SQL 버퍼로 읽어 들인다.
EXIT	오라클을 종료한다.

3.3.1 파일에 내용을 메모장에서 편집하게 하는 EDIT(ED)

- SQL은 파일의 내용을 메모장에서 쉽게 편집할 수 있도록 ED[IT] 명령어를 제공한다.
- 주의 할 점은 SQL 버퍼를 편집기로 열었을 때 명령어문 끝에 붙였던 종결문자 ; 가 편집화면에서는 /로 대체 된다는 점이다.

```
SQL> SELECT DISTINCT DEPTNO  
FROM EMP;  
SQL> ED
```

3.3.2 DOS 프롬프트로 나가게 하는 HOST

- SQL 명령문이 저장된 버퍼는 파일 형태인데 그 파일 이름은 "afiedt.buf"입니다.
- 오라클을 종료하지 않고 DOS 명령어인 dir를 사용하여 afiedt.buf 파일이 존재하는 것을 확인하려면 DOS 환경으로 나가는 HOST 명령어를 사용한다.

```
SQL> HOST  
Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
D:\Temp>EXIT  
  
SQL>
```

3.3.3 사용자가 현재 수행 중인 쿼리문을 저장하는 SAVE

- SQL*Plus 에서는 사용자가 가장 최근에 수행한 쿼리문을 파일로 저장할 수 있도록 하는 SAVE 명령어를 제공한다.

```
SQL> L  
1 SELECT DISTINCT DEPTNO  
2* FROM EMP  
SQL> SAVE TEST.SQL  
FILE TEST.SQL(이)가 생성되었습니다  
SQL> HOST  
Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
D:\Temp>DIR
```

3.3.4 SQL 파일에 저장된 명령어를 실행하는 @

- @ 명령어는 확장자가 .SQL인 파일에 저장되어 있는 쿼리문을 실행시키기 위해서 사용한다.

```
SQL> @TEST.SQL
```

```
DEPTNO
-----
      30
      20
      10
```

```
SQL>
```

3.3.5 갈무리 기능을 하는 SP00L

- SAVE 명령어가 SQL문 자체를 저장하는데 비해 SP00L명령어는 SQL문과 실행된 쿼리 결과를 파일로 기록하는 명령어이다.
- 즉 화면에 보여지는 내용 전체를 갈무리 해서 하나의 파일로 만든다.

3.3.6 저장한 SQL 명령어를 가져오는 GET

- SAVE 명령어를 사용하여 저장한 SQL 명령어를 다시 사용할 수 있는데 이때 사용하는 명령어가 GET 이다. GET 명령어도 SAVE 명령어와 마찬가지로 파일 이름만 기술하고 확장자를 기술하지 않으면 기본적으로 확장자를 .SQL로 인식한다.

```
SQL> GET TEST.SQL
1  SELECT DISTINCT DEPTNO
2* FROM EMP
SQL>
```

3.4 시스템 변수 조작을 위한 SET 명령어

3.4.1 컬럼 제목의 출력 여부를 결정하는 HEADING(HEA) 변수

- HEADING은 SELECT 명령어를 수행한 후 실행결과가 출력될 때 컬럼의 제목을 출력할 것인지의 여부를 제어한다.

```
SQL> SET HEADING OFF
SQL> SET HEADING ON
```

3.4.2 라인 당 출력할 문자의 수를 결정하는 LINESIZE 변수

- 시스템 변수 LINESIZE 는 라인 당 출력될 문자 수를 결정한다.
- 디폴트 값은 80 이므로 SELECT 문의 출력 결과를 출력할 때 한 라인에 80까지만 출력한다.

```
SQL> SET LINESIZE 100
```

3.4.3 페이지 당 출력할 라인의 수를 결정하는 PAGESIZE 변수

- PAGESIZE 변수는 SQL 명령문의 실행 결과에 대해서 출력될 수 있는 페이지의 크기를 설정하는 변수로서 한 페이지에 컬럼 제목, 컬럼 제목과 데이터 구분선, 페이지를 구분하기 위한 공백 라인을 포함한다.

```
SQL> SET PAGESIZE 10
```

3.4.4 컬럼에 저장된 데이터의 출력 형식 변경을 위한 COLUMN FORMAT

```
SQL> COLUMN ENAME FORMAT A25  
SQL> COLUMN SAL FORMAT 9,999,999  
SQL> COLUMN COMM FORMAT 0,000,000
```

4 SELECT로 특정 데이터를 추출하기

4.1 WHERE 조건과 비교 연산자

- 원하는 로우만 얻으려면 제한하는 조건을 SELECT 문에 WHERE 절을 추가하여 제시해야 한다.

```
SELECT *  
FROM EMP  
WHERE SAL >= 3000;
```

4.1.1 비교 연산자

연산자	의 미	예 제
=	같다.	SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL=3000;
>	보다 크다.	SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL>3000;
<	보다 작다.	SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL<3000;
>=	보다 크거나 같다.	SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL>=3000;
<=	보다 작거나 같다.	SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL<=3000;
<>, !=, ^=	다르다.	SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL<>3000;

4.1.2 문자 데이터 조회

- SQL에서 문자열이나 날짜는 반드시 작은따옴표(')안에 표시해야 한다.

```
SELECT EMPNO, ENAME, SAL  
FROM EMP  
WHERE ENAME='FORD';  
  
SELECT EMPNO, ENAME, SAL  
FROM EMP  
WHERE ENAME='ford'; -- 테이블 내에 저장된 데이터 값은 대소문자를 구분한다.
```

4.1.3 날짜 데이터 조회

- 날짜는 문자열과 마찬가지로 단일 따옴표 안에 기술해야 한다.

```
SELECT *  
FROM EMP  
WHERE HIREDATE<='1982/01/01';
```

4.2 논리 연산자

연산자	의미
AND	두 가지 조건을 모두 만족해야만 검색할 수 있다. SELECT * FROM emp WHERE deptno=10 AND job='MANAGER';
OR	두 가지 조건 중에서 한 가지만 만족하더라도 검색할 수 있다. SELECT * FROM emp WHERE deptno=10 OR job='MANAGER';
NOT	조건에 만족하지 못하는 것만 검색한다. SELECT * FROM emp WHERE NOT deptno=10;

4.2.1 AND 연산자

- 두 가지 조건을 모두 만족할 경우에만 검색할 수 있도록 하기 위해서는 AND 연산자를 사용한다.

```
SELECT *  
FROM EMP  
WHERE DEPTNO=10 AND JOB='MANAGER';
```

4.2.2 OR 연산자

- 두 가지 조건 중에서 한 가지만 만족하더라도 검색할 수 있도록 하기 위해서는 OR 연산자를 사용한다.

```
SELECT *  
FROM EMP  
WHERE DEPTNO=10 OR JOB='MANAGER';
```

4.2.3 NOT 연산자

- NOT 연산자는 참은 거짓으로 거짓은 참으로 즉 반대되는 논리값을 구하는 연산자이다.

```
SELECT *  
FROM EMP  
WHERE NOT DEPTNO=10;  
  
SELECT *  
FROM EMP  
WHERE DEPTNO<>10;
```

4.3 BETWEEN AND 연산자

- 오라클에서는 특정 범위의 값을 조회하기 위해서는 BETWEEN AND 연산자를 사용할 수 있다.

```
SELECT *
```

```

FROM EMP
WHERE SAL >= 2000 AND SAL <= 3000;

SELECT *
FROM EMP
WHERE SAL BETWEEN 2000 AND 3000;

SELECT *
FROM EMP
WHERE SAL NOT BETWEEN 2000 AND 3000;

SELECT *
FROM EMP
WHERE HIREDATE BETWEEN '1987/01/01' AND '1987/12/31';

```

4.4 IN 연산자

- 동일한 필드가 여러 개의 값 중에 하나인 경우인지를 살펴보기 위해서 비교 연산자와 논리 연산자 OR를 사용하여 복잡하게 쿼리문을 작성하지 않고 IN 연산자를 사용하여 훨씬 간단하게 표현할 수 있다.

```

SELECT *
FROM EMP
WHERE COMM=300 OR COMM=500 OR COMM=1400;

SELECT *
FROM EMP
WHERE COMM IN(300, 500, 1400);

SELECT *
FROM EMP
WHERE COMM <> 300 AND COMM <> 500 AND COMM <> 1400;

SELECT *
FROM EMP
WHERE COMM NOT IN(300, 500, 1400);

```

4.5 LIKE 연산자와 와일드카드

- LIKE 연산자는 검색하고자 하는 값을 정확히 모를 경우에도 검색 가능하도록 하기 위해서 와일드카드와 함께 사용하여 원하는 내용을 검색하도록 한다.
- LIKE 다음에는 pattern을 기술해야 하는데 pattern에 다음과 같이 두 가지 와일드카드가 사용된다.

와일드카드	의미
%	문자가 없거나, 하나 이상의 문자가 어떤 값이 와도 상관없다.
_	하나의 문자가 어떤 값이 와도 상관없다.

4.5.1 와일드카드(%) 사용하기

- 검색하고자 하는 값을 정확히 모를 경우 즉, 특정 문자 포함되지만 하고 그 이전이나 이후에 어떤 문자가 몇 개가 오든지 상관없다는 의미를 표현하기 위해서는 LIKE 연산자와 함께 %를 사용해야 한다.

```

SELECT *
FROM EMP
WHERE ENAME LIKE 'F%';

SELECT *
FROM EMP
WHERE ENAME LIKE '%A%';

SELECT *
FROM EMP
WHERE ENAME LIKE '%N';

```

4.5.2 와일드카드(_) 사용하기

- _ 역시 %와 마찬가지로 어떤 문자가 오든 상관없다는 의미로 사용되는 와일드카드이다.
- 차이점은 %는 몇 개의 문자가 오든 상관없지만 _ 는 단 한 문자에 대해서만 와일드카드 역할을 한다.

```

SELECT *
FROM EMP
WHERE ENAME LIKE '_A%';

SELECT *
FROM EMP
WHERE ENAME LIKE '_ _A%';

```

4.5.3 NOT LIKE 연산자

```

SELECT *
FROM EMP
WHERE ENAME NOT LIKE '%A%';

```

4.6 NULL을 위한 연산자

- 어떤 컬럼을 NULL 즉, 모르는 값과 같다(=)라는 것은 의미상으로 말이 되지 않기 때문에 = 대신 IS NULL 연산자를 사용해야 한다.

```

SELECT *
FROM EMP
WHERE COMM=NULL; -- NULL 값을 = 연산자로 판단할 수 없다.

SELECT *
FROM EMP
WHERE COMM IS NULL; -- EMP 테이블에서 COMM 컬럼이 NULL 사원을 출력한다.

SELECT *
FROM EMP
WHERE COMM IS NOT NULL;

```

4.7 정렬을 위한 ORDER BY절

- 정렬이란 크기 순서대로 나열하는 것을 의미한다.
- 오름차순(ascending) 정렬 방식
 - 작은 것이 위에 출력되고 아래로 갈수록 큰 값이 출력
- 내림차순(descending) 정렬 방식
 - 큰 값이 위에 출력되고 아래로 갈수록 작은 값이 출력
- 로우를 정렬하기 위해서는 SELECT 문에 ORDER BY 절을 추가하고 어떤 컬럼을 기준으로 어떤 정렬을 할 것인지를 결정해야 한다.

	ASC(오름차순)	DESC(내림차순)
숫자	작은 값부터 정렬	큰 값부터 정렬
문자	사전 순서로 정렬	사전 반대 순서로 정렬
날짜	빠른 날짜 순서로 정렬	늦은 날짜 순서로 정렬
NULL	가장 마지막에 나온다.	가장 먼저 나온다.

4.7.1 오름차순 정렬을 위한 ASC

- 오름차순 정렬은 작은 값부터 큰 값으로 정렬하는 것을 의미한다.(예:1~9, 'A'~'Z') 이를 위해서는 ASC를 컬럼 다음에 기술해야 하는데 만일 생략하게 되면 디폴트로 ASC로 지정되어 있기 때문에 오름차순으로 출력된다.

```
SELECT *  
FROM EMP  
ORDER BY SAL ASC;
```

```
SELECT *  
FROM EMP  
ORDER BY SAL;
```

```
SELECT *  
FROM EMP  
ORDER BY ENAME; -- 문자 데이터의 경우 아스키 코드 값으로 저장되므로 아스키 코드 값을 기준으로 정렬된다.
```

4.7.2 내림차순 정렬을 위한 DESC

- 내림차순 정렬은 큰 값부터 작은 값으로 정렬을 하는 것이다.(예:9~1, Z~A)

```
SELECT *  
FROM EMP  
ORDER BY SAL DESC;
```

```
SELECT *  
FROM EMP  
ORDER BY HIREDATE DESC;
```

```
SELECT *  
FROM EMP  
ORDER BY SAL DESC, ENAME ASC;
```


[과제] 과제-04-01.TXT

SQL> CONN SCOTT/TIGER 로 접속하여 SQL문을 작성하세요.

1. 사원 테이블(EMP)에서 가장 최근에 입사한 사원부터 출력하되, 동일한 입사일인 경우에는 사원번호(EMPNO)를 기준으로 오름차순으로 정렬해서 출력하는 SQL문을 작성하세요?

<정답>

2. 와일드 카드를 사용하여 사원중에서 이름이 K로 시작하는 사원의 사원번호와 이름을 출력하세요?

<정답>

3. 와일드 카드를 사용하여 이름중에서 K를 포함하는 사원의 사원번호와 이름을 출력 하세요?

<정답>

4. 와일드 카드를 사용하여 이름중에서 끝에서 두번째 글자가 K로 끝나는 사원의 사원번호와 이름을 출력 하세요?

<정답>

5 SQL 주요 함수

5.1 DUAL 테이블과 SQL 함수 분류

- DUAL 테이블은 DUMMY라는 단 하나의 컬럼에 X라는 단 하나의 로우만을 저장하고 있으나 이 값은 아무런 의미가 없다.
- DUAL 테이블은 산술 연산의 결과를 한 줄로 얻기 위해서 오라클에서 제공하는 테이블이다.

```
SELECT *  
FROM DUAL;  
  
SELECT 24*60  
FROM EMP;  
  
SELECT 24*60  
FROM DUAL;  
  
SELECT SYSDATE  
FROM DUAL;  
-- 현재 날짜를 얻는 쿼리문
```

5.2 숫자 함수

구 분	설 명
ABS	절대값을 구한다.
COS	COSINE 값을 반환한다.
EXP	e(2.71828183...)의 n승을 반환한다.
FLOOR	소수점 아래를 잘라낸다.(버림)
LOG	LOG값을 반환한다.
POWER	POWER(m, n) m의 n승을 반환한다.
SIGN	SIGN (n) n<0이면 -1, n=0이면 0, n>0이면 1을 반환한다.
SIN	SINE값을 반환한다.
TAN	TANGENT값을 반환한다.
ROUND	특정 자릿수에서 반올림한다.
TRUNC	특정 자릿수에서 잘라낸다. (버림)
MOD	입력 받은 수를 나눈 나머지 값을 반환한다.

5.2.1 절댓값 구하는 ABS 함수

- ABS 함수는 절대값을 구한다. 절대값은 방향은 없고 크기만 있는 것으로서 주어진 데이터가 음수일 경우 양수로 표현한다.

```
SELECT -10, ABS(-10)  
FROM DUAL;
```

5.2.2 소수점 아래를 버리는 FLOOR 함수

- FLOOR 함수는 소수점 아래를 버린다. 34.5678를 FLOOR 함수에 적용하면 34가 구해진다.

```
SELECT 34.5678, FLOOR(34.5678)
FROM DUAL;
```

5.2.3 특정 자릿수에서 반올림하는 ROUND 함수

- 34.5678를 반올림하면 35이다. 이와 같이 반올림한 결과를 구하기 위한 함수로 오라클에서는 ROUND가 제공된다.

```
SELECT 34.5678, ROUND(34.5678)
FROM DUAL;

SELECT 34.5678, ROUND(34.5678, 2)
FROM DUAL;

SELECT 34.5678, ROUND(34.5678, -1)
FROM DUAL;
```

5.2.4 특정 자릿수에서 잘라내는 TRUNC 함수

- TRUNC 함수는 지정한 자리 수 이하를 버린 결과를 구해주는 함수이다.

```
SELECT TRUNC(34.5678, 2), TRUNC(34.5678, -1), TRUNC(34.5678)
FROM DUAL;
```

5.2.5 나머지 구하는 MOD 함수

- MOD 함수는 나누기 연산을 한 후에 구한 몫이 아닌 나머지를 결과로 되돌려주는 함수이다.

```
SELECT MOD(27, 2), MOD(27, 5), MOD(27, 7)
FROM DUAL;
```

5.3 문자 처리 함수

5.3.1 UPPER 함수

- UPPER 함수는 입력한 문자값을 대문자로 변환하는 함수이다.

```
SELECT 'Welcome to Oracle', UPPER('Welcome to Oracle')
FROM DUAL;
```

5.3.2 LOWER 함수

- LOWER 함수는 문자열을 모두 소문자로 변경한다.

```
SELECT 'Welcome to Oracle', LOWER('Welcome to Oracle')
FROM DUAL;
```

5.3.3 INITCAP 함수

- INITCAP 함수는 문자열의 이니셜만 대문자로 변경한다.

```
SELECT 'WELCOME TO ORACLE', INITCAP('WELCOME TO ORACLE')
FROM DUAL;
```

5.3.4 LENGTH 함수

- LENGTH 함수는 컬럼에 저장된 데이터 값이 몇 개의 문자로 구성되었는지 길이를 알려주는 함수이다.

```
SELECT LENGTH('Oracle'), LENGTH('오라클')
FROM DUAL;
```

5.3.5 LENGTHB 함수

- LENGTHB 함수는 바이트 수를 알려주는 함수이다. 특히, 한글 1자는 2바이트를 차지한다. 그렇기 때문에 수행 결과를 보면 한글 3자로 구성된 '오라클'의 LENGTHB 함수의 결과는 9이 된다.

```
SELECT LENGTHB('Oracle'), LENGTHB('오라클')
FROM DUAL;
```

5.3.6 SUBSTR, SUBSTRB 함수

- SUBSTR 과 SUBSTRB 함수는 대상 문자열이나 컬럼의 자료에서 시작위치부터 선택 개수만큼의 문자를 추출한다. SUBSTRB 함수도 같은 형식이지만 명시된 개수만큼의 문자가 아닌 바이트 수를 잘라낸다는 점에서만 차이가 있다.

```
SELECT SUBSTR('Welcome to Oracle', 4, 3)
FROM DUAL;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
W	e	l	c	o	m	e		t	o		O	r	a	c	l	e

5.3.7 INSTR 함수

- INSTR 함수는 대상 문자열이나 칼럼에서 특정 문자가 나타나는 위치를 알려준다.

```
SELECT INSTR('WELCOME TO ORACLE', 'O')
FROM DUAL;
```

5.3.8 INSTRB 함수

- INSTRB 함수 역시 SUBSTRB 함수에서와 마찬가지로 문자의 위치를 알아내기 위한 바이트 기준으로 한다.

```
SELECT INSTR('데이터베이스', '이', 3, 1), INSTRB('데이터베이스', '이', 3, 1)
FROM DUAL;
```

5.3.9 LPAD/RPAD 함수

- LPAD(LEFT PADDING) 함수는 칼럼이나 대상 문자열을 명시된 자릿수에서 오른쪽에 나타내고, 남은 왼쪽 자리를 특정 기호로 채운다. RPAD(RIGHT PADDING) 함수는 반대로 칼럼이나 대상 문자열을 명시된 자릿수에서 왼쪽에 나타내고, 남은 오른쪽 자리를 특정 기호로 채운다.

```
SELECT LPAD('Oracle', 20, '#')
FROM DUAL;
SELECT RPAD('Oracle', 20, '#')
FROM DUAL;
```

5.3.10 LTRIM와 RTRIM 함수

- LTRIM 함수는 문자열의 왼쪽(앞)의 공백 문자들을 삭제한다. RTRIM 함수 역시 오른쪽(뒤)의 공백 문자를 잘라낸다.

```
SELECT LTRIM(' Oracle ')
FROM DUAL;
SELECT RTRIM(' Oracle ')
FROM DUAL;
```

5.3.11 TRIM 함수

- TRIM 함수는 칼럼이나 대상 문자열에서 특정 문자가 첫 번째 글자이거나 마지막 글자이면 잘라내고 남은 문자열만 반환한다.

```
SELECT TRIM('a' FROM 'aaaaOracleaaaa')
FROM DUAL;
```

5.3.12 REPLACE 함수

- REPLACE()는 문자열을 치환하는 함수이다. 예를 들어 담당자의 실수로 도서의 제목을 잘못 입력한 경우 REPLACE 함수를 사용하면 일일이 변경하지 않고 한꺼번에 변경할 수 있다.

```
CONN MADANG/MADANG;
SELECT BOOKID, REPLACE(BOOKNAME, '야구', '농구') BOOKNAME, PUBLISHER, PRICE FROM BOOK;
```

5.4 날짜 함수

5.4.1 현재 날짜를 반환하는 SYSDATE 함수

- SYSDATE 함수는 시스템에 저장된 현재 날짜를 반환하는 함수이다.

```
SELECT SYSDATE
FROM DUAL;
```

5.4.2 날짜 연산

- 날짜 형 데이터에 숫자를 더하면(날짜+숫자) 그 날짜로부터 그 기간만큼 지난 날짜를 계산한다. 날짜 형 데이터에 숫자를 빼면(날짜-숫자) 그 날짜로부터 그 기간만큼 이전 날짜를 구한다.

```
SELECT SYSDATE-1 어제, SYSDATE 오늘, SYSDATE+1 내일
FROM DUAL;
```

5.4.3 특정 기준으로 반올림하는 ROUND 함수

- ROUND 함수는 숫자를 반올림하는 함수로 학습하였다. 하지만, 이 함수에 포맷 모델을 지정하면 숫자 이외에 날짜에 대해서도 반올림을 할 수 있다.

```
-- 형식
ROUND (date, format)

-- 예
```

```
SELECT HIREDATE, ROUND(HIREDATE, 'MONTH')
FROM EMP;
```

5.4.4 특정 기준으로 버리는 TRUNC 함수

- TRUNC 함수 역시 숫자를 잘라내는 것뿐만 아니라 날짜를 잘라낼 수 있다. ROUND 함수와 마찬가지로 포맷 형식을 주어 다양한 기준으로 날짜를 잘라낼 수 있다.

```
-- 형식
TRUNC (date, format)

-- 예
SELECT HIREDATE, TRUNC(HIREDATE, 'MONTH')
FROM EMP;
```

5.4.5 두 날짜 사이의 날수를 구하는 MONTHS_BETWEEN 함수

- MONTHS_BETWEEN 함수는 날짜와 날짜 사이의 개월 수를 구하는 함수이다.

```
-- 형식
MONTHS_BETWEEN (date1, date2)

-- 예
SELECT ENAME, SYSDATE, HIREDATE, MONTHS_BETWEEN (SYSDATE, HIREDATE)
FROM EMP;
```

5.4.6 개월 수를 더하는 ADD_MONTHS 함수

- ADD_MONTHS 함수는 특정 개월 수를 더한 날짜를 구하는 함수이다.

```
-- 형식
ADD_MONTHS (date, number)

-- 예
SELECT ENAME, HIREDATE, ADD_MONTHS(HIREDATE, 6)
FROM EMP;
```

5.4.7 해당 요일의 가장 가까운 날짜를 반환하는 NEXT_DAY 함수

- NEXT_DAY 함수는 해당 날짜를 기준으로 최초로 도래하는 요일에 해당되는 날짜를 반환하는 함수이다.

```
-- 형식
NEXT_DAY (date, 요일)

-- 예
```

```
SELECT SYSDATE, NEXT_DAY(SYSDATE, '수요일')
FROM DUAL;
```

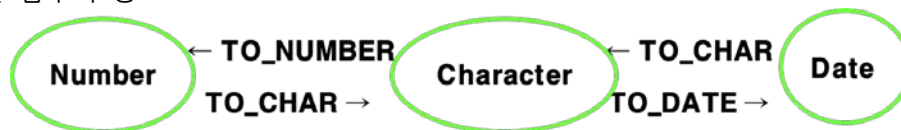
5.4.8 해당 달의 마지막 날짜를 반환하는 LAST_DAY 함수

- LAST_DAY 함수는 해당 날짜가 속한 달의 마지막 날짜를 반환하는 함수이다.

```
SELECT HIREDATE, LAST_DAY(HIREDATE)
FROM EMP;
```

5.5 형 변환 함수

- 오라클을 사용하다 보면 숫자, 문자, 날짜의 데이터 형을 다른 데이터형으로 변환해야 하는 경우가 생긴다.
- 이럴 때 사용하는 함수가 형 변환 함수이다. 형 변환 함수로는 TO_NUMBER, TO_CHAR, TO_DATE 가 있다.
- 형 변환 함수의 종류



5.5.1 문자형으로 변환하는 TO_CHAR 함수

(1) 날짜형을 문자형으로 변환하기

- DATE 형태의 데이터를 지정한 양식에 의해 VARCHAR2 형의 문자로 변환한다.

```
-- 형식
TO_CHAR ( 날짜 데이터, '출력형식' )

-- 예
SELECT SYSDATE, TO_CHAR(SYSDATE, 'YYYY-MM-DD')
FROM DUAL;
```

(2) 숫자형을 문자형으로 변환하기

```
SELECT ENAME, SAL, TO_CHAR (SAL, 'L999,999')
FROM EMP;

SELECT TO_CHAR (123456, '000000000'), TO_CHAR (123456, '999,999,999')
FROM DUAL;
```


5.5.2 날짜형으로 변환하는 TO_DATE 함수

- 날짜 형은 세기, 년도, 월, 일, 시간, 분, 초와 같이 날짜와 시간에 대한 정보를 저장한다.
- 오라클에서 기본 날짜 형식은 'YY/MM/DD' 형식으로 '년/월/일' 예를 들면 '06/03/08' 식으로 나타낸다. 만일 년도를 4자리로 출력하려면 'YYYY/MM/DD' 형식으로 지정한다.
- TO_DATE 함수는 문자열을 날짜 형으로 변환한다.

```
SELECT ENAME, HIREDATE FROM EMP
WHERE HIREDATE=TO_DATE(19810220,'YYYYMMDD');

SELECT TRUNC(SYSDATE-TO_DATE('2008/01/01','YYYY/MM/DD'))
FROM DUAL;
```

5.5.3 숫자형으로 변환하는 TO_NUMBER 함수

- TO_NUMBER 함수는 특정 데이터를 숫자형으로 변환해 주는 함수이다.

```
SELECT TO_NUMBER('20,000', '99,999') - TO_NUMBER('10,000', '99,999')
FROM DUAL;
```

5.6 NULL을 다른 값으로 변환하는 NVL 함수

- NVL 함수는 NULL을 0 또는 다른 값으로 변환하기 위해서 사용하는 함수이다.

```
SELECT ENAME, SAL, COMM, SAL*12+COMM,
NVL(COMM, 0), SAL*12+NVL(COMM, 0)
FROM EMP
ORDER BY JOB;
```

5.7 선택을 위한 DECODE 함수

- DECODE 함수는 프로그램 언어에서 가장 많이 사용되는 switch case 문과 같은 기능을 갖는다. 즉, 여러 가지 경우에 대해서 선택할 수 있도록 한다.

```
-- 형식
DECODE (표현식, 조건1, 결과1,
        조건2, 결과2,
        조건3, 결과3,
        기본결과n
)

-- 예
SELECT ENAME, DEPTNO,
       DECODE(DEPTNO, 10, 'ACCOUNTING',
                20, 'RESEARCH',
                30, 'SALES',
                40, 'OPERATIONS' )
```

```
AS DNAME  
FROM EMP;
```

5.8 조건에 따라 서로 다른 처리가 가능한 CASE 함수

- CASE 함수 역시 여러 가지 경우에 대해서 하나를 선택하는 함수이다.
- DECODE 함수와 차이점이 있다면 DECODE 함수는 조건이 일치(= 비교 연산자)하는 경우에 대해서만 적용되는 반면, CASE 함수는 다양한 비교 연산자를 이용하여 조건을 제시할 수 있으므로 범위를 지정할 수도 있다.
- CASE 함수는 프로그램 언어의 if else if else 와 유사한 구조를 갖는다.

```
-- 형식  
CASE 표현식 WHEN 조건1 THEN 결과1  
            WHEN 조건2 THEN 결과2  
            WHEN 조건3 THEN 결과3  
            ELSE 결과n  
END  
  
-- 예  
SELECT ENAME, DEPTNO,  
       CASE WHEN DEPTNO=10 THEN 'ACCOUNTING'  
            WHEN DEPTNO=20 THEN 'RESEARCH'  
            WHEN DEPTNO=30 THEN 'SALES'  
            WHEN DEPTNO=40 THEN 'OPERATIONS'  
       END AS DNAME  
FROM EMP;
```

[과제] 과제-05-01.TXT

SQL> CONN SCOTT/TIGER 로 접속하여 SQL문을 작성하세요.

1. 사원테이블(EMP)에서 입사일(HIREDATE)을 4자리 연도로 출력되도록 SQL문을 작성하세요? (ex. 1980/01/01)

<정답>

2. 사원테이블(EMP)에서 MGR컬럼의 값이 null 인 데이터의 MGR의 값을 CEO로 출력하는 SQL문을 작성 하세요?

<정답>

3. 사원 테이블(EMP)에서 가장 최근에 입사한 직원명을 출력하는 SQL문을 작성 하세요? (힌트: 서브쿼리와 MAX()함수 이용)

<정답>

4. 사원 테이블(EMP)에서 최대 급여를 받는 직원명과 최대급여금액을 출력하는 SQL문을 작성하세

요? (힌트: 서브쿼리와 MAX()함수 이용)

<정답>

[과제] 과제-05-02.TXT

SQL> CONN ORA_USER/HONG 로 접속하여 SQL 문을 작성하세요.

1. 직원테이블(employees)에는 phone_number 라는 컬럼에 사원의 전화번호가 ###.###.#### 형태로 저장되어 있다. 여기서 처음 3 자리 숫자 대신 서울 지역번호인 (02)를 붙여 전화번호를 출력하도록 쿼리를 작성해 보자.

<정답>

2. 현재일자 기준으로 직원테이블의 입사일자(hire_date)를 참조해서 근속년수가 10 년 이상인 직원을 다음과 같은 형태의 결과를 출력하도록 쿼리를 작성해보자. (근속년수가 많은 직원순서대로 결과를 나오도록 하자)

사원번호 사원명 입사일자 근속년수

<정답>

3. 고객 테이블(CUSTOMERS)에는 고객 전화번호(cust_main_phone_number) 컬럼이 있다. 이 컬럼 값은 ###-###-#### 형태인데, '-' 대신 '/'로 바꿔 출력하는 쿼리를 작성해 보자. (힌트: REPLACE() 함수 사용)

<정답>

4. 고객 테이블(CUSTOMERS)의 고객 전화번호(cust_main_phone_number) 컬럼을 다른 문자로 대체(일종의 암호화)하도록 쿼리를 작성해 보자. (힌트: TRANSLATE() 함수 사용)

<정답>

5. 고객 테이블(CUSTOMERS)에는 고객의 출생년도(cust_year_of_birth) 컬럼이 있다. 현재일 기준으로 이 컬럼을 활용해 30 대, 40 대, 50 대를 구분해 출력하고, 나머지 연령대는 '기타'로 출력하는 쿼리를 작성해보자.

<정답>

6. 5 번 문제는 30~50 대 까지만 표시했는데, 전 연령대를 표시하도록 쿼리를 작성하는데, 이번에는 DECODE 대신 CASE 표현식을 사용해보자.

<정답>

6. 그룹 쿼리와 집합 연산자

6.1. 기본 집계 함수

- 그룹 함수는 하나 이상의 행을 그룹으로 묶어 연산하여 총합, 평균 등 하나의 결과로 나타난다.
- 그룹 함수의 종류

구분	설명
SUM	그룹의 누적 합계를 반환한다.
AVG	그룹의 평균을 반환한다.
COUNT	그룹의 총 개수를 반환한다.
MAX	그룹의 최대값을 반환한다.
MIN	그룹의 최소값을 반환한다.
STDDEV	그룹의 표준편차를 반환한다.
VARIANCE	그룹의 분산을 반환한다.

6.1.1. 합계를 구하는 SUM 함수

- SUM 함수는 해당 칼럼 값들에 대한 총합을 구하는 함수이다.
- 그룹 함수는 다른 연산자와는 달리 해당 칼럼 값이 NULL 인 것을 제외하고 계산한다.

```
SELECT SUM(SAL)
FROM EMP;

SELECT SUM(COMM)
FROM EMP; -- 그룹 함수와 NULL
```

6.1.2. 평균 구하는 AVG 함수

- AVG 함수는 해당 칼럼 값들에 대해 평균을 구하는 함수이다.
- 해당 칼럼 값이 NULL 인 것에 대해서는 제외하고 계산한다.

```
SELECT AVG(SAL)
FROM EMP;
```

6.1.3. 최대값 구하는 MAX, 최소값 구하는 MIN 함수

- 지정한 칼럼 값들 중에서 최대값을 구하는 함수가 MAX이고, 최소값을 구하는 함수가 MIN이다.

```
SELECT MAX(SAL), MIN(SAL)
FROM EMP;
```

6.1.4. 그룹 함수와 단순 컬럼

- SELECT문에 그룹 함수를 사용하는 경우, 그룹 함수를 적용하지 않은 단순 컬럼은 올 수 없다.

```
SELECT MAX(SAL)
FROM EMP;

SELECT ENAME, MAX(SAL)
FROM EMP; -- 오류 발생
```

6.1.5. 로우 개수 구하는 COUNT 함수

- COUNT 함수는 테이블에서 조건을 만족하는 행의 개수를 반환하는 함수이다.

```
SELECT COUNT(COMM)
FROM EMP;

SELECT COUNT(*), COUNT(COMM)
FROM EMP;

SELECT COUNT(JOB) 업무수
FROM EMP;

SELECT COUNT(DISTINCT JOB) 업무수
FROM EMP;
```

6.2. GROUP BY절

- 그룹함수를 쓰되 어떤 컬럼 값을 기준으로 그룹함수를 적용할 경우 GROUP BY 절 뒤에 해당 컬럼을 기술하면 된다.
- 합계, 평균, 최대값이나, 최소값 등을 어떤 칼럼을 기준으로 그 칼럼의 값 별로 보고자 할 때 GROUP BY 절 뒤에 해당 칼럼을 기술하면 된다.
- GROUP BY 절을 사용할 때 주의할 점은 GROUP BY 절 다음에는 칼럼의 별칭을 사용할 수 없고, 반드시 칼럼명을 기술해야 한다.

```
-- 형식
SELECT 칼럼명, 그룹함수
FROM 테이블명
WHERE 조건 (연산자)
GROUP BY 칼럼명;

SELECT DEPTNO
FROM EMP
GROUP BY DEPTNO;

SELECT DEPTNO, AVG(SAL)
FROM EMP
GROUP BY DEPTNO;

SELECT DEPTNO, MAX(SAL), MIN(SAL)
FROM EMP
GROUP BY DEPTNO;
```

6.3. HAVING 조건

- SELECT 절에 조건을 사용하여 결과를 제한할 때는 WHERE 절을 사용하지만 그룹의 결과를 제한할 때는 HAVING 절을 사용한다.

```
SELECT DEPTNO, AVG(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING AVG(SAL) >= 2000;

SELECT DEPTNO, MAX(SAL), MIN(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MAX(SAL) > 2900;
```

6.4. 집합 연산자

6.4.1 UNION

- 집합의 합집합 개념
- 두 개 이상의 개별 SELECT 쿼리를 연결
- 개별 SELECT 쿼리 반환 결과가 중복될 경우 UNION 연산 결과는 한 로우만 반환됨

```
-- 집합 연산자
DROP TABLE EXP_GOODS_ASIA;
CREATE TABLE EXP_GOODS_ASIA (
    COUNTRY VARCHAR2(10),
    SEQ      NUMBER,
    GOODS    VARCHAR2(80));

INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 1, '원유제외 석유류');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 2, '자동차');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 3, '전자집적회로');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 4, '선박');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 5, 'LCD');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 6, '자동차부품');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 7, '휴대전화');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 8, '환식탄화수소');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 9, '무선송신기 디스플레이 부속품');
INSERT INTO EXP_GOODS_ASIA VALUES ('한국', 10, '철 또는 비합금강');

INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 1, '자동차');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 2, '자동차부품');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 3, '전자집적회로');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 4, '선박');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 5, '반도체웨이퍼');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 6, '화물차');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 7, '원유제외 석유류');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 8, '건설기계');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 9, '다이오드, 트랜지스터');
INSERT INTO EXP_GOODS_ASIA VALUES ('일본', 10, '기계류');

COMMIT;
```

```

SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
 ORDER BY SEQ;

SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본'
 ORDER BY SEQ;

-- UNION
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
UNION
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';

```

6.4.2 UNION ALL

- UNION과 유사
- 개별 SELECT 쿼리 반환 결과가 중복될 경우, 중복되는 건까지 모두 반환

```

-- UNION ALL
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
UNION ALL
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';

```

6.4.3 INTERSECT

- 집합의 교집합 개념
- 두 개 이상의 개별 SELECT 쿼리를 연결
- 개별 SELECT 쿼리 반환 결과 중 공통된 항목만 추출

```

-- INTERSECT
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
INTERSECT
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';

```

6.4.4 MINUS

- 집합의 차집합 개념

- 두 개 이상의 개별 SELECT 쿼리를 연결
- 개별 SELECT 쿼리 반환 결과 중 중복된 건을 제외한 선행 쿼리 결과 추출

```
-- MINUS
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
MINUS
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';

SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본'
MINUS
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국';
```

6.4.5 집합 연산자 제한사항

- 개별 SELECT 쿼리의 SELECT 리스트 개수와 데이터 타입이 일치해야 한다.
- ORDER BY 절은 맨 마지막 개별 SELECT 쿼리에만 명시 가능하다.
- BLOB, CLOB, BFILE 타입의 컬럼에 대해서는 집합 연산자를 사용할 수 없다.
- UNION, INTERSECT, MINUS 연산자는 LONG형 컬럼에는 사용할 수 없다.

```
-- 집합 연산자 제한사항
--1. 집합 연산자로 연결되는 각 SELECT문의 SELECT 리스트의 개수와 데이터 타입은 일치해야 한다.
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
UNION
SELECT SEQ, GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';
--ORA-01789: 질의 블록은 부정확한 수의 결과 열을 가지고 있습니다.
--01789. 00000 - "query block has incorrect number of result columns"

SELECT SEQ
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
UNION
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';
--ORA-01790: 대응하는 식과 같은 데이터 유형이어야 합니다
--01790. 00000 - "expression must have same datatype as corresponding expression"

--2. 집합 연산자로 SELECT 문을 연결할 때 ORDER BY절은 맨 마지막 문장에서만 사용할 수 있다.
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
 ORDER BY GOODS
UNION
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본';
```



```
--ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다
--00933. 00000 - "SQL command not properly ended"
```

```
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '한국'
UNION
SELECT GOODS
  FROM EXP_GOODS_ASIA
 WHERE COUNTRY = '일본'
ORDER BY GOODS;
```

[과제] 과제-06-01.TXT

SQL> CONN ORA_USER/HONG 로 접속하여 SQL문을 작성하세요.

1. 사원테이블에서 입사년도별 사원수를 구하는 쿼리를 작성해보자.

<정답>

2. kor_loan_status 테이블에서 2012년도 월별, 지역별 대출 총 잔액을 구하는 쿼리를 작성하라.

<예시>

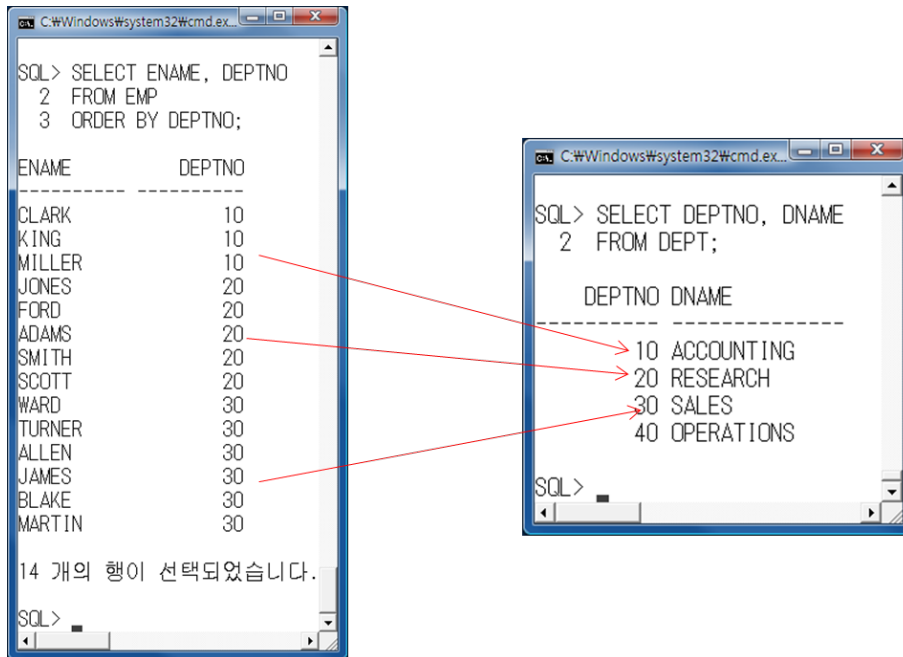
201211	전북	23762
201211	제주	6351.7
201211	충남	30933.2
201211	충북	18850.3
201212	강원	17449.7
201212	경기	281501.2
201212	경남	51368.5

<정답>

7 조인

7.1 조인의 필요성

- 특정 부서 번호에 대한 부서이름은 무엇인지는 부서(DEPT) 테이블에 있다. 특정 사원에 대한 부서명을 알아내기 위해서는 부서 테이블에서 정보를 얻어 와야 한다.
- SQL에서는 두 개 이상의 테이블을 결합해야만 원하는 결과를 얻을 수 있을 때 한 번의 질의로 원하는 결과를 얻을 수 있는 조인 기능을 제공한다.



7.2 Cross Join

- 특별한 키워드 없이 SELECT 문의 FROM 절에 사원(EMP) 테이블과 부서(DEPT) 테이블을 콤마로 연결하여 연속하여 기술하는 것이다.

```
SELECT *
FROM EMP, DEPT;
```

7.3 Equi Join

- EQUI JOIN은 가장 많이 사용하는 조인 방법으로서 조인 대상이 되는 두 테이블에서 공통적으로 존재하는 컬럼의 값이 일치되는 행을 연결하여 결과를 생성하는 조인 방법이다.

```
SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;

SELECT ENAME, DNAME
FROM EMP, DEPT
```

```

WHERE EMP.DEPTNO=DEPT.DEPTNO
AND ENAME='SCOTT';

SELECT ENAME, DNAME, DEPTNO -- 오류발생, 컬럼명의 모호성 문제
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
AND ENAME='SCOTT';

SELECT EMP.ENAME, DEPT.DNAME, EMP.DEPTNO -- 컬럼명의 모호성 해결
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO
AND ENAME='SCOTT';

SELECT E.ENAME, D.DNAME, E.DEPTNO, D.DEPTNO
FROM EMP E, DEPT D -- 테이블에 별칭을 부여함
WHERE E.DEPTNO = D.DEPTNO
AND E.ENAME='SCOTT';

```

[과제] 과제-07-01.TXT

SQL> CONN SCOTT/TIGER 로 접속하여 SQL문을 작성하세요.

1. 조인을 사용하여 뉴욕에서 근무하는 사원의 이름과 급여를 출력하세요.

<정답>

2. 조인을 사용하여 ACCOUNTING 부서 소속 사원의 이름과 입사일을 출력하세요.

<정답>

3. 직급이 MANAGER인 사원의 이름, 부서명을 출력하세요.

<정답>

7.4 Non-Equi Join

- Non-Equi Join은 조인 조건에 특정 범위 내에 있는지를 조사하기 위해서 WHERE 절에 조인 조건을 = 연산자 이외의 비교 연산자를 사용한다.

```

SELECT * FROM SALGRADE;

SELECT ENAME, SAL, GRADE
FROM EMP, SALGRADE
WHERE SAL BETWEEN LOSAL AND HISAL;

SELECT E.ENAME, E.SAL, S.GRADE
FROM EMP E, SALGRADE S
WHERE E.SAL >= S.LOSAL AND E.SAL <= S.HISAL;

```

7.5 Self Join

- 조인은 두 개 이상의 서로 다른 테이블을 서로 연결하는 것뿐만 아니라, 하나의 테이블 내에서 조인을 해야만 원하는 자료를 얻는 경우가 생긴다. Self Join이란 말 그대로 자기 자신과 조인을 맺는 것을 말한다.

ENAME	MGR
SMITH	7902
ALLEN	7698
WARD	7698
JONES	7839
MARTIN	7698
BLAKE	7839
CLARK	7839
SCOTT	7566
KING	
TURNER	7698
ADAMS	7788
JAMES	7698
FORD	7566
MILLER	7782

EMPNO	ENAME
7369	SMITH
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES
7902	FORD
7934	MILLER

```
SELECT EMPLOYEE.ENAME || '의 매니저는 ' || MANAGER.ENAME || '입니다.'  
FROM EMP EMPLOYEE, EMP MANAGER  
WHERE EMPLOYEE.MGR = MANAGER.EMPNO;
```

[과제] 과제-07-03.TXT

SQL> CONN SCOTT/TIGER 로 접속하여 SQL문을 작성하세요.

1. 매니저가 KING인 사원들의 이름과 직급을 출력하세요.

<정답>

2. SCOTT과 동일한 근무지에서 근무하는 사원의 이름을 출력하세요.

<정답>

7.6 Outer Join

- 조인 조건에 만족하지 못하였더라도 해당 로우를 나타내고 싶을 때에 사용하는 것이 외부 조인(Outer Join)이다. 외부 조인은 NULL 값이기에 배제된 행을 결과에 포함시킬 수 있으며 "+" 기호를 조인 조건에서 정보가 부족한 칼럼 이름 뒤에 덧붙인다.

```
SELECT EMPLOYEE.ENAME || '의 매니저는 ' || MANAGER.ENAME || '입니다.'
FROM EMP EMPLOYEE, EMP MANAGER
WHERE EMPLOYEE.MGR = MANAGER.EMPNO(+);
```

7.7 ANSI Join

- 오라클뿐만 아니라 현재 대부분의 상용 데이터베이스 시스템에서 표준 언어로 ANSI(미국표준 협회) SQL에서 제시한 표준 기능을 대부분 준수하고 있다.

7.7.1 ANSI Cross Join

```
-- Oracle Cross Join
SELECT *
FROM EMP, DEPT;

-- ANSI Cross Join
SELECT *
FROM EMP CROSS JOIN DEPT;
```

7.7.2 ANSI Inner Join

- Using을 이용한 조인 조건(공통컬럼) 지정하기
- Natural Join : 자동적으로 모든 컬럼을 대상으로 공통 컬럼을 조사하여 내부적으로 조인문을 생성한다.

```
-- Oracle Equi Join
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;

-- ANSI Inner Join
SELECT ENAME, DNAME
FROM EMP INNER JOIN DEPT
ON EMP.DEPTNO=DEPT.DEPTNO;

-- USING을 이용한 조인 조건(공통컬럼) 지정
SELECT EMP.ENAME, DEPT.DNAME
FROM EMP INNER JOIN DEPT
USING (DEPTNO);

-- NATURAL Join
SELECT EMP.ENAME, DEPT.DNAME
FROM EMP NATURAL JOIN DEPT;
```

7.7.3 ANSI Outer Join

- 새로운 ANSI 구문에서 Outer Join은 LEFT Outer Join, RIGHT Outer Join 그리고 FULL Outer Join 세 가지 타입의 조인을 제공한다.

```

DROP TABLE DEPT01;
CREATE TABLE DEPT01(
    DEPTNO NUMBER(2),
    DNAME VARCHAR2(14)
);
INSERT INTO DEPT01 VALUES(10,'ACCOUNTING');
INSERT INTO DEPT01 VALUES(20,'RESEARCH');
SELECT * FROM DEPT01;

DROP TABLE DEPT02;
CREATE TABLE DEPT02(
    DEPTNO NUMBER(2),
    DNAME VARCHAR2(14)
);
INSERT INTO DEPT02 VALUES(10,'ACCOUNTING');
INSERT INTO DEPT02 VALUES(30,'SALES');
SELECT * FROM DEPT02;

-- Left Outer Join
SELECT *
FROM DEPT01 LEFT OUTER JOIN DEPT02
ON DEPT01.DEPTNO = DEPT02.DEPTNO;

-- Right Outer Join
SELECT *
FROM DEPT01 RIGHT OUTER JOIN DEPT02
USING(DEPTNO);

-- Full Outer Join
SELECT *
FROM DEPT01 FULL OUTER JOIN DEPT02
USING(DEPTNO);

```

[과제] 과제-07-02.TXT

SQL> CONN SCOTT/TIGER 로 접속하여 SQL문을 작성하세요.

1. 직급이 MANAGER인 사원의 이름, 부서명을 출력하는 SQL문을 작성 하세요? (ORACLE EQUI JOIN, ANSI INNER JOIN, ANSI NATURAL JOIN을 사용하여 처리)

<정답>

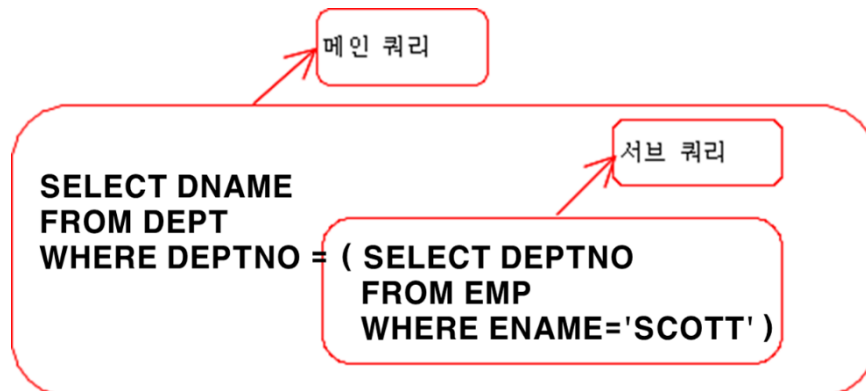
2. SMITH와 동일한 직급(JOB컬럼)을 가진 사원의 이름과 직급을 출력하는 SQL문을 작성 하세요?

<정답>

8 서브 쿼리

8.1 서브 쿼리의 기본 개념

- 서브 쿼리는 하나의 SELECT 문장의 절 안에 포함된 또 하나의 SELECT 문장이다.
- 서브 쿼리를 포함하고 있는 쿼리문을 메인 쿼리, 포함된 또 하나의 쿼리를 서브 쿼리라 한다.
- 서브 쿼리는 비교 연산자의 오른쪽에 기술해야 하고 반드시 괄호로 둘러싸아야 한다
- 서브 쿼리는 메인 쿼리가 실행되기 이전에 한번만 실행이 된다.



8.2 단일행 서브 쿼리

- 단일 행(Single Row) 서브 쿼리는 수행 결과가 오직 하나의 로우(행, row)만을 반환하는 서브 쿼리를 갖는 것을 말한다.
- 단일 행 서브 쿼리문에서는 이렇게 오직 하나의 로우(행, row)로 반환되는 서브 쿼리의 결과는 메인 쿼리에 보내게 되는데 메인 쿼리의 WHERE 절에서는 단일 행 비교 연산자인 =, >, >=, <, <=, <>를 사용해야 한다.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ( SELECT AVG(SAL)
              FROM EMP );
```

8.3 다중행 서브 쿼리

- 다중 행 서브 쿼리는 서브 쿼리에서 반환되는 결과가 하나 이상의 행일 때 사용하는 서브 쿼리이다. 다중 행 서브 쿼리는 반드시 다중 행 연산자(Multiple Row Operator)와 함께 사용해야 한다.

종류	의미
IN	메인 쿼리의 비교 조건('=' 연산자로 비교할 경우)이 서브 쿼리의 결과 중에서 하나라도 일치하면 참이다.
ANY, SOME	메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 하나 이상이 일치하면 참이다.
ALL	메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 모든 값이 일치하면 참이다.
EXIST	메인 쿼리의 비교 조건이 서브 쿼리의 결과 중에서 만족하는 값이 하나라도 존재하면 참이다.

8.3.1 IN 연산자

- IN 연산자는 메인 쿼리의 비교 조건에서 서브 쿼리의 출력 결과와 하나라도 일치하면 메인 쿼리의 WHERE절이 참이 되는 연산자이다.

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE DEPTNO IN ( SELECT DISTINCT DEPTNO
FROM EMP
WHERE SAL >= 3000 );
```

8.3.2 ALL 연산자

- ALL 조건은 메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 모든 값이 일치하면 참이다.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ALL (SELECT SAL
FROM EMP
WHERE DEPTNO = 30);
```

8.3.3 ANY 연산자

- ANY 조건은 메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 하나 이상만 일치하면 참이다.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ANY ( SELECT SAL
FROM EMP
WHERE DEPTNO = 30 );
```

8.3.4 EXISTS 연산자

- 서브쿼리에 결과 값이 하나 이상 존재하면 조건식이 모두 true, 존재하지 않으면 모두 false가 되는 연산자이다.

```
SELECT *
FROM EMP
WHERE EXISTS (SELECT DNAME FROM DEPT WHERE DEPTNO = 10);

SELECT *
FROM EMP
WHERE EXISTS (SELECT DNAME FROM DEPT WHERE DEPTNO = 50);
```



```
CONN MADANG/MADANG
```

```
SELECT NAME, ADDRESS  
FROM CUSTOMER CS  
WHERE EXISTS (SELECT * FROM ORDERS OD WHERE CS.CUSTID=OD.CUSTID);
```

```
-- EXISTS 연산자를 사용하여 대한민국에 거주하는 고객에게 판매한 도서의 총 판매액을 구하기  
SELECT SUM(SALEPRICE) "TOTAL"  
FROM ORDERS OD  
WHERE EXISTS (  
    SELECT *  
    FROM CUSTOMER CS  
    WHERE ADDRESS LIKE '%대한민국%' AND CS.CUSTID=OD.CUSTID  
);
```

[과제] 과제-08-01.TXT

SQL> CONN ORA_USER/HONG 로 접속하여 SQL문을 작성하세요.

1. 101번 사원에 대해 아래의 결과를 산출하는 쿼리를 작성해 보자.

사번	사원명	job명칭	job시작일자	job종료일자	job수행부서명
----	-----	-------	---------	---------	----------

<정답>

2. 다음의 쿼리를 ANSI 문법으로 변경해 보자.

```
SELECT A.DEPARTMENT_ID, A.DEPARTMENT_NAME  
FROM DEPARTMENTS A, EMPLOYEES B  
WHERE A.DEPARTMENT_ID = B.DEPARTMENT_ID  
AND B.SALARY > 3000  
ORDER BY A.DEPARTMENT_NAME;
```

<정답>

[과제] 과제-08-02.TXT

1. 다음의 두 쿼리를 ANY, ALL을 사용해서 동일한 결과를 추출하도록 변경해보자.
(단, SQL> CONN ORA USER/HONG 로 접속하여 SQL문을 작성하세요.)

```
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;
```

<정답>

2. IN연산자를 이용하여 부서별로 가장 많은 급여를 받는 사원의 정보(사원번호,사원명,급여,부서번호)를 출력하는 SQL문을 작성 하세요?
(단, SQL> CONN SCOTT/TIGER 로 접속하여 SQL문을 작성하세요.)

<정답>

9 테이블 구조 생성, 변경 및 삭제하는 DDL

9.1 테이블 구조를 정의하는 CREATE TABLE

- CREATE TABLE 명령어는 새로운 테이블을 생성한다.

```
-- 형식
CREATE TABLE table_name
(column_name data_type expr, ...);

-- 예
CREATE TABLE EX2_1 (
    COLUMN1    CHAR(10),
    COLUMN2    VARCHAR2(10),
    COLUMN3    VARCHAR2(10),
    COLUMN4    NUMBER
);
```

9.1.1 데이터형

- 데이터베이스에는 문자, 숫자, 날짜, 이미지 등과 같은 다양한 형태의 데이터가 저장된다. 다음은 오라클에서 제공되는 데이터 형의 종류이다.

이름	비고
CHAR(size)	고정 길이 문자 데이터. VARCHAR2와 동일한 형태의 자료를 저장할 수 있고, 입력된 자료의 길이와는 상관없이 정해진 길이만큼 저장 영역 차지. 최소 크기는 1
VARCHAR2(size)	Up to 2000 Bytes 가변 길이 문자 데이터. 실제 입력된 문자열의 길이만큼 저장 영역을 차지. 최대 크기는 명시해야 하며, 최소 크기는 1
NUMBER	Internal Number Format 최고 40자리까지의 숫자를 저장할 수 있다. 이때 소수점이나 부호는 길이에 포함되지 않는다.
NUMBER(w)	W자리까지의 수치로 최대 38자리까지 가능하다. (38자리가 유효 숫자이다.)
NUMBER(w, d)	W는 전체 길이, d는 소수점 이하 자릿수이다. 소수점은 자릿수에 포함되지 않는다.
DATE	BC 4712년 1월 1일~AD 4712년 12월 31일까지의 날짜
LONG	가변 길이의 문자형 데이터 타입, 최대 크기는 2GB
LOB	2GB까지의 가변 길이 바이너리 데이터를 저장시킬 수 있다. 이미지 문서, 실행 파일을 저장할 수 있다.
ROWID	ROWID는 Tree-piece Format을 가짐. ROWID는 DB에 저장되어 있지 않으며, DB Data도 아니다.
BFILE	대용량의 바이너리 데이터를 파일 형태로 저장 최대 4GB
TIMESTAMP(n)	DATE 형의 확장된 형태
INTERVAL YEAR TO MONTH	년과 월을 이용하여 기간을 저장 형식: INTERVAL YEAR(년도에 대한 자릿수) TO MONTH(달에 대한 자릿수)
INTERVAL DAY TO SECOND	일, 시, 분, 초를 이용하여 기간을 저장 두 날짜 간의 정확한 차이를 표현하는데 유용하다. 형식: INTERVAL DAY(일수에 대한 자릿수) TO SECOND(초에 대한 자릿수)

9.1.2 식별자 명명 규칙

- 테이블 명과 컬럼 명과 같이 사용자가 이름을 부여하는 것을 식별자라고 하는데 식별자를 부여하기 위해서는 규칙을 준수해야 한다.
 - 반드시 문자로 시작해야 함 (1~30자까지 가능함)
 - A~Z까지의 대소문자와 0~9까지의 숫자, 특수 기호는 (_, \$, #)만 포함 가능

- 오라클에서 사용되는 예약어나 다른 객체명과 중복 불가
- 공백 허용 안함

[실습] 새롭게 테이블 생성하기

- 지금까지 실습에 사용했던 사원 테이블과 유사한 구조(사원번호, 사원명, 급여 3개의 컬럼으로 구성)의 EMP01 테이블을 생성해 보겠다.

[LAB-09-02.SQL]

```
01  -- 테이블 생성
02  DROP TABLE EMP01 CASCADE CONSTRAINTS;
03  CREATE TABLE EMP01(
04  EMPNO NUMBER(4),
05  ENAME VARCHAR2(20),
06  SAL NUMBER(7, 2));
07
08  -- 생성된 테이블의 내용을 확인한다.
09  SELECT * FROM EMP01;
10
11  -- 테이블의 구조를 DESC 명령어로 확인할 수 있다.
12  DESC EMP01
```

[실습] 서브 쿼리로 테이블 생성하기

- CREATE TABLE문에서 서브 쿼리를 사용하여 이미 존재하는 테이블과 동일한 구조와 내용을 갖는 새로운 테이블을 생성할 수 있다.

[LAB-09-03.SQL]

```
01  -- EMP 테이블과 동일한 내용과 구조를 갖는 EMP02 테이블을 생성한다.
02  -- DROP TABLE EMP02 CASCADE CONSTRAINTS;
03  CREATE TABLE EMP02
04  AS
05  SELECT * FROM EMP;
06
07  -- 새롭게 생성된 EMP02 테이블의 구조는 EMP 테이블과 동일하다.
08  DESC EMP02
```

[실습] 원하는 컬럼으로 구성된 복제 테이블 생성하기

- 기존 테이블에서 원하는 컬럼만 선택적으로 복사해서 생성할 수도 있다.

[LAB-09-04.SQL]

```
01  -- DROP TABLE EMP03 CASCADE CONSTRAINTS;
02  CREATE TABLE EMP03
03  AS
04  SELECT EMPNO, ENAME FROM EMP;
05
06  SELECT * FROM EMP03;
```

[실습] 원하는 행으로 구성된 복제 테이블 생성하기

- 기존 테이블에서 원하는 행만 선택적으로 복사해서 생성할 수도 있다.

[LAB-09-05.SQL]

```
01  -- DROP TABLE EMP05 CASCADE CONSTRAINTS;
02  CREATE TABLE EMP05
03  AS
04  SELECT * FROM EMP
05  WHERE DEPTNO=10;
06
07  SELECT * FROM EMP05;
```

9.1.3 테이블의 구조만 복사하기

- 테이블의 구조만 복사하는 것은 별도의 명령이 있는 것이 아니다. 이 역시 서브 쿼리를 이용해야 하는데 WHERE 조건 절에 항상 거짓이 되는 조건을 지정하게 되면 테이블에서 얻어질 수 있는 로우가 없게 되므로 빈 테이블이 생성되게 된다.
- WHERE 1=0; 조건은 항상 거짓이다. 이를 이용하여 테이블의 데이터는 가져오지 않고 구조만 복사하게 된다.

```
CREATE TABLE EMP06
AS
SELECT * FROM EMP WHERE 1=0;
```

9.2 테이블 구조를 변경하는 ALTER TABLE

9.2.1 새로운 컬럼 추가하기

- ALTER TABLE ADD 문은 기존 테이블에 새로운 컬럼을 추가한다.
- 새로운 컬럼은 테이블 맨 마지막에 추가되므로 자신이 원하는 위치에 만들어 넣을 수 없다.
- 이미 이전에 추가해 놓은 로우가 존재한다면 그 로우에도 컬럼이 추가되지만, 컬럼 값은 NULL 값으로 입력된다.

```
-- 형식
ALTER TABLE table_name
ADD (column_name, data_type expr, ...);
```

[실습] EMP01 테이블에 JOB 컬럼 추가하기

- EMP01 테이블에 문자 타입의 직급(JOB) 컬럼을 추가해 보겠다.

[LAB-09-06.SQL]

```
01  DESC EMP01
02
03  ALTER TABLE EMP01
04  ADD(JOB VARCHAR2(9));
```

```
05
06 DESC EMP01
```

9.2.2 기존 컬럼 속성 변경하기

- ALTER TABLE MODIFY 문을 다음과 같은 형식으로 사용하면 테이블에 이미 존재하는 컬럼을 변경할 수 있게 된다.
- 컬럼을 변경한다는 것은 컬럼에 대해서 데이터 타입이나 크기, 기본 값들을 변경한다는 의미이다.

```
-- 형식
ALTER TABLE table_name
MODIFY (column_name, data_type expr, ...);

-- 예
ALTER TABLE EMP01
MODIFY(JOB VARCHAR2(30));
DESC EMP01
```

9.2.3 기존 컬럼 삭제하기

- ALTER TABLE ~ DROP COLUMN 명령어로 컬럼을 삭제할 수 있다.

```
-- 형식
ALTER TABLE table_name
DROP COLUMN column_name;

-- 예
ALTER TABLE EMP01
DROP COLUMN JOB;
DESC EMP01
```

9.2.4 SET UNUSED 옵션 적용하기

- 특정 테이블(EMP02)에서 컬럼(JOB)을 삭제하는 경우 다음과 같이 무조건 삭제하는 것은 위험하다.
- 테이블에 저장된 내용이 많을 경우(몇 만 건에 대한 자료) 해당 테이블에서 컬럼을 삭제하는데 꽤 오랜 시간이 걸리게 될 것이다. 컬럼을 삭제하는 동안에 다른 사용자가 해당 컬럼을 사용하려고 접근하게 되면 지금 현재 테이블이 사용되고 있기 때문에 다른 사용자는 해당 테이블을 이용할 수 없게 된다. 이런 경우 작업이 원활하게 진행되지 않고 락(lock)이 발생하게 된다.
- ALTER TABLE 에 SET UNUSED 옵션을 지정하면 컬럼을 삭제하는 것은 아니지만 컬럼의 사용을 논리적으로 제한할 수 있게 된다.
- SET UNUSED 옵션은 사용을 논리적으로 제한할 뿐 실제로 컬럼을 삭제하지 않기 때문에 작업 시간이 오래 걸리지 않는다. 그렇기 때문에 락이 걸리는 일도 일어나지 않게 된다.

[실습] 직급 컬럼 사용 제한하기

SET UNUSED 옵션을 사용해 보겠습니다.

```
ALTER TABLE EMP02
SET UNUSED(JOB);

SELECT * FROM EMP02;
DESC EMP02;

ALTER TABLE EMP02
DROP UNUSED COLUMNS;
```

9.3 테이블 구조를 삭제하는 DROP TABLE

- DROP TABLE문은 기존 테이블을 제거한다.

```
-- 형식
DROP TABLE table_name;

-- 예
DROP TABLE EMP01;
DROP TABLE EMP01 CASCADE CONSTRAINTS;
SELECT * FROM EMP01;
```

9.4 테이블의 모든 로우를 제거하는 TRUNCATE

- 기존에 사용하던 테이블의 모든 로우를 제거하기 위한 명령어로 TRUNCATE가 제공된다.

```
-- 형식
TRUNCATE TABLE
table_name

-- 예
SELECT * FROM EMP02;
TRUNCATE TABLE EMP02;
SELECT * FROM EMP02;
```

9.5 테이블명을 변경하는 RENAME

- 기존에 사용하던 테이블의 이름을 변경하기 위한 명령어로 RENAME이 제공된다.

```
-- 형식
RENAME old_name TO new_name

-- 예
RENAME EMP02 TO TEST;
SELECT * FROM EMP02;
SELECT * FROM TEST;
```

9.6 데이터 디렉터리와 데이터 디렉터리 뷰

- 데이터베이스 자원을 효율적으로 관리하기 위한 다양한 정보를 저장하는 시스템 테이블을 데이터 디렉터리라고 한다.
- 데이터 디렉터리는 사용자가 테이블을 생성하거나 사용자를 변경하는 등의 작업을 할 때 데이터베이스 서버에 의해 자동으로 갱신되는 테이블로 사용자는 데이터 디렉터리의 내용을 직접 수정하거나 삭제 할 수 없다.

9.6.1 USER_ 데이터 디렉터리

- 접두어로 USER가 붙은 데이터 디렉터리는 자신의 계정이 소유한 객체 등에 관한 정보를 조회한다.

```
DESC USER_TABLES;  
  
SHOW USER  
  
SELECT TABLE_NAME FROM USER_TABLES  
ORDER BY TABLE_NAME DESC;
```

9.6.2 ALL_ 데이터 디렉터리

- 사용자 계정이 소유한 객체는 자신의 소유이므로 당연히 접근이 가능하다.
- 그러나 만일 자신의 계정이 아닌 다른 계정 소유의 테이블이나 시퀀스 등은 어떨까?
- 오라클에서는 타계정의 객체는 원천적으로 접근 불가능하다.
- 하지만 그 객체의 소유자가 접근할 수 있도록 권한을 부여하면 타 계정의 객체에도 접근이 가능하다.
- ALL_ 데이터 디렉터리 뷰는 현재 계정이 접근 가능한 객체, 즉 자신 계정의 소유이거나 접근 권한을 부여 받은 타계정의 객체 등을 조회 할 수 있는 데이터 디렉터리 뷰이다.
- 현재 계정이 접근 가능한 테이블의 정보 조회하는 뷰이다.

```
DESC ALL_TABLES;  
  
SELECT OWNER, TABLE_NAME FROM ALL_TABLES;
```

9.6.3 DBA_ 데이터 디렉터리 뷰

- DBA_ 데이터디렉터리는 DBA가 접근 가능한 객체 등을 조회 할 수 있는 뷰이다.
- 앞서도 언급했지만 DBA가 접근 불가능한 정보는 없기에 데이터베이스에 있는 모든 객체 등의 의미라 할 수 있다.
- USER_ 와 ALL_ 와 달리 DBA_ 데이터디렉터리뷰는 DBA 시스템 권한을 가진 사용자만 접근할 수 있다.

```
SELECT TABLE_NAME, OWNER  
FROM DBA_TABLES;
```



```
CONN system/manager
SELECT TABLE_NAME, OWNER
FROM DBA_TABLES;
```

10 테이블의 내용 추가, 수정, 삭제하는 DML

10.1 테이블에 새로운 행을 추가하는 INSERT문

- INSERT 문은 테이블에 새로운 데이터를 입력하기 위해 사용하는 데이터 조작어이다.

```
-- 형식
INSERT INTO table_name
(column_name, ...)
VALUES(column_value, ...);

-- 예
INSERT INTO DEPT01
VALUES(10, 'ACCOUNTING', 'NEW YORK');

INSERT INTO DEPT01
(DEPTNO, DNAME)
VALUES(10, 'ACCOUNTING');
```

[실습] INSERT문 실습에 사용할 테이블 생성하기

- INSERT문을 위한 실습을 하기에 앞서 실습에 사용할 테이블을 CREATE TABLE 명령어로 새롭게 만들어 보겠다.

```
-- 기존에 있던 부서 테이블(DEPT)과 동일한 구조를 갖되 데이터는 복사하지 않는 테이블을 생성한다.
DROP TABLE DEPT01;
CREATE TABLE DEPT01
AS
SELECT * FROM DEPT WHERE 1=0;

-- 생성된 테이블의 구조를 살펴본다.
DESC DEPT01

-- 수록된 데이터를 살펴본다.
SELECT * FROM DEPT01;
```

10.1.1 INSERT 구문의 오류 발생 예

- 칼럼 명에 기술된 목록의 수보다 VALUES 다음에 나오는 괄호 안에 기술한 값의 개수가 적으면 에러가 발생한다.
- 칼럼 명에 기술된 목록의 수보다 VALUES 다음에 나오는 괄호에 기술한 값의 개수가 많으면 에러가 발생한다.
- 칼럼 명이 잘못 입력되었을 때에도 에러가 발생한다.
- 칼럼과 입력할 값의 데이터 타입이 서로 맞지 않을 경우에도 에러가 발생한다.

```
INSERT INTO DEPT01
(DEPTNO, DNAME, LOC)
VALUES (10, 'ACCOUNTING');

INSERT INTO DEPT01
(DEPTNO, DNAME, LOC)
```

```
VALUES(10, 'ACCOUNTING', 'NEW YORK', 20);
```

```
INSERT INTO DEPT01  
(NUM, DNAME, LOC)  
VALUES(10, 'ACCOUNTING', 'NEW YORK');
```

```
INSERT INTO DEPT01  
(DEPTNO, DNAME, LOC)  
VALUES(10, 'ACCOUNTING', 'NEW YORK');
```

10.1.2 컬러명을 생략한 INSERT 구문

- 테이블에 로우를 추가할 때 모든 컬럼에 모두 자료를 입력하는 경우에는 굳이 컬럼 목록을 기술하지 않아도 된다.
- 컬럼 목록을 생략하면 VALUES 절 다음의 값들을 테이블의 기본 컬럼 순서대로 입력한다.
- 테이블의 컬럼 순서는 CREATE TABLE로 테이블을 생성할 때의 순서를 따른다.
- 테이블의 기본 컬럼 순서는 DESC 문으로 조회했을 때 보여 지는 순서이다.

```
INSERT INTO DEPT01  
VALUES (20, 'RESEARCH', 'DALLAS');
```

```
SELECT * FROM DEPT01;
```

10.1.3 NULL 값을 삽입하는 다양한 방법

- 부서 테이블에 컬럼이 NULL값을 허용하는지 살펴보기 위해서 DESC 명령을 실행한다.
- DEPT 테이블의 DEPTNO 컬럼은 NOT NULL 제약조건이 지정되어 있다.
- NOT NULL 제약조건이 지정된 DEPTNO 컬럼은 널 값을 입력하지 못한다.
- 오라클이 제공해 주는 DEPT 테이블의 DEPTNO 컬럼에 널값을 허용하지 못하도록 오라클 내부에서 이미 컬럼에 제약조건을 지정해 놓은 상태이다.
- 컬럼에 널값을 허용하지 못하도록 하려면 컬럼에 제약조건을 지정해야 한다.

(1) 암시적으로 NULL 값의 삽입

- 저장할 값을 명확하게 알고 있는 컬럼 명만 명시적으로 기술한 후에 그에 매칭되는 값을 VALUES 절 다음에 기술한다.

```
INSERT INTO DEPT01  
(DEPTNO, DNAME)  
VALUES (30, 'SALES');
```

(2) 명시적으로 NULL 값의 삽입

- 컬럼명을 명시적으로 기술하지 않으면 테이블이 갖고 있는 모든 컬럼에 값을 지정해야 한다.

```
INSERT INTO DEPT01  
VALUES (40, 'OPERATIONS', NULL);
```

10.1.4 서브 쿼리로 데이터 삽입하기

- INSERT INTO 다음에 VALUES 절을 사용하는 대신에 서브 쿼리를 사용할 수 있다.
- 주의할 점은 INSERT 명령문에서 지정한 컬럼의 개수나 데이터 타입이 서브 쿼리를 수행한 결과와 동일해야 한다.

[실습] 서브 쿼리로 데이터를 삽입하는 예제

- INSERT INTO 다음에 VALUES 절을 사용하는 대신 서브 쿼리를 사용하여 데이터를 추가한다.

```
DROP TABLE DEPT02;  
CREATE TABLE DEPT02  
AS  
SELECT * FROM DEPT WHERE 1=0;  
  
SELECT * FROM DEPT02;  
  
INSERT INTO DEPT02  
SELECT * FROM DEPT;  
  
SELECT * FROM DEPT02;
```

10.2 다중 테이블에 다중행 입력하기

- INSERT ALL 명령문은 서브 쿼리의 결과 집합을 조건 없이 여러 테이블에 동시에 입력하기 위한 명령문입니다.

[실습] INSERT ALL 명령문으로 다중 테이블에 다중행 입력하기

```
DESC EMP  
SELECT * FROM EMP  
WHERE DEPTNO=20;  
  
DROP TABLE EMP_HIR;  
CREATE TABLE EMP_HIR  
AS  
SELECT EMPNO, ENAME, HIREDATE FROM EMP  
WHERE 1=0;  
SELECT * FROM EMP_HIR;  
  
DROP TABLE EMP_MGR;  
CREATE TABLE EMP_MGR  
AS  
SELECT EMPNO, ENAME, MGR FROM EMP  
WHERE 1=0;  
SELECT * FROM EMP_MGR;  
  
INSERT ALL  
INTO EMP_HIR VALUES(EMPNO, ENAME, HIREDATE)  
INTO EMP_MGR VALUES(EMPNO, ENAME, MGR)
```

```

SELECT EMPNO, ENAME, HIREDATE, MGR
FROM EMP
WHERE DEPTNO=20;

SELECT * FROM EMP_HIR;
SELECT * FROM EMP_MGR;

```

[실습] INSERT ALL 명령에 조건(WHEN)으로 다중 테이블에 다중행 입력하기

```

DROP TABLE EMP_HIR02;
CREATE TABLE EMP_HIR02
AS
SELECT EMPNO, ENAME, HIREDATE FROM EMP
WHERE 1=0;
SELECT * FROM EMP_HIR02;

DROP TABLE EMP_SAL;
CREATE TABLE EMP_SAL
AS
SELECT EMPNO, ENAME, SAL FROM EMP
WHERE 1=0;
SELECT * FROM EMP_SAL;

INSERT ALL
WHEN HIREDATE > '1982/01/01' THEN
INTO EMP_HIR02 VALUES(EMPNO, ENAME, HIREDATE)
WHEN SAL > 2000 THEN
INTO EMP_SAL VALUES(EMPNO, ENAME, SAL)
SELECT EMPNO, ENAME, HIREDATE, SAL FROM EMP;

SELECT * FROM EMP_HIR02;
SELECT * FROM EMP_SAL;

```

10.3 테이블의 내용을 수정하기 위한 UPDATE문

- UPDATE 문은 테이블에 저장된 데이터를 수정하기 위해서 사용한다.

```

-- 형식
UPDATE table_name
SET column_name1 = value1, column_name2 = value2, ...
WHERE conditions;

```

10.3.1 테이블의 모든 행 변경

```

DROP TABLE EMP01;
DROP TABLE EMP01 CASCADE CONSTRAINTS;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

UPDATE EMP01

```

```

SET DEPTNO=30;
SELECT * FROM EMP01;

UPDATE EMP01
SET SAL = SAL * 1.1;
SELECT * FROM EMP01;

UPDATE EMP01
SET HIREDATE = SYSDATE;
SELECT * FROM EMP01;

```

10.3.2 테이블의 특정 행만 변경

- UPDATE 문에 WHERE 절을 추가하면 테이블의 특정 행이 변경된다.

```

DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

UPDATE EMP01
SET DEPTNO=30
WHERE DEPTNO=10;
SELECT * FROM EMP01;

UPDATE EMP01
SET SAL = SAL * 1.1
WHERE SAL >= 3000;
SELECT * FROM EMP01;

UPDATE EMP01
SET HIREDATE = SYSDATE
WHERE SUBSTR(HIREDATE, 1, 2)='87';
SELECT * FROM EMP01;

```

10.3.3 테이블에서 2개 이상의 컬럼값 변경

- 테이블에서 하나의 컬럼이 아닌 복수 개 컬럼의 값을 변경하려면 기존 SET 절에 콤마를 추가하고 컬럼=값을 추가 기술하면 된다.

```

DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

UPDATE EMP01
SET DEPTNO=20, JOB='MANAGER'
WHERE ENAME='SCOTT';
SELECT * FROM EMP01 WHERE ENAME='SCOTT';

UPDATE EMP01
SET HIREDATE = SYSDATE, SAL=50, COMM=4000
WHERE ENAME='SCOTT';
SELECT * FROM EMP01 WHERE ENAME='SCOTT';

```

10.3.4 서브 쿼리를 이용한 데이터 수정하기

- UPDATE 문의 SET 절에서 서브 쿼리를 기술하면 서브 쿼리를 수행한 결과로 내용이 변경된다.
- 이러한 방법으로 다른 테이블에 저장된 데이터로 해당 컬럼 값을 변경할 수 있다.

```
-- 형식 1
UPDATE table_name
SET column_name1 = (sub_query1), column_name2 = (sub_query2), ...
WHERE 조건

-- 형식 2
UPDATE table_name
SET (column_name1, column_name2, ...) = (sub_query)
WHERE 조건

-- 예
UPDATE DEPT01
SET LOC=(SELECT LOC
          FROM DEPT01
          WHERE DEPTNO=20);
```

10.4 테이블의 불필요한 행을 삭제하기 위한 DELETE문

10.4.1 DELETE문을 이용한 행 삭제

- DELETE 문은 테이블의 기존 행을 삭제하며 특정한 로우(행)을 삭제하기 위해서 WHERE 절을 이용하여 조건을 지정한다.

```
-- 형식
DELETE FROM table_name
WHERE conditions;

-- 예
DELETE FROM DEPT01;
```

10.4.2 조건을 제시하여 특정 행만 삭제하는 법

```
DELETE FROM DEPT01
WHERE DEPTNO=30;
```

10.4.3 서브 쿼리를 이용한 데이터 삭제

```
DELETE FROM DEPT01
WHERE DEPTNO=(SELECT DEPTNO
              FROM DEPT
```

```
WHERE DNAME='SALES');
```

10.5 테이블을 합병하는 MERGE

- MERGE는 합병이란 의미이므로 구조가 같은 두개의 테이블을 하나의 테이블로 합치는 기능을 한다.
- MERGE 명령을 수행하기 위해서 수행하는 테이블에 기존에 존재하는 행이 있다면 새로운 값으로 갱신(UPDATE)되고, 존재하지 않으면 새로운 행으로 추가(INSERT)된다.

```
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP
WHERE JOB='MANAGER';
SELECT * FROM EMP02;

UPDATE EMP02
SET JOB='TEST';
SELECT * FROM EMP02;

INSERT INTO EMP02
VALUES(8000, 'SYJ', 'TOP', 7566, '2009/01/12', 1200, 10, 20);

MERGE INTO EMP01
USING EMP02
ON(EMP01.EMPNO=EMP02.EMPNO)
WHEN MATCHED THEN
UPDATE SET
EMP01.ENAME=EMP02.ENAME,
EMP01.JOB=EMP02.JOB,
EMP01.MGR=EMP02.MGR,
EMP01.HIREDATE=EMP02.HIREDATE,
EMP01.SAL=EMP02.SAL,
EMP01.COMM=EMP02.COMM,
EMP01.DEPTNO=EMP02.DEPTNO
WHEN NOT MATCHED THEN
INSERT VALUES(EMP02.EMPNO, EMP02.ENAME, EMP02.JOB,
EMP02.MGR, EMP02.HIREDATE, EMP02.SAL,
EMP02.COMM, EMP02.DEPTNO);

SELECT * FROM EMP01;
```

[과제] 과제-10-01.TXT

SQL> CONN ORA_USER/HONG 로 접속하여 SQL문을 작성하세요.

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

<정답>

2. 다음 문장을 실행해보자.

```
CREATE TABLE EX3_3 (  
    EMPLOYEE_ID NUMBER,  
    BONUS_AMT    NUMBER DEFAULT 0);  
INSERT INTO EX3_3 VALUES(148,0);  
INSERT INTO EX3_3 VALUES(153,0);  
INSERT INTO EX3_3 VALUES(154,0);  
INSERT INTO EX3_3 VALUES(155,0);  
INSERT INTO EX3_3 VALUES(161,0);  
COMMIT;
```

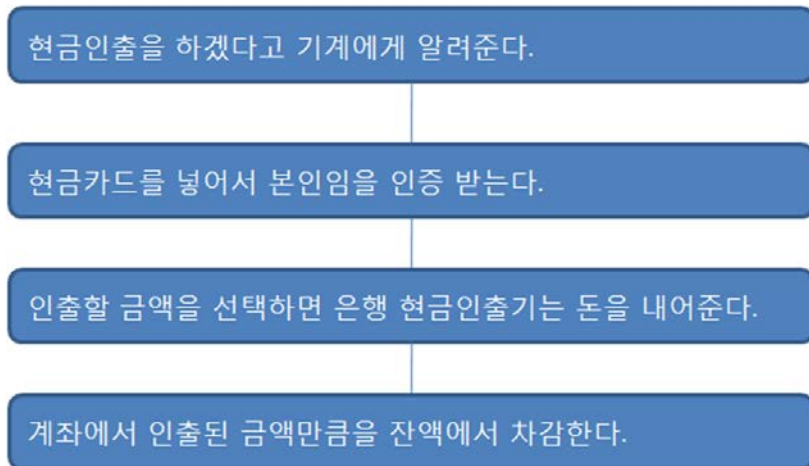
관리자사번(manager_id)이 145번인 사원을 찾아 위 테이블에 있는 사원의 사번과 일치하면 보너스 금액(bonus_amt)에 자신의 급여의 1%를 보너스로 갱신하고, 사원의 사번과 일치하지 않는 사원을 EX3_3 테이블에 신규 입력 (이때 보너스 금액은 급여의 0.5%로 한다) 하는 MERGE 문을 작성하라.

<정답>

11. 트랜잭션 관리

11.1. 트랜잭션

- 오라클에서 발생하는 여러 개의 SQL 명령문들을 하나의 논리적인 작업 단위로 처리하는데 이를 트랜잭션이라고 한다.
- 하나의 트랜잭션은 All-OR-Nothing 방식으로 처리된다. 즉, 여러 개의 명령어의 집합이 정상적으로 처리되면 정상 종료하도록 하고 여러 개의 명령어 중에서 하나의 명령어라도 잘못되었다면 전체를 취소해버린다.
- 데이터베이스에서 작업의 단위로 트랜잭션이란 개념을 도입한 이유는 데이터의 일관성을 유지하면서 안정적으로 데이터를 복구시키기 위해서이다.
- 예를들어 은행 현금인출기(ATM)에서 돈을 인출하는 과정으로 트랜잭션을 설명해 보면..

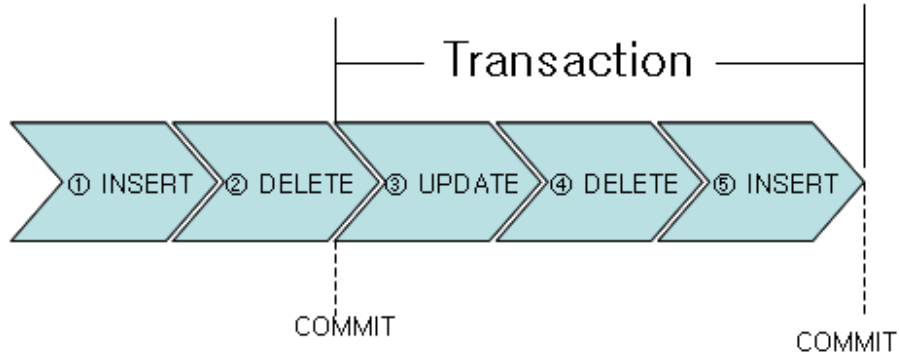


- 트랜잭션 제어를 위한 명령어(Transaction Control Language) : COMMIT, SAVEPOINT, ROLLBACK

11.2. COMMIT과 ROLLBACK

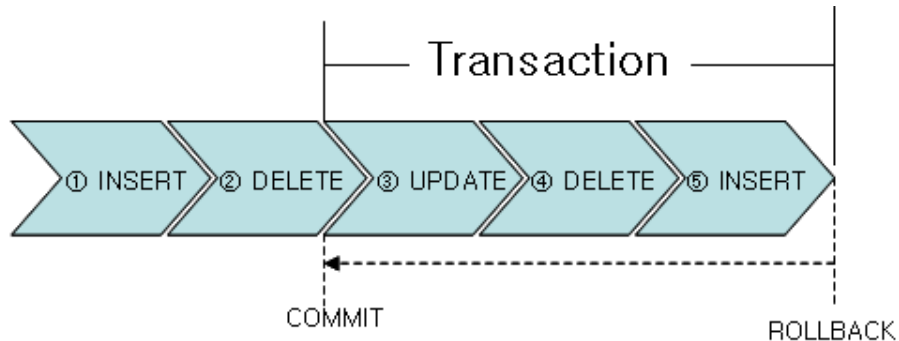
11.2.1. COMMIT과 ROLLBACK의 개념

- DML 작업이 성공적으로 처리되도록 하기 위해서는 COMMIT 명령을, 작업을 취소하기 위해서는 ROLLBACK 명령으로 종료해야 한다.
- COMMIT
 - COMMIT은 모든 작업들을 정상적으로 처리하겠다고 확정하는 명령어로 트랜잭션의 처리 과정을 데이터베이스에 모두 반영하기 위해서 변경된 내용을 모두 영구 저장한다.
 - COMMIT 명령어를 수행하게 되면 하나의 트랜잭션 과정을 종료하게 된다.



■ ROLLBACK

- ROLLBACK은 작업 중 문제가 발생되어서 트랜잭션의 처리 과정에서 발생한 변경사항을 취소하는 명령어이다. 즉, 트랜잭션으로 인한 하나의 묶음 처리가 시작되기 이전의 상태로 되돌린다.
- ROLLBACK 명령어 역시 트랜잭션 과정을 종료하게 된다.



(1) COMMIT 명령어와 ROLLBACK 명령어의 장점

- 데이터 무결성이 보장된다.
- 영구적인 변경 전에 데이터의 변경 사항을 확인할 수 있다.
- 논리적으로 연관된 작업을 그룹화할 수 있다.

(2) COMMIT 명령어

- Transaction(INSERT, UPDATE, DELETE) 작업 내용을 실제 DB에 저장한다.
- 이전 데이터가 완전히 UPDATE 된다.
- 모든 사용자가 변경된 데이터의 결과를 볼 수 있다.

(3) ROLLBACK 명령어

- Transaction 작업 내용을 취소한다.
- 이전 COMMIT한 곳 까지만 복구한다.

(4) 자동 COMMIT 명령과 자동 ROLLBACK 명령이 되는 경우

- SQL* PLUS가 정상 종료되었다면 자동으로 COMMIT되지만, 비정상 종료되었다면 자동으로 ROLLBACK 한다.
- DDL과 DCL(Data Control Language, User 관리) 명령문이 수행된 경우 자동으로 COMMIT 된다.
- 정전이 발생했거나 컴퓨터 Down시(컴퓨터의 전원이 끊긴) 자동으로 ROLLBACK 된다.

11.3. 자동 커밋

- DDL문에는 CREATE, ALTER, DROP, RENAME, TRUNCATE 등이 있다. 이러한 DDL문은 자동으로 커밋(AUTO COMMIT)을 실행한다.

[실습] CREATE문에 의한 자동 커밋

- 이전에 수행했던 DML 명령어가 CREATE문에 의해 자동 커밋되는지 확인해 본다.

```
-- CREATE문에 의한 자동 커밋
DROP TABLE DEPT02;
CREATE TABLE DEPT02
AS
SELECT * FROM DEPT;
SELECT * FROM DEPT02;

DELETE FROM DEPT02
WHERE DEPTNO=40;

CREATE TABLE DEPT03
AS
SELECT * FROM DEPT;

ROLLBACK; -- ROLLBACK 명령을 수행하였지만, DELETE된 정보가 되살아나지 않는 이유는 CREATE 명령문이 수행되면서
자동으로 커밋이 발생되었기 때문이다.

SELECT * FROM DEPT02;
```

[실습] TRUNCATE문의 실패에 의한 자동 커밋

- TRUNCATE문이 실패했더라도 이전에 수행했던 DML 명령어가 자동으로 커밋되는 것을 확인해 본다.

```
DROP TABLE DEPT03;
CREATE TABLE DEPT03
AS
SELECT * FROM DEPT;

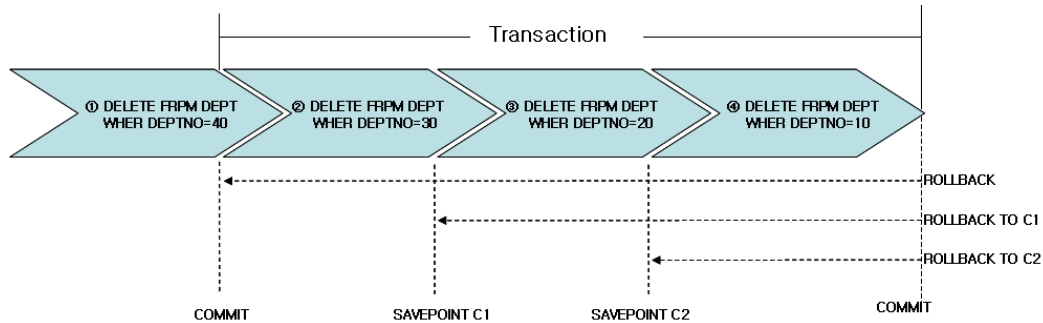
SELECT * FROM DEPT03;
DELETE FROM DEPT03
WHERE DEPTNO=20;

TRUNCATE TABLE DEPTPPP;
ROLLBACK;

SELECT * FROM DEPT03;
```

11.4. 트랜잭션을 작게 분할하는 SAVEPOINT

- SAVEPOINT 명령을 써서 현재의 트랜잭션을 작게 분할할 수 있다.
- 저장된 SAVEPOINT는 ROLLBACK TO SAVEPOINT 문을 사용하여 표시한 곳까지 ROLLBACK할 수 있다.
- 여러 개의 SQL 문의 실행을 수반하는 트랜잭션의 경우, 사용자가 트랜잭션 중간 단계에서 세이브포인트를 지정할 수 있다.
- 이 세이브포인트는 차후 롤백과 함께 사용해서 현재 트랜잭션 내의 특정 세이브포인트까지 롤백할 수 있게 됩니다.



```
-- 형식
SAVEPOINT LABEL_NAME; -- 특정 위치 지정
ROLLBACK TO LABEL_NAME; -- 특정 위치로 되돌아가기

-- 예 : 트랜잭션 중간 단계에서 세이브포인트 지정하기
DROP TABLE DEPT01;
CREATE TABLE DEPT01
AS
SELECT * FROM DEPT;

DELETE FROM DEPT01 WHERE DEPTNO=40;
COMMIT;
SELECT * FROM DEPT01;

DELETE FROM DEPT01
WHERE DEPTNO=30;

SAVEPOINT C1;
DELETE FROM DEPT01
WHERE DEPTNO=20;
SELECT * FROM DEPT01;

SAVEPOINT C2;
DELETE FROM DEPT01
WHERE DEPTNO=10;
SELECT * FROM DEPT01;

ROLLBACK TO C2;
SELECT * FROM DEPT01;

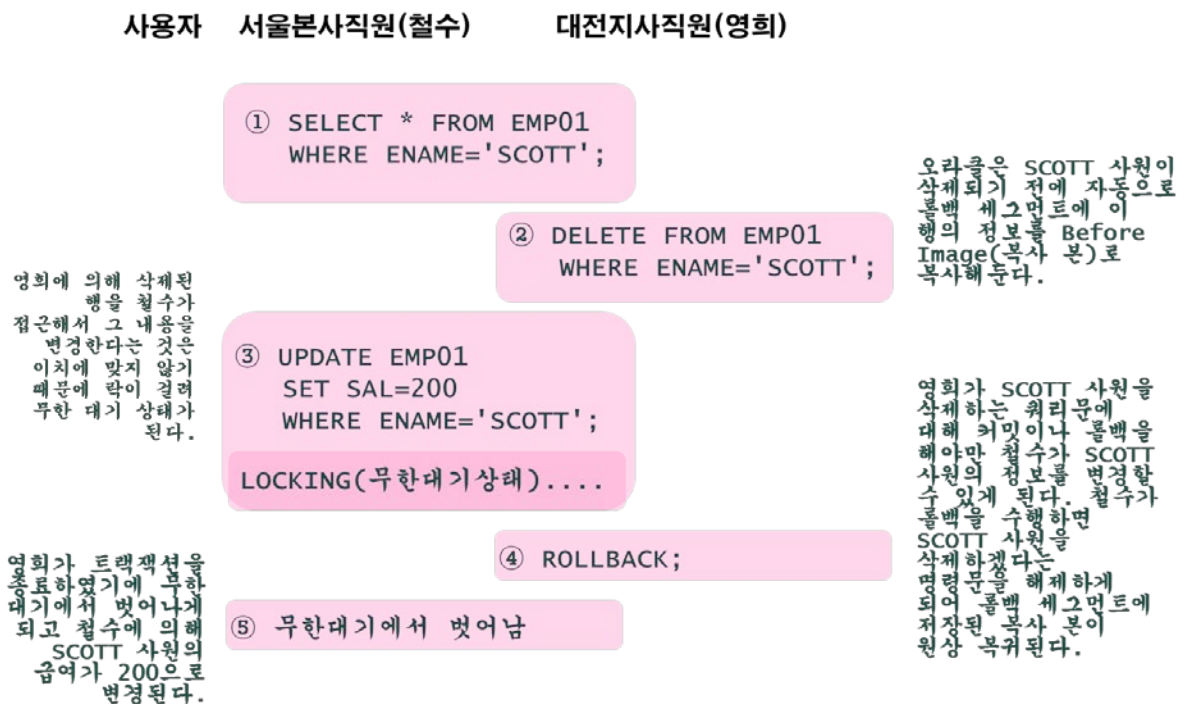
ROLLBACK TO C1;
SELECT * FROM DEPT01;

ROLLBACK;
SELECT * FROM DEPT01;
```

12. 데이터 읽기 일관성과 락

12.1. 데이터 읽기의 일관성과 락

- 오라클이 데이터 읽기의 일관성을 제공해 준다는 것을 증명을 하기 위해서 우선 다음과 같은 가정을 하겠다.
- 오라클 서버가 서울 본사에 설치되어 있고 이 데이터베이스를 서울 본사 직원과 대전 지사 직원이 공유하고 있다고 하자.
- 어느 날 서울 본사 직원과 대전 지사 직원이 동일한 테이블을 같은 시간에 접근해서 사용할 경우 어떻게 일이 일어날 수 있는지 살펴보도록 하자.



[실습] 데이터 읽기의 일관성과 락

```
-- 1. 도스창(A)을 띄운 후 SCOTT 계정으로 접속한 뒤, EMP 테이블과 구조와 내용이 동일한 EMP01 테이블을 만든다.
SET LINESIZE 100
SET PAGESIZE 30
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

-- 2. SCOTT 사원이 존재하는지 확인한다.
SELECT * FROM EMP01
WHERE ENAME='SCOTT';

-- 3. 또 다른 도스창(B)을 띄운 후 SCOTT 계정으로 접속한다. 도스창이 2개 뜬 상태가 된다. 나중에 띄운 도스창에서 SCOTT 사원을 삭제한다.
DELETE FROM EMP01
WHERE ENAME='SCOTT';
```

```
-- 4. 먼저 띄운 도스창(A)에서 SCOTT 사원의 정보를 변경한다. 도스창(A)에 SQL 프롬프트가 나타나지 못하여 무한 대기 상태로 빠지게 된다.
UPDATE EMP01
SET SAL=200
WHERE ENAME='SCOTT';

-- 5. 도스창(B)는 대기 상태가 아니기에 롤백을 수행한다.
ROLLBACK;

-- 6. 도스창(A)의 UPDATE 작업이 성공적으로 수행된다. 명령어도 완벽히 끝내려면 커밋이나 롤백을 수행해야 한다.
ROLLBACK;
```

12.2. 데드 락

- 데드락이란 한마디로 무한 교착상태이다. 하나의 상자에 들어 있는 물건을 서로 가지려고 두 사람이 자기만의 자물쇠로 상자를 잠궜 버린 경우라고 할 수 있다. 이렇게 되면 서로 상자의 물건을 어느 누구도 가질 수 없게 된다.

철수(사용자: 서울본사직원)	영희(사용자: 대전지사직원)
1. UPDATE EMP01 SET SAL=100 WHERE ENAME='SCOTT';	2. UPDATE EMP01 SET SAL=20 WHERE ENAME='SMITH';
3. UPDATE EMP01 SET SAL=300 WHERE ENAME='SMITH';	4. UPDATE EMP01 SET SAL=400 WHERE ENAME='SCOTT';
LOCKING(무한 대기 상태) ...	LOCKING(무한 대기 상태) ...

[실습] 데드 락

```
-- 1. 왼쪽 사용자가 SCOTT 사원의 정보를 변경하려고 시도한다.
UPDATE EMP01 SET SAL=100
WHERE ENAME='SCOTT';

-- 2. 오른쪽 사용자가 SMITH 사원의 정보를 변경하려고 시도한다.
UPDATE EMP01 SET SAL=20
WHERE ENAME='SMITH';

-- 3. 왼쪽 사용자가 SMITH 사원의 정보를 갱신하려고 시도한다.
UPDATE EMP01 SET SAL=300
WHERE ENAME='SMITH';

-- 4. 왼쪽 사용자가 대기 상태인데 오른쪽 사용자가 커밋이나 롤백을 하지 않고 왼쪽 사용자가 사용 중인 SCOTT의 정보를 변경하려고 시도한다.
UPDATE EMP01 SET SAL=400
WHERE ENAME='SCOTT';

-- 5. 오른쪽 사용자가 ROLLBACK을 입력하여 트랜잭션을 마무리 짓고,
```

12.3. SET UNUSED

- DDL 작업을 하는 동안에도 락이 발생하게 되는데, 이럴경우에 SET UNUSED라는 키워드를 사용하여 해결한다.
- SET UNUSED는 해당 컬럼을 실제로 삭제하는 것이 아니라 삭제된다는 표시만 하는 논리적 삭제만 일어나게 하므로 실제 수행 시간은 극히 짧다. 따라서 락이 걸리는 시간도 짧으므로 다른 사용자의 사용 제한 시간이 상대적으로 짧다.

```
DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP;

ALTER TABLE EMP02
SET UNUSED (JOB);
SELECT * FROM EMP02; -- 실제로 JOB 컬럼이 존재하지만 논리적으로 사용을 제한하기 위해서 UNUSED 속성을 지정함.
```

12.4. DDL 명령의 롤백

- DDL은 자동 커밋이 일어나므로 이전 상태로 되돌리기 위해서 롤백할 수 없다.
- DDL 작업을 수행하기 전에 원본 테이블을 복사해 놓고 롤백이 필요할 경우에 복사본을 원본 테이블로 대체한다.

```
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP; -- 원본 테이블(EMP01)

DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP01; -- DDL 작업을 수행하기 전에 원본 테이블의 복사본(EMP02)을 생성한다.

ALTER TABLE EMP01
DROP COLUMN JOB; -- 원본 테이블에서 컬럼을 삭제한다.

SELECT * FROM EMP01;
SELECT * FROM EMP02;

DROP TABLE EMP01;
RENAME EMP02 TO EMP01; -- 원본 테이블을 삭제하고 복사본을 원본 테이블명으로 변경한다. (롤백과 같은 효과)

SELECT * FROM EMP01;
```

12.5. TRUNCATE와 DELETE의 차이

- TRUNCATE
 - 모든 행을 삭제한다.
 - DDL 명령어로 자동으로 커밋이 발생한다.
- DELETE
 - WHERE 절을 추가하여 조건에 만족하는 행만 삭제할 수 있다.
 - DML 명령어로 롤백이 가능하다는 장점이 있지만 롤백을 위해서 무수히 많은 BEFORE IMAGE가 생성되어야 하므로 삭제하는 속도도 늦고 이를 위한 자원이 마련되어야 한다.


```
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

TRUNCATE TABLE EMP01; -- DDL 명령어로서 자동 커밋이 발생하기 때문에 이전으로 되돌릴 수 없다.
SELECT * FROM EMP01;
ROLLBACK;
SELECT * FROM EMP01;

DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP;
SELECT * FROM EMP02;

DELETE FROM EMP02; -- DML 명령어이기 때문에 롤백을 수행하면 삭제 이전으로 되돌릴 수 있다.
SELECT * FROM EMP02;
ROLLBACK;
SELECT * FROM EMP02;

DROP TABLE EMP03;
CREATE TABLE EMP03
AS
SELECT * FROM EMP;
SELECT * FROM EMP03;

DELETE FROM EMP03
WHERE DEPTNO>=10 OR DEPTNO=20;
SELECT * FROM EMP03;
COMMIT; -- 커밋 후에는 트랜잭션이 종료된 것이므로 롤백 이미지가 제거된다.
ROLLBACK; -- 따라서 삭제 이전 상태로 돌아갈 수 없다.
SELECT * FROM EMP03;
```

13. 데이터 무결성을 위한 제약 조건

13.1. 무결성 제약 조건의 개념과 종류

13.1.1. 데이터 무결성 제약 조건이란?

- 데이터 무결성 제약 조건(Data Integrity Constraint Rule)이란 테이블에 부적절한 자료가 입력되는 것을 방지하기 위해서 테이블을 생성할 때 각 컬럼에 대해서 정의하는 여러 가지 규칙을 말한다.

13.1.2. 무결한 데이터의 5가지 제약 조건

무결성 제약 조건	역할
NOT NULL	NULL을 허용하지 않는다.
UNIQUE	중복된 값을 허용하지 않는다. 항상 유일한 값을 갖도록 한다.
PRIMARY KEY	NULL을 허용하지 않고 중복된 값을 허용하지 않는다. NOT NULL 조건과 UNIQUE 조건을 결합한 형태이다.
FOREIGN KEY	참조되는 테이블의 컬럼의 값이 존재하면 허용한다.
CHECK	저장 가능한 데이터 값의 범위나 조건을 지정하여 설정한 값만을 허용한다.

13.1.3. 제약 조건 확인하기

- CONSTRAINT_NAME은 제약 조건 명이다.
- CONSTRAINT_TYPE는 제약 조건 유형을 저장하는 컬럼이다.

CONSTRAINT_TYPE	의미
P	PRIMARY KEY
R	FOREIGN KEY
U	UNIQUE
C	CHECK, NOT NULL

```
INSERT INTO DEPT
VALUES(10, 'TEST', 'SEOUL');
-- 오류발생: 무결성 제약 조건에 위배됩니다.
```

```
DESC DEPT
SELECT * FROM DEPT;
```

```
DESC USER_CONSTRAINTS;
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM USER_CONSTRAINTS
WHERE TABLE_NAME='DEPT';
```

13.1.4. 제약 조건 살피기

```
COLUMN CONSTRAINT_TYPE FORMAT A18
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM USER_CONSTRAINTS;
```

```
COLUMN OWNER FORMAT A10
COLUMN TABLE_NAME FORMAT A15
```

```
COLUMN COLUMN_NAME FORMAT A15
SELECT *
FROM USER_CONS_COLUMNS;
```

13.2. 필수 입력을 위한 NOT NULL 제약 조건

- NOT NULL 제한 조건은 해당 컬럼에 데이터를 추가하거나 수정할 때 NULL 값이 저장되지 않게 제약을 걸어주는 것으로서 사원번호와 사원명과 같이 자료가 꼭 입력되게 하고 싶을 때 사용한다.

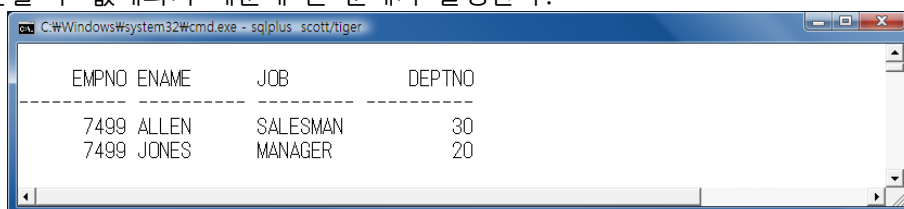
```
-- NOT NULL 제약 조건을 설정하지 않고 테이블 생성하기
DROP TABLE EMP01;
CREATE TABLE EMP01(
EMPNO NUMBER(4),
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
DEPTNO NUMBER(2)
);
SELECT * FROM EMP01;
INSERT INTO EMP01
VALUES(NULL, NULL, 'SALESMAN', 30);

-- NOT NULL 제약 조건을 설정하여 테이블 생성하기
DROP TABLE EMP02;
CREATE TABLE EMP02(
EMPNO NUMBER(4) NOT NULL,
ENAME VARCHAR2(10) NOT NULL,
JOB VARCHAR2(9),
DEPTNO NUMBER(2)
);
INSERT INTO EMP02
VALUES(NULL, NULL, 'SALESMAN', 30); -- 오류발생: cannot insert NULL into ("SCOTT"."EMP02"."EMPNO")

DESC EMP02
INSERT INTO EMP02
VALUES(7499, 'ALLEN', 'SALESMAN', 30);
SELECT * FROM EMP02;
```

13.3. 유일한 값만 허용하는 UNIQUE 제약 조건

- UNIQUE 제약 조건이란 특정 컬럼에 대해 자료가 중복되지 않게 하는 것이다. 즉, 지정된 컬럼에는 유일한 값이 수록되게 하는 것이다.
- 새로운 사원이 입사하여 이 사원의 정보를 입력했는데, 이미 존재하는 사원의 번호와 동일한 사원번호로 입력하였더니 성공적으로 추가된다면 어떻게 될까? -> 사원 번호로는 사원을 구분할 수 없게되기 때문에 큰 문제가 발생한다.



EMPNO	ENAME	JOB	DEPTNO
7499	ALLEN	SALESMAN	30
7499	JONES	MANAGER	20

```
-- UNIQUE 제약 조건을 설정하여 테이블 생성하기
DROP TABLE EMP03;
CREATE TABLE EMP03(
EMPNO NUMBER(4) UNIQUE,
ENAME VARCHAR2(10) NOT NULL,
JOB VARCHAR2(9),
DEPTNO NUMBER(2)
);

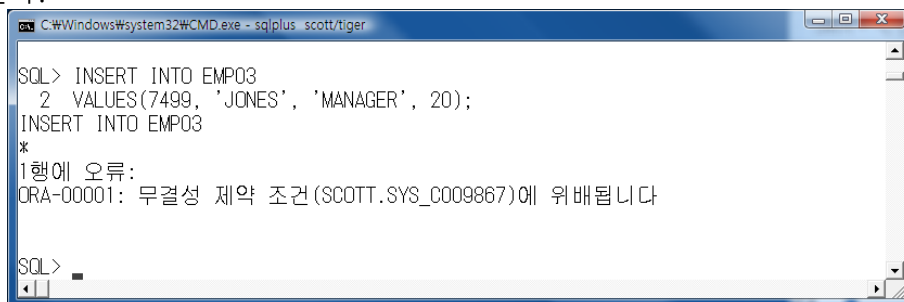
INSERT INTO EMP03
VALUES(7499, 'ALLEN', 'SALESMAN', 30);
SELECT * FROM EMP03;

INSERT INTO EMP03
VALUES(7499, 'JONES', 'MANAGER', 20);
INSERT INTO EMP03
VALUES(NULL, 'JONES', 'MANAGER', 20); -- 오류발생: unique constraint (SCOTT.SYS_C0011406) violated

INSERT INTO EMP03
VALUES(NULL, 'JONES', 'SALESMAN', 10); -- UNIQUE는 NULL값은 예외로 간주한다.
SELECT * FROM EMP03; -- 만약 NULL값도 입력되지 않게 제한하려면 UNIQUE NOT NULL처럼 기술하면 된다.
```

13.4. 컬럼 레벨로 제약 조건명을 명시하여 제약 조건 설정하기

- 사용자가 제약조건을 주면 오라클은 SYS_ 다음에 숫자를 나열하여 제약 조건 명을 자동 부여한다.



- 사용자가 직접 제약 조건 명을 설정하려면 CONSTRAINT라는 키워드를 사용한다.

```
-- 형식
column_name data_type CONSTRAINT constraint_name constraint_type

-- 예: 컬럼 레벨로 제약 조건명 명시하기
DROP TABLE EMP04;
CREATE TABLE EMP04(
EMPNO NUMBER(4) CONSTRAINT EMP04_EMPNO_UK UNIQUE,
ENAME VARCHAR2(10));

SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM USER_CONSTRAINTS
WHERE TABLE_NAME='EMP04';
```

13.5. 데이터 구분을 위한 PRIMARY KEY 제약 조건

- UNIQUE 제약 조건과 NOT NULL 제약 조건을 모두 갖는 것이 기본 키(PRIMARY KEY) 제약 조건

이다.

- UNIQUE 제약 조건을 지정한 칼럼은 중복된 데이터를 저장하지는 못하지만 NULL 값을 저장하는 것은 허용한다. 이와 같이 동명이인이 입사를 했다면 이를 구분할 수 있는 유일한 키가 있어야 하는데 사원번호에 NULL 값이 저장되는 바람에 이들을 구분할 수 없게 된다.

```
C:\Windows\system32\CMD.exe - sqlplus scott/tiger

SQL> INSERT INTO EMP03
  2  VALUES(NULL, 'JONES', 'MANAGER', 20);

1 개의 행이 만들어졌습니다.

SQL> INSERT INTO EMP03
  2  VALUES(NULL, 'JONES', 'SALESMAN', 10);

1 개의 행이 만들어졌습니다.

SQL> SELECT * FROM EMP03;

   EMPNO ENAME      JOB          DEPTNO
-----
   7499 ALLEN      SALESMAN           30
      JONES      MANAGER           20
      JONES      SALESMAN           10

SQL>
```

```
-- PRIMARY KEY 제약 조건 설정하기
DROP TABLE EMP05;
CREATE TABLE EMP05(
EMPNO NUMBER(4) CONSTRAINT EMP05_EMPNO_PK PRIMARY KEY,
ENAME VARCHAR2(10) CONSTRAINT EMP05_ENAME_NN NOT NULL,
JOB VARCHAR2(9),
DEPTNO NUMBER(2)
);

INSERT INTO EMP05 VALUES(7499,'ALLEN','SALESMAN',30);
SELECT * FROM EMP05;

INSERT INTO EMP05 VALUES(7499,'JONES','MANAGER',20); -- SQL Error: ORA-00001: unique constraint
(SCOTT.EMP05_EMPNO_PK) violated
INSERT INTO EMP05 VALUES(NULL,'JONES','MANAGER',20); -- SQL Error: ORA-01400: cannot insert NULL into
(SCOTT"."EMP05"."EMPNO")
```

13.6. 참조 무결성을 위한 FOREIGN KEY 제약 조건

- 참조의 무결성은 테이블 사이의 관계에서 발생하는 개념이므로 사원 테이블과 부서 테이블의 관계를 예를 들어 설명하면 ...

사원은 회사 내에 존재하는 부서에 소속되어 있어야 한다.

- 테이블을 생성하기에 앞서 데이터베이스 모델링 과정에서 업무를 분석한 후 얻어낸 테이블의 관계(개체와 관계)를 다이어그램으로 나타낸 것이다. (ERD, Entity Relationship Diagram)



- 외래 키(FOREIGN KEY) 제약조건은 사원 테이블의 부서 번호는 반드시 부서 테이블에 존재하는 부서 번호를 참조 가능하도록 하는 것이다. (참조의 무결성)

부서(dept) 테이블의 기본 키인 부서 번호(deptno) 컬럼을 부모 키라고 함

```
SQL> SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

사원(emp) 테이블의 부서 번호(deptno) 컬럼은 외래 키로 지정해야만 참조의 무결성이 설정됨

```
SQL> SELECT * FROM EMP;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80/12/17	800		20
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
7566	JONES	MANAGER	7839	81/04/02	2975		20
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850		30
7782	CLARK	MANAGER	7839	81/06/09	2450		10
7788	SCOTT	ANALYST	7566	87/04/19	3000		20
7839	KING	PRESIDENT		81/11/17	5000		10
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
7876	ADAMS	CLERK	7788	87/05/23	1100		20
7900	JAMES	CLERK	7698	81/12/03	950		30
7902	FORD	ANALYST	7566	81/12/03	3000		20
7934	MILLER	CLERK	7782	82/01/23	1300		10

14 개의 행이 선택되었습니다.

-- 다음은 EMP 테이블과 DEPT 테이블의 제약 조건을 확인한다.

```
SELECT TABLE_NAME, CONSTRAINT_TYPE,
CONSTRAINT_NAME, R_CONSTRAINT_NAME
FROM USER_CONSTRAINTS
WHERE TABLE_NAME IN ('DEPT', 'EMP');
```

-- 외래 키 제약 조건 설정하기

```
DROP TABLE EMP06;
CREATE TABLE EMP06(
EMPNO NUMBER(4) CONSTRAINT EMP06_EMPNO_PK PRIMARY KEY ,
ENAME VARCHAR2(10) CONSTRAINT EMP06_ENAME_NN NOT NULL,
JOB VARCHAR2(9),
DEPTNO NUMBER(2) CONSTRAINT EMP06_DEPTNO_FK REFERENCES DEPT(DEPTNO)
);
```

```
INSERT INTO EMP06 VALUES(7499,'ALLEN','SALESMAN',30);
```

```
SELECT * FROM EMP06;
```

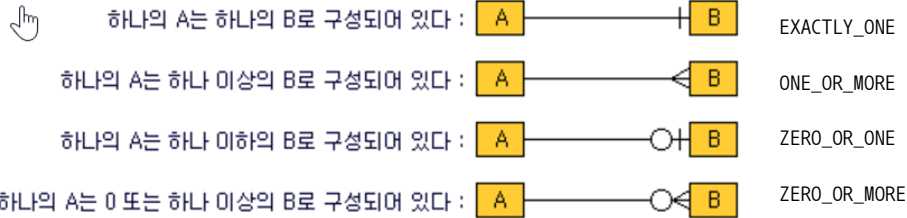
```
INSERT INTO EMP06 VALUES(7499,'JONES','MANAGER',50); -- SQL Error: ORA-00001: unique constraint
(SCOTT.EMP06_EMPNO_PK) violated
```

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, R_CONSTRAINT_NAME
```

```
FROM USER_CONSTRAINTS
WHERE TABLE_NAME='EMP06';
```

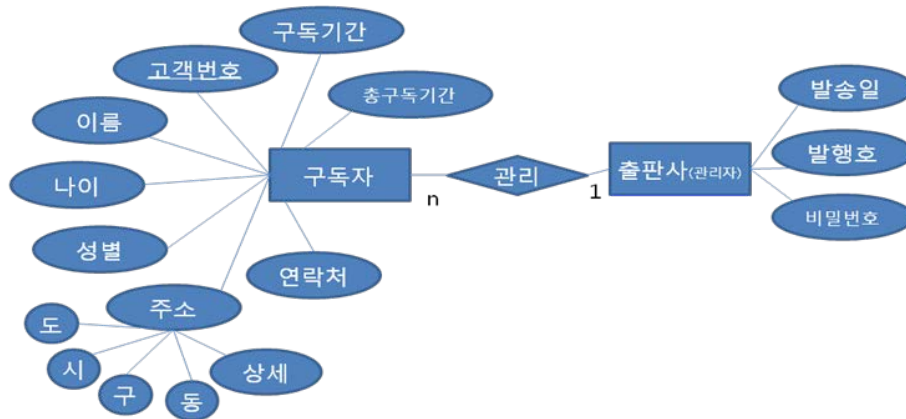
[참조] ERD(Entity Relationship Diagram)

1. ERD 표기 방법

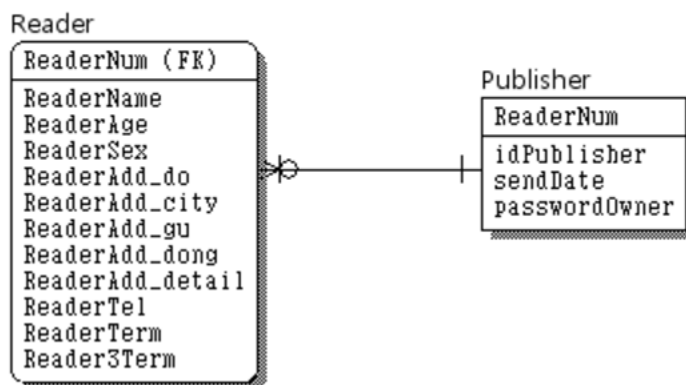


2. 논리적 설계

(1) 데이터베이스 모델링



(2) ERD 작성 (새발표기법)



3. 물리적 설계 (DDL 작성)

```
CREATE TABLE Publisher(
  sendDate DATE,
  passwordOwner NUMBER,
  idPublisher NUMBER,
  ReaderNum NUMBER NOT NULL
```

```
);

DROP INDEX XPKPublisher;
CREATE UNIQUE INDEX XPKPublisher ON Publisher (
    ReaderNum ASC
);

ALTER TABLE Publisher
ADD PRIMARY KEY(ReaderNum);
...
```

13.7. CHECK와 DEFAULT의 제약 조건

- CHECK 제약 조건은 입력되는 값을 체크하여 설정된 값 이외의 값이 들어오면 오류 메시지와 함께 명령이 수행되지 못하게 하는 것이다.
- 디폴트는 아무런 값을 입력 하지 않았을 때 디폴트제약의 값이 입력이 된다.

```
-- CHECK 제약 조건 설정하기
DROP TABLE EMP07;
CREATE TABLE EMP07(
    EMPNO NUMBER(4) CONSTRAINT EMP07_EMPNO_PK PRIMARY KEY,
    ENAME VARCHAR2(10) CONSTRAINT EMP07_ENAME_NN NOT NULL,
    SAL NUMBER(7,2) CONSTRAINT EMP07_SAL_CK CHECK(SAL BETWEEN 500 AND 5000),
    GENDER VARCHAR2(1) CONSTRAINT EMP07_GENDER_CK CHECK(GENDER IN('M','F'))
);

INSERT INTO EMP07 VALUES(7499,'ALLEN',500,'M');
INSERT INTO EMP07 VALUES(7499,'ALLEN',7000,'A');    -- SQL Error:  ORA-02290:  check constraint
(SCOTT.EMP07_GENDER_CK) violated

SELECT TABLE_NAME, CONSTRAINT_TYPE, CONSTRAINT_NAME, SEARCH_CONDITION
FROM USER_CONSTRAINTS
WHERE TABLE_NAME='EMP07';

-- DEFAULT 제약 조건 설정하기
DROP TABLE DEPT01;
CREATE TABLE DEPT01(
    DEPTNO NUMBER(2) PRIMARY KEY,
    DNAME VARCHAR2(14),
    LOC VARCHAR2(13) DEFAULT 'SEOUL '
);

INSERT INTO DEPT01(DEPTNO,DNAME) VALUES(10,'ACCOUNTING');
SELECT * FROM DEPT01;
```

13.8. 테이블 레벨 방식으로 제약 조건 지정하기

■ 컬럼 레벨 제약 조건

- CREATE TABLE로 테이블을 생성하면서 컬럼을 정의하게 되는데 하나의 컬럼 정의가 다 마무리되기 전에 컬럼 명 다음에 타입을 지정하고 그 뒤에 연이어서 제약 조건을 지정하는 방식이다.

■ 테이블 레벨의 제약 조건

- 컬럼을 모두 정의하고 나서 테이블 정의를 마무리 짓기 전에 따로 생성된 컬럼들에 대한 제약 조건을 한꺼번에 지정하는 것이다.

■ 복합키로 기본키를 지정할 경우

- 지금까지는 한 개의 컬럼으로 기본키를 지정했다. 하지만, 경우에 따라서는 2개 이상의 컬럼이 하나의 기본키를 구성하는 경우가 있는데 이를 복합키라고 한다. 복합키 형태로 제약조건을 지정할 경우에는 컬럼 레벨 형식으로는 불가능하고 반드시 테이블 레벨 방식을 사용해야 한다.

■ ALTER TABLE로 제약 조건을 추가할 때

- 테이블의 정의가 완료되어서 이미 테이블의 구조가 결정된 후에 나중에 테이블에 제약 조건을 추가하고 할 때에는 테이블 레벨 방식으로 제약 조건을 지정해야 한다.

[실습] 컬럼레벨 제약조건과 테이블레벨 제약조건 설정하기

■ 컬럼 레벨과 테이블 레벨로 제약 조건을 지정하는 방법의 차이점을 살펴본다.

```
-- 컬럼 레벨로 제약 조건을 지정하는 방법
DROP TABLE EMP01;
CREATE TABLE EMP01(
EMPNO NUMBER(4) PRIMARY KEY,
ENAME VARCHAR2(10) NOT NULL,
JOB VARCHAR2(9) UNIQUE,
DEPTNO NUMBER(4) REFERENCES DEPT(DEPTNO)
);

-- 테이블 레벨로 제약 조건을 지정하는 방법
DROP TABLE EMP02;
CREATE TABLE EMP02(
EMPNO NUMBER(4),
ENAME VARCHAR2(10) NOT NULL,
JOB VARCHAR2(9),
DEPTNO NUMBER(4),
PRIMARY KEY(EMPNO),
UNIQUE(JOB),
FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO)
);
```

13.9. 제약 조건 변경하기

13.9.1. 제약 조건 추가하기

- 테이블 구조를 결정하는 DDL을 학습하면서 테이블이 이미 생성된 이후에 테이블의 구조를 변경하기 위한 명령어로 ALTER TABLE을 사용한다는 것을 이미 학습하였다.
- 제약조건 역시 이미 테이블을 생성하면서 지정해주는 것이었기에 테이블 생성이 끝난 후에 제약 조건을 추가하기 위해서는 ALTER TABLE로 추가해 주어야 한다.

```
-- 형식
ALTER TABLE table_name
ADD [CONSTRAINT constraint_name]
constraint_type (column_name);
```

13.9.2. MODIFY로 NOT NULL 제약 조건 추가하기

- 이미 존재하는 테이블에 무결성 제약 조건을 추가로 생성하기 위해서 ALTER TABLE . . . ADD . . . 명령문을 사용하였다.
- 하지만 NOT NULL 제약 조건은 ADD 대신 MODIFY 명령문을 사용하므로 사용에 주의해야 한다. 이는 'NULL을 허용하는 상태'에서 'NULL을 허용하지 않는 상태'로 변경하겠다는 의미로 이해하기 바란다.

13.9.3. 제약 조건 제거하기

- 제약 조건을 제거하기 위해서 DROP CONSTRAINT 다음에 제거하고자 하는 제약 조건 명을 명시해야 한다.

```
-- 형식
ALTER TABLE table_name
DROP [CONSTRAINT constraint_name];
```

13.10. 제약 조건의 비활성화와 CASCADE

13.10.1. 제약 조건에 의해 작업이 어려운 경우

- 제약 조건이 설정되면 항상 그 규칙에 따라 데이터 무결성이 보장된다.
- 특별한 업무를 수행하는 과정에서 이러한 제약 조건 때문에 작업이 진행되지 못하는 경우가 생긴다.
- 그렇다고 제약 조건을 삭제해 버리면 데이터 무결성을 보장받지 못하게 된다.
- 그렇기 때문에 오라클에서는 제약 조건을 비활성화시킴으로서 제약 조건을 삭제하지 않고도 제약 조건 사용을 잠시 보류할 수 있는 방법을 제공해준다.
- 이렇게 비활성화 된 제약 조건은 원하는 작업을 한 후에는 다시 활성화 상태로 만들어 주어야 한다.

13.10.2. 제약 조건의 비활성화

- 테이블에서 제약 조건을 삭제하지 않고 일시적으로 적용시키지 않도록 하는 방법으로 제약 조건을 비활성화하는 방법이 있다.

```
-- 형식
ALTER TABLE table_name
DISABLE [CONSTRAINT constraint_name];
```

13.10.3. 제약 조건의 활성화

- 제약 조건을 비활성화 해 보았으므로 이번에는 제약 조건을 활성화 해보도록 한다.

```
-- 형식
ALTER TABLE table_name
ENABLE [CONSTRAINT constraint_name];
```

13.10.4. CASCADE 옵션

- CASCADE 옵션은 부모 테이블과 자식 테이블 간의 참조 설정이 되어 있을 때 부모 테이블의 제약 조건을 비활성화하면 이를 참조하고 있는 자식 테이블의 제약 조건까지 같이 비활성화시켜 주는 옵션이다.
- 또한 제약 조건의 비활성화뿐만 아니라 제약 조건이 삭제에도 활용되며, 역시 같은 이치로 부모 테이블의 제약 조건을 삭제하면 이를 참조하고 있는 자식 테이블의 제약 조건도 같이 삭제된다.

[실습] CASCADE 옵션으로 제약 조건 연속적으로 비활성화 / 제거하기

```
DROP TABLE DEPT01 CASCADE CONSTRAINTS;
CREATE TABLE DEPT01
AS
SELECT * FROM DEPT;
ALTER TABLE DEPT01
ADD CONSTRAINT DEPT01_DEPTNO_PK PRIMARY KEY (DEPTNO);
DESC DEPT01;

DROP TABLE EMP01 CASCADE CONSTRAINTS;
CREATE TABLE EMP01
AS
SELECT * FROM EMP WHERE 1=0;
ALTER TABLE EMP01
ADD CONSTRAINT EMP01_DEPTNO_FK FOREIGN KEY(DEPTNO) REFERENCES DEPT01(DEPTNO);

SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE,
TABLE_NAME, R_CONSTRAINT_NAME, STATUS
FROM USER_CONSTRAINTS
WHERE TABLE_NAME IN('DEPT01', 'EMP01');

SELECT * FROM DEPT01;
SELECT * FROM EMP01;

INSERT INTO EMP01 (DEPTNO) VALUES (10);
SELECT * FROM EMP01;
INSERT INTO EMP01 (DEPTNO) VALUES (50);

ALTER TABLE DEPT01
DROP PRIMARY KEY CASCADE;
```

[과제] 과제-13-01.TXT

1. 다음과 같은 구조의 테이블을 생성해 보자. (단, 테이블 레벨방식 사용)

- 테이블 : ORDERS
- 컬럼 :

ORDER_ID	NUMBER(12,0)
ORDER_DATE	DATE
ORDER_MODE	VARCHAR2(8 BYTE)
CUSTOMER_ID	NUMBER(6,0)
ORDER_STATUS	NUMBER(2,0)
ORDER_TOTAL	NUMBER(8,2)
SALES_REP_ID	NUMBER(6,0)

- PROMOTION_ID NUMBER(6,0)
- 제약사항 : 기본키는 ORDER_ID
ORDER_MODE에는 'direct', 'online'만 입력가능
ORDER_TOTAL의 디폴트 값은 0

<정답>

2. 다음과 같은 구조의 테이블을 생성해 보자. (단, 테이블 레벨방식 사용)

- 테이블 : ORDER_ITEMS
- 컬럼 : ORDER_ID NUMBER(12,0)
LINE_ITEM_ID NUMBER(3,0)
PRODUCT_ID NUMBER(3,0)
UNIT_PRICE NUMBER(8,2)
QUANTITY NUMBER(8,0)
- 제약사항 : 기본키는 ORDER_ID와 LINE_ITEM_ID
UNIT_PRICE, QUANTITY 의 디폴트 값은 0

<정답>

14. 가상 테이블인 뷰

14.1. 뷰의 기본 다루기

- 뷰(View)는 한마디로 물리적인 테이블을 근거한 논리적인 가상 테이블이라고 정의할 수 있다.
- 가상이란 단어는 실질적으로 데이터를 저장하고 있지 않기 때문에 붙인 것이고, 테이블이란 단어는 실질적으로 데이터를 저장하고 있지 않더라도 사용자는 마치 테이블을 사용하는 것과 동일하게 뷰를 사용할 수 있기 때문에 붙인 것이다.
- 뷰는 기본 테이블에서 파생된 객체로서 기본 테이블에 대한 하나의 쿼리문이다.
- 뷰(View)란 '보다'란 의미를 갖고 있는 점을 감안해 보면 알 수 있듯이 실제 테이블에 저장된 데이터를 뷰를 통해서 볼 수 있도록 한다.
- 사용자에게 주어진 뷰를 통해서 기본 테이블을 제한적으로 사용하게 된다.

14.1.1. 뷰의 기본 테이블

- 뷰는 이미 존재하고 있는 테이블에 제한적으로 접근하도록 한다.
- 뷰를 생성하기 위해서는 실질적으로 데이터를 저장하고 있는 물리적인 테이블이 존재해야 하는데 이 테이블을 기본 테이블이라고 한다.

```
-- 뷰의 기본 테이블 생성하기
DROP TABLE DEPT_COPY;
CREATE TABLE DEPT_COPY
AS
SELECT * FROM DEPT;

DROP TABLE EMP_COPY;
CREATE TABLE EMP_COPY
AS
SELECT * FROM EMP;

SELECT * FROM DEPT_COPY;
SELECT * FROM EMP_COPY;
```

14.1.2. 뷰 정의하기

- 테이블을 생성하기 위해서 CREATE TABLE 로 시작하지만, 뷰를 생성하기 위해서는 CREATE VIEW로 시작한다. AS 다음은 마치 서브 쿼리문과 유사한다.

```
-- 형식
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name
[(alias, alias, alias, ...)]
AS subquery
[WITH CHECK OPTION]
[WITH READ ONLY];

-- 뷰를 생성할 권한이 불충분한 경우
CREATE OR REPLACE VIEW EMP_VIEW30
AS
SELECT EMPNO, ENAME, SAL, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30; -- SQL Error: ORA-01031: insufficient privileges
```

```

CONN system/sys
GRANT CREATE VIEW TO scott; -- system계정으로 로그인하여 뷰를 생성할 수 있는 권한을 부여한다.
CONN scott/tiger

SELECT * FROM EMP_VIEW30;

-- 뷰 정의하기
DROP VIEW EMP_VIEW30;
CREATE VIEW EMP_VIEW30
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30;

DESC EMP_VIEW30

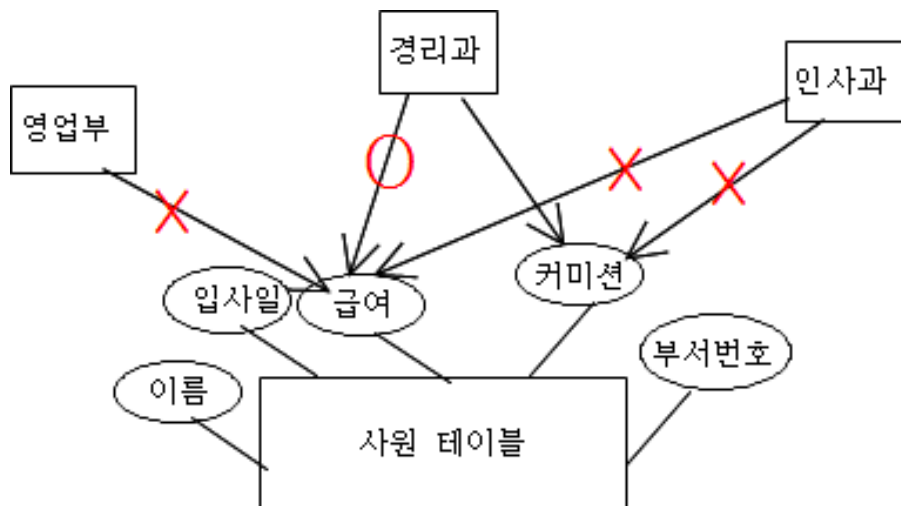
SELECT * FROM EMP_VIEW30;

```

14.2. 뷰 고급 다루기

14.2.1. 뷰를 사용하는 이유

- 뷰를 사용하는 이유는 두 가지로 설명할 수 있다.
 - 복잡하고 긴 쿼리문을 뷰로 정의하면 접근을 단순화시킬 수 있다.
 - 보안에 유리하다.
- 사용자마다 특정 객체만 조회할 수 있도록 권한을 부여를 할 수 있기에 동일한 테이블을 접근하는 사용자마다에 따라 서로 다르게 보도록 여러 개의 뷰를 정의해 놓고 특정 사용자만이 해당 뷰에 접근할 수 있도록 한다.



14.2.2. 단순뷰와 복합뷰

- 뷰는 뷰를 정의하기 위해서 사용되는 기본 테이블의 수에 따라 단순 뷰(Simple View)와 복합 뷰(Complex View)로 나뉜다.

단순 뷰	복합 뷰
하나의 테이블로 생성	여러 개의 테이블로 생성
그룹 함수의 사용이 불가능	그룹 함수의 사용이 가능

DISTINCT 사용이 불가능	DISTINCT 사용이 가능
DML 사용 가능	DML 사용 불가능

```
-- 단순 뷰의 컬럼에 별칭 부여하기
DESC EMP_VIEW30;
CREATE OR REPLACE
VIEW EMP_VIEW(사원번호, 사원명, 부서번호)
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMP_COPY;

SELECT * FROM EMP_VIEW
WHERE 부서번호=30;

-- 복합 뷰 만들기
CREATE OR REPLACE
VIEW EMP_VIEW_DEPT
AS
SELECT E.EMPNO, E.ENAME, E.SAL, E.DEPTNO, D.DNAME, D.LOC
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
ORDER BY EMPNO DESC;

SELECT * FROM EMP_VIEW_DEPT;
```

14.2.3. 뷰 삭제 알아보기

```
-- 뷰 삭제하기
SELECT VIEW_NAME, TEXT
FROM USER_VIEWS;

SELECT * FROM EMP_VIEW;
DROP VIEW EMP_VIEW;
SELECT * FROM EMP_VIEW;
```

14.3. 뷰 생성에 사용되는 다양한 옵션

14.3.1. 뷰 수정을 위한 OR REPLACE 옵션

- CREATE OR REPLACE VIEW 를 사용하면 존재하지 않은 뷰이면 새로운 뷰를 생성하고 기존에 존재하는 뷰이면 그 내용을 변경한다.

```
CREATE OR REPLACE VIEW EMP_VIEW30
AS
SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30;
```

14.3.2. 기본 테이블 없이 뷰를 생성하기 위한 FORCE 옵션

- 뷰를 생성하는 경우에 일반적으로 기본 테이블이 존재한다는 가정 하에서 쿼리문을 작성한다.
- 극히 드물기는 하지만, 기본 테이블이 존재하지 않는 경우에도 뷰를 생성해야 할 경우가 있습니다. 이럴 경우에 사용하는 것이 FORCE 옵션이다.
- FORCE 옵션과 반대로 동작하는 것으로서 NOFORCE 옵션이 있다.
- NOFORCE 옵션은 반드시 기본 테이블이 존재해야 할 경우에만 뷰가 생성된다.
- 특별한 설정이 없으면 디폴트로 NOFORCE 옵션이 지정된 것이므로 간주한다.

```
-- FORCE 옵션으로 기본 테이블 없이 뷰 생성하기
DESC EMPLOYEES

CREATE OR REPLACE VIEW EMPLOYEES_VIEW
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMPLOYEES
WHERE DEPTNO=30; -- SQL Error: ORA-00942: table or view does not exist

CREATE OR REPLACE FORCE VIEW NOTABLE_VIEW
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMPLOYEES
WHERE DEPTNO=30;

SELECT VIEW_NAME, TEXT
FROM USER_VIEWS;

-- NOFORCE는 뷰 생성의 디폴트값으로 FORCE의 반대 기능을 가진 옵션이다.
CREATE OR REPLACE NOFORCE VIEW EXISTTABLE_VIEW
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMPLOYEES
WHERE DEPTNO=30; -- SQL Error: ORA-00942: table or view does not exist

SELECT VIEW_NAME, TEXT
FROM USER_VIEWS;

CREATE OR REPLACE VIEW EXISTTABLE_VIEW
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMPLOYEES
WHERE DEPTNO=30;
```

14.3.3. 조건 컬럼값을 변경하지 못하게 하는 WITH CHECK OPTION

- 뷰를 생성할 때 조건 제시에 사용된 컬럼 값을 변경 못하도록 하는 기능을 제공한다.

```
CREATE OR REPLACE VIEW VIEW_CHK30
AS
SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30 WITH CHECK OPTION;

UPDATE VIEW_CHK30 SET DEPTNO=20
WHERE SAL>=1200;
SELECT * FROM VIEW_CHK30;
```


14.3.4. 기본 테이블 변경을 막는 WITH READ ONLY 옵션 WITH CHECK OPTION 비교

- WITH READ ONLY 옵션은 뷰를 통해서는 기본 테이블의 어떤 컬럼에 대해서도 내용을 절대 변경할 수 없도록 하는 것이다.

```
UPDATE VIEW_CHK30 SET COMM=1000;
SELECT * FROM VIEW_CHK30;

CREATE OR REPLACE VIEW VIEW_READ30
AS
SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30 WITH READ ONLY;

UPDATE VIEW_READ30 SET COMM=1000;
SELECT * FROM VIEW_READ30;
```

14.4. 뷰 활용하여 Top-N 구하기

- TOP-N을 구하기 위해서는 ROWNUM과 인라인 뷰가 사용된다.

14.4.1. ROWNUM 컬럼 성격 파악하기

```
SELECT ROWNUM, EMPNO, ENAME, HIREDATE
FROM EMP;

SELECT EMPNO, ENAME, HIREDATE
FROM EMP
ORDER BY HIREDATE;

SELECT ROWNUM, EMPNO, ENAME, HIREDATE
FROM EMP
ORDER BY HIREDATE;
```

14.4.2. 인라인 뷰로 구하는 TOP-N의 개념

- 인라인 뷰란 메인 쿼리의 SELECT 문의 FROM 절 내부에 사용된 서브 쿼리문을 말한다.
- 우리가 지금까지 생성한 뷰는 CREATE 명령어로 뷰를 생성했지만, 인라인 뷰는 SQL 문 내부에 뷰를 정의하고 이를 테이블처럼 사용한다.

```
-- 급여(SAL)를 많이 받는 6~10째 사원을 출력하기 위해서는 인라인 뷰안에 또 다른 인라인 뷰를 사용해야 한다.
-- 또한 ROWNUM 컬럼에 별칭을 부여해야 검색이 가능하다.
SELECT ROWNUM, RNUM, ENAME, SAL FROM
(SELECT ROWNUM RNUM, ENAME, SAL FROM
(SELECT * FROM EMP ORDER BY SAL DESC))
WHERE RNUM BETWEEN 6 AND 10;
```

[꿀팁] MySQL에서 ROW COUNT 방법

```
SELECT @no:=@no+1 AS no, article_no, writer_id, title, regdate  
FROM article, (SELECT @no:=0) article  
ORDER BY article_no DESC;
```

15. 시퀀스

15.1. 시퀀스의 개념 이해와 시퀀스 생성

- 오라클에서는 행을 구분하기 위해서 기본 키를 두고 있다. 기본 키는 중복된 값을 가질 수 없으므로 항상 유일한 값을 가져야 한다.
- 기본 키가 유일한 값을 갖도록 사용자가 직접 값을 생성해내려면 부담이 클 것이다.
- 시퀀스는 테이블 내의 유일한 숫자를 자동으로 생성하는 자동 번호 발생기이므로 시퀀스를 기본 키로 사용하게 되면 사용자의 부담을 줄일 수 있다.

```
-- 형식
CREATE SEQUENCE sequence_name
    [START WITH n]                               ①
    [INCREMENT BY n]                             ②
    [{MAXVALUE n | NOMAXVALUE}]                   ③
    [{MINVALUE n | NOMINVALUE}]                   ④
    [{CYCLE | NOCYCLE}]                             ⑤
    [{CACHE n | NOCACHE}]                         ⑥

-- 예
CREATE SEQUENCE DEPT_DEPTNO_SEQ
INCREMENT BY 10
START WITH 10;
```

- ① START WITH : 시퀀스 번호의 시작값을 지정할 때 사용된다. 만일 1부터 시작되는 시퀀스를 생성하려면 START WITH 1이라고 기술하면 된다.
- ② INCREMENT BY : 연속적인 시퀀스 번호의 증가치를 지정할 때 사용된다. 만일 1씩 증가하는 시퀀스를 생성하려면 INCREMENT BY 1이라고 기술하면 된다.
- ③ MAXVALUE *n* | NOMAXVALUE : MAXVALUE 은 시퀀스가 가질 수 있는 최대값을 지정한다. 만일 NOMAXVALUE를 지정하게 되면 ASCENDING 순서일 경우에는 10^{27} 승이고 DESCENDING 순서일 경우에는 -1로 설정된다.
- ④ MINVALUE *n* | NOMINVALUE : MINVALUE 은 시퀀스가 가질 수 있는 최소값을 지정한다. 만일 NOMINVALUE을 지정하게 되면 ASCENDING 순서일 경우에는 1이고 DESCENDING 순서일 경우에는 10^{26} 승으로 설정된다.
- ⑤ CYCLE | NOCYCLE : CYCLE 은 지정된 시퀀스 값이 최대값까지 증가가 완료되게 되면 다시 START WITH 옵션에 지정한 시작 값에서 다시 시퀀스를 시작하도록 한다. NOCYCLE은 증가가 완료되게 되면 에러를 유발시킨다.
- ⑥ CACHE *n* | NOCACHE : CACHE 은 메모리상의 시퀀스 값을 관리하도록 하는 것인데 기본 값은 20이다. NOCACHE는 원칙적으로 메모리 상에서 시퀀스를 관리하지 않는다.

15.2. 시퀀스 관련 데이터 덱서너리

- 생성된 시퀀스 객체에 대한 정보를 저장하는 데이터 덱서너리는 USER_SEQUENCES 이다.

```
DESC USER_SEQUENCES

SELECT SEQUENCE_NAME, MIN_VALUE, MAX_VALUE,
INCREMENT_BY, CYCLE_FLAG
FROM USER_SEQUENCES;
```

15.3. 시퀀스 값을 알아보는 CURRVAL과 NEXTVAL

- 시퀀스의 현재 값을 알아내기 위해서 CURRVAL를 사용하고, 다음 값을 알아내기 위해서는 NEXTVAL를 사용한다.

```
SELECT DEPT_DEPTNO_SEQ.NEXTVAL FROM DUAL;  
SELECT DEPT_DEPTNO_SEQ.CURRVAL FROM DUAL;
```

15.4. 시퀀스 실무에 적용하기

- 시퀀스는 99.9%가 INSERT 연산과 같이 사용되어 컬럼 값을 자동으로 발생시키는 용도로 사용된다.
- 아래 예는 사원 테이블을 생성하면서 사원 번호를 기본 키로 설정하였다.
- 기본 키는 반드시 유일한 값을 가져야 한다. 사용자가 새로운 사원을 추가할 때마다 유일한 사원번호를 INSERT 해야 하는 번거로움이 있다.
- 사원 번호를 생성하는 시퀀스 객체를 사용하여 사원 번호가 자동 생성되도록 한다면 이러한 번거로움을 덜어줄 수 있다.

```
-- 예: 시퀀스를 테이블의 기본 키에 적용하기  
CREATE SEQUENCE EMP_SEQ  
START WITH 1  
INCREMENT BY 1  
MAXVALUE 100000 ;  
  
DROP TABLE EMP01;  
CREATE TABLE EMP01(  
EMPNO NUMBER(4) PRIMARY KEY,  
ENAME VARCHAR(10),  
HIREDATE DATE  
);  
  
INSERT INTO EMP01  
VALUES(EMP_SEQ.NEXTVAL, 'JULIA' , SYSDATE);  
  
SELECT * FROM EMP01;
```

15.5. 시퀀스 제거와 수정

15.5.1. 시퀀스를 삭제하는 방법

- 생성되어 있는 시퀀스를 제거하려면 DROP SEQUENCE문을 사용해야 한다.

```
SELECT SEQUENCE_NAME, MAX_VALUE, INCREMENT_BY, CYCLE_FLAG  
FROM USER_SEQUENCES;  
  
DROP SEQUENCE DEPT_DEPTNO_SEQ;
```

15.5.2. 시퀀스를 수정하는 방법

- 시퀀스를 변경하려면 ALTER SEQUENCE 문을 사용해야 한다.
- ALTER SEQUENCE는 START WITH 절이 없다는 점을 빼고는 CREATE SEQUENCE와 구조가 동일하다.
- START WITH 옵션은 ALTER SEQUENCE를 써서 변경할 수 없다.
- 다른 번호에서 다시 시작하려면 이전 시퀀스를 삭제하고 다시 생성해야 한다.

```
-- 형식
ALTER SEQUENCE sequence_name
[INCREMENT BY n]
[{{MAXVALUE n | NOMAXVALUE}}]
[{{MINVALUE n | NOMINVALUE}}]
[{{CYCLE | NOCYCLE}}]
[{{CACHE n | NOCACHE}}]

-- 예 : 시퀀스 최대값을 변경하기
DROP SEQUENCE DEPT_DEPTNO_SEQ;
CREATE SEQUENCE DEPT_DEPTNO_SEQ
START WITH 10
INCREMENT BY 10
MAXVALUE 30; -- 10부터 10씩 증가하면서 최대 30까지의 값을 갖는 시퀀스를 생성한다.

SELECT DEPT_DEPTNO_SEQ.NEXTVAL FROM DUAL;
SELECT DEPT_DEPTNO_SEQ.NEXTVAL FROM DUAL;
SELECT DEPT_DEPTNO_SEQ.NEXTVAL FROM DUAL;
SELECT DEPT_DEPTNO_SEQ.NEXTVAL FROM DUAL; -- 부서 번호를 계속 생성하다 보면 최대값을 넘게 된다.

ALTER SEQUENCE DEPT_DEPTNO_SEQ
MAXVALUE 100; -- 최대값을 넘을 때까지 시퀀스를 생성한다.

SELECT SEQUENCE_NAME, MAX_VALUE, INCREMENT_BY, CYCLE_FLAG
FROM USER_SEQUENCES; -- USER_SEQUENCES 를 조회하면 시퀀스가 수정되었는지 확인할 수 있다.
```

[과제] 과제-15-01.TXT

1. 최소값 1, 최대값 99999999, 1000부터 시작해서 1씩 증가하는 ORDERS_SEQ 라는 시퀀스를 만들어보자.

<정답>

16 인덱스

16.1 인덱스의 개요

16.1.1 인덱스란?

- 인덱스란 SQL 명령문의 처리 속도를 향상시키기 위해서 컬럼에 대해서 생성하는 오라클 객체이다.
- 인덱스의 장점
 - 검색 속도가 빨라진다.
 - 시스템에 부하를 줄여서 시스템의 전체 성능을 향상시킨다.
- 인덱스의 단점
 - 인덱스를 위한 추가 공간이 필요하다.
 - 인덱스를 생성하는 데 시간이 걸린다.
 - 데이터의 변경 작업(INSERT/UPDATE/DELETE)이 자주 일어날 때는 오히려 성능이 저하된다.

16.1.2 인덱스 정보 조회

- USER_COLUMNS와 USER_IND_COLUMNS 데이터 디렉터리 뷰를 살펴보아야 한다.
- 인덱스는 기본 키나 유일 키와 같은 제약 조건을 지정하면 따로 생성하지 않더라도 자동으로 생성된다.

```
COLUMN TABLE_NAME FORMAT A15
COLUMN COLUMN_NAME FORMAT A15  -- 컬럼의 출력 형식 조정하기

SELECT INDEX_NAME, TABLE_NAME , COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME IN('EMP', 'DEPT');
```

16.1.3 조회 속도 비교하기

- 인덱스가 조회 속도를 빠르게 해 준다는 것을 증명하기 위해서 기본 키나 유일 키로 지정하지 않는 컬럼인 사원 이름으로 검색해 본다. 아마도 시간이 어느 정도 소요될 것이다.
- 검색을 위해서 WHERE 절에 사용되는 컬럼인 사원 이름 컬럼을 인덱스로 생성한 후에 다시 한번 사원 이름으로 검색해보면 검색시간이 현저하게 줄어드는 것을 확인할 수 있다.

[실습] 인덱스가 아닌 컬럼으로 검색하기

```
-- 사원 테이블 복사하기
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;

SELECT TABLE_NAME, INDEX_NAME, COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME IN('EMP', 'EMP01');
```

```
-- 인덱스가 아닌 컬럼으로 검색하기
INSERT INTO EMP01 SELECT * FROM EMP01;
--... 테이블 자체 복사를 여러 번 반복해서 상당히 많은 양의 행을 생성한다.
INSERT INTO EMP01 SELECT * FROM EMP01;
INSERT INTO EMP01(EMPNO, ENAME) VALUES(1111, 'SYJ');

SET TIMING ON
SELECT DISTINCT EMPNO, ENAME
FROM EMP01
WHERE ENAME='SYJ';
```

16.1.4 인덱스 생성 및 제거하기

- 제약 조건에 의해 자동으로 생성되는 인덱스 외에 CREATE INDEX 명령어로 직접 인덱스를 생성할 수도 있다.

```
-- 형식
CREATE INDEX index_name
ON table_name (column_name);

DROP INDEX index_name;
```

[실습] 인덱스 설정하여 조회속도 비교하기

```
-- 인덱스 설정하기
CREATE INDEX IDX_EMP01_ENAME
ON EMP01(ENAME);

SELECT DISTINCT EMPNO, ENAME
FROM EMP01
WHERE ENAME='SYJ';

-- 인덱스 제거하기
SELECT INDEX_NAME, TABLE_NAME , COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME IN('EMP01');

DROP INDEX IDX_EMP01_ENAME;

SELECT DISTINCT EMPNO, ENAME
FROM EMP01
WHERE ENAME='SYJ';
```

16.1.5 인덱스를 사용해야 하는 경우 판단하기

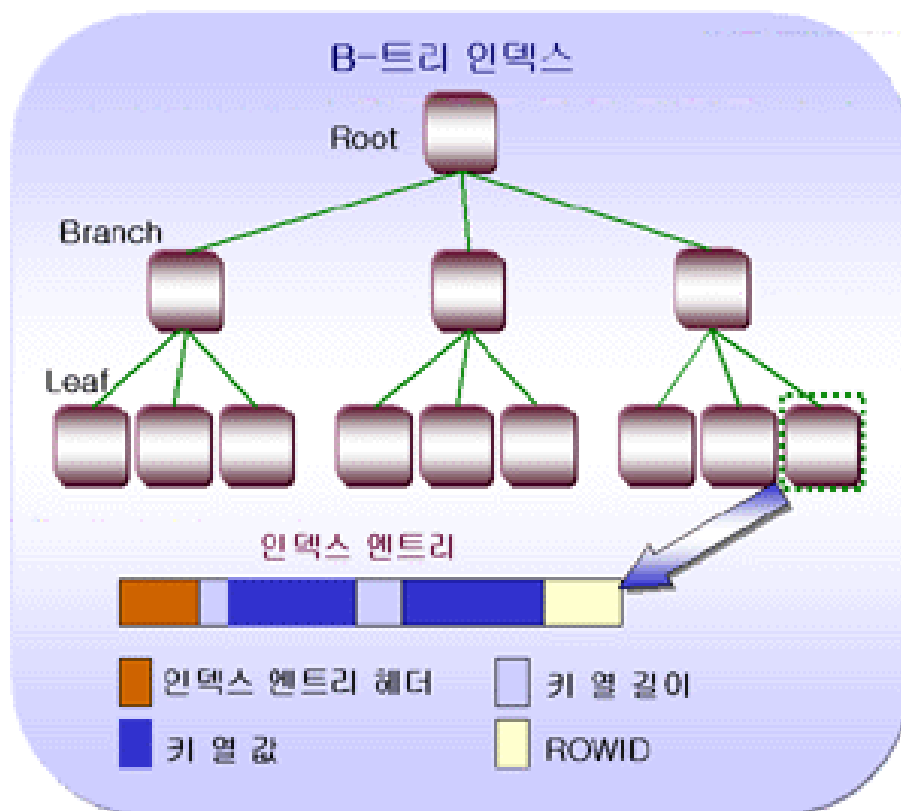
인덱스를 사용해야 하는 경우	인덱스를 사용하지 말아야 하는 경우
테이블에 행의 수가 많을 때	테이블에 행의 수가 적을 때
WHERE 문에 해당 컬럼이 많이 사용될 때	WHERE 문에 해당 컬럼이 자주 사용되지 않을 때
검색 결과가 전체 데이터의 2%~4% 정도 일 때	검색 결과가 전체 데이터의 10%~15% 이상 일 때
JOIN에 자주 사용되는 컬럼이나 NULL을 포함하는 컬럼이 많은 경우	테이블에 DML 작업이 많은 경우 즉, 입력 수정 삭제 등이 자주 일어 날 때

- 다음과 같은 조건에서 사원 테이블의 부서번호(DEPTNO)에 인덱스를 거는 것이 좋을까요?
 - 테이블에 전체 행의 수는 10000 건이다.
 - 위의 쿼리문을 전체 쿼리문들 중에서 95% 사용된다.
 - 쿼리문의 결과로 구해지는 행은 10건 정도이다.
- 조건을 위 표를 비추어보고 판단해 보면 부서번호(DEPTNO) 컬럼을 인덱스로 사용하기에 알맞다는 결론이 난다.

16.2 인덱스의 물리적인 구조와 재생성

16.2.1 B-트리 인덱스 구조

- 오라클에서의 인덱스의 내부 구조는 B-트리 형식으로 구성되어 있다.
- 뿌리(루트)를 근거로 아래로 나무뿌리 들이 뻗어 있는 모양을 하고 있다.



16.2.2 B-트리 인덱스의 추가, 삭제

- DML 작업 특히 DELETE 명령을 수행한 경우에는 해당 인덱스 엔트리가 논리적으로만 제거 되고 실제 인덱스 엔트리는 그냥 남아 있게 된다.
- 인덱스에 제거된 엔트리가 많아질 경우에는 제거된 인덱스들이 필요 없는 공간을 차지하고 있기 때문에 종종 인덱스를 재생성시켜야 한다.

```
-- 형식
ALTER INDEX index_name REBUILD;

-- 예
```



```
ALTER INDEX IDX_EMP01_ENAME REBUILD;
```

16.3 인덱스의 종류 살펴보기

16.3.1 고유와 비고유 인덱스

- 고유 인덱스(유일 인덱스라고도 부름)는 기본키나 유일키처럼 유일한 값을 갖는 컬럼에 대해서 생성하는 인덱스이다.
- 반면 비고유 인덱스는 중복된 데이터를 갖는 컬럼에 대해서 인덱스를 생성하는 경우를 말한다.
- 고유 인덱스를 설정하려면 UNIQUE 옵션을 추가해서 인덱스를 생성해야 한다.

```
-- 형식  
CREATE UNIQUE INDEX index_name  
ON table_name (column_name);
```

[실습] 고유 인덱스와 비고유 인덱스 정의하기

- 고유 인덱스와 비고유 인덱스를 비교하기 위해서 중복된 데이터가 없는 컬럼(DEPTNO)과 있는 컬럼(LOC)으로 구성된 부서 테이블을 만들어 보겠다.

```
DROP TABLE DEPT01;  
CREATE TABLE DEPT01  
AS  
SELECT * FROM DEPT WHERE 1=0;  
  
INSERT INTO DEPT01 VALUES(10, '인사과', '서울');  
INSERT INTO DEPT01 VALUES(20, '총무과', '대전');  
INSERT INTO DEPT01 VALUES(30, '교육팀', '대전');  
SELECT * FROM DEPT01;  
  
CREATE UNIQUE INDEX IDX_DEPT01_DEPTNO  
ON DEPT01(DEPTNO);  
  
CREATE UNIQUE INDEX IDX_DEPT01_LOC  
ON DEPT01(LOC);  
-- 중복된 데이터를 갖는 컬럼을 인덱스로 지정하면 오류가 발생한다.  
  
CREATE INDEX IDX_DEPT01_LOC  
ON DEPT01(LOC);
```

16.3.2 결합 인덱스 정의하기

- 두 개 이상의 컬럼으로 인덱스를 구성하는 것을 결합 인덱스라고 한다.

```
CREATE INDEX IDX_DEPT01_COM  
ON DEPT01(DEPTNO, DNAME);  
  
SELECT INDEX_NAME, COLUMN_NAME
```

```
FROM USER_IND_COLUMNS
WHERE TABLE_NAME = 'DEPT01';
```

16.3.3 함수 기반 인덱스 정의하기

- 검색조건으로 WHERE SAL = 300이 아니라 WHERE SAL*12 = 3600와 같이 SELECT 문 WHERE 절에 산술 표현 또는 함수를 사용하는 경우가 있다.
- 이 경우 만약 SAL 컬럼에 인덱스가 걸려 있다면 인덱스를 타서 빠르리라 생각 할 수도 있지만 실상은 SAL 컬럼에 인덱스가 있어도 SAL*12는 인덱스를 타지 못한다.
- 인덱스 걸린 컬럼이 수식으로 정의 되어 있거나 SUBSTR 등의 함수를 사용해서 변형이 일어난 경우는 인덱스를 타지 못하기 때문이다.
- 이러한 수식으로 검색하는 경우가 많다면 아예 수식이나 함수를 적용하여 인덱스를 만들 수 있다. SAL*12로 인덱스를 만들어 놓으면 SAL*12가 검색 조건으로 사용될 시 해당 인덱스를 타게 된다.

```
SET TIMING ON
-- 인덱스 이전 검색하기
SELECT * FROM EMP01
WHERE SAL*12 = 3600;

CREATE INDEX IDX_EMP01_ANNSAL
ON EMP01(SAL*12);

SELECT INDEX_NAME, COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME = 'EMP01';

-- 인덱스 이후 검색하기
SELECT * FROM EMP01
WHERE SAL*12 = 3600;
```

17 사용자 관리

17.1 데이터베이스 보안을 위한 권한

17.1.1 보안을 위한 데이터베이스 관리자

- 데이터베이스 관리자는 사용자가 데이터베이스의 객체(테이블, 뷰 등)에 대한 특정 권한을 가질 수 있도록 함으로서 다수의 사용자가 데이터베이스에 저장된 정보를 공유하면서도 정보에 대한 보안이 이루어지도록 한다.
- 사용자마다 서로 다른 권한과 롤을 부여함으로서 보안을 설정할 수 있다.

17.1.2 권한의 역할과 종류

- 권한은 사용자가 특정 테이블을 접근할 수 있도록 하거나 해당 테이블에 SQL(SELECT/INSERT/UPDATE/DELETE) 문을 사용할 수 있도록 제한을 두는 것을 말한다.
- 데이터베이스 보안을 위한 권한은 시스템 권한(System Privileges)과 객체 권한(Object Privileges)으로 나뉜다.
- 시스템 권한은 사용자의 생성과 제거, DB 접근 및 각종 객체를 생성할 수 있는 권한 등 주로 DBA에 의해 부여되며 그 권한의 수가 80 가지가 넘기에 대표적인 시스템 권한은 다음과 같다.

시스템 권한	기능
CREATE USER	새롭게 사용자를 생성하는 권한
DROP USER	사용자를 삭제하는 권한
DROP ANY TABLE	임의의 테이블을 삭제할 수 있는 권한
QUERY REWRITE	함수 기반 인덱스를 생성하는 권한
BACKUP ANY TABLE	임의의 테이블을 백업할 수 있는 권한
CREATE SESSION	데이터베이스에 접속할 수 있는 권한
CREATE TABLE	사용자 스키마에서 테이블을 생성할 수 있는 권한
CREATE VIEW	사용자 스키마에서 뷰를 생성할 수 있는 권한
CREATE SEQUENCE	사용자 스키마에서 시퀀스를 생성할 수 있는 권한
CREATE PROCEDURE	사용자 스키마에서 함수를 생성할 수 있는 권한

17.2 사용자 생성하기

- 오라클 데이터베이스를 설치할 때 자동으로 생성되는 디폴트 사용자 가운데 시스템 권한을 가진 데이터베이스 관리자인 DBA는 SYS, SYSTEM 이다.
- 사용자 계정을 발급 받기 위해서 시스템 권한을 가진 SYSTEM으로 접속해야 한다.

```
-- 형식
CREATE USER USER_NAME IDENTIFIED BY PASSWORD;

-- 예
CREATE USER USER10 IDENTIFIED BY USER10;
ALTER USER SYSTEM IDENTIFIED BY MANAGER; -- 비밀번호 변경하기
ALTER USER SCOTT IDENTIFIED BY TIGER;
```

[실습] CREATE USER 명령어를 이용한 사용자 생성

- CREATE USER 명령어를 사용하여 사용자명은 USER01 암호는 TIGGER로 사용자를 생성해 보겠

다.

```
CONN SYSTEM/MANAGER
SHOW USER

DROP USER USER01 CASCADE;
CREATE USER USER01 IDENTIFIED BY TIGER;

CONN USER01/TIGER
-- 접속에 실패한다. 사용자에게 아무런 권한이 부여되지 않았기 때문이다.
```

[LAB_17_1.SQL]

```
01 CONN SYSTEM/MANAGER
02 SHOW USER
03
04 DROP USER USER01 CASCADE;
05 CREATE USER USER01 IDENTIFIED BY TIGER;
06
07 CONN USER01/TIGER
08 -- 접속에 실패한다. 사용자에게 아무런 권한이 부여되지 않았기 때문이다.
```

17.3 권한을 부여하는 GRANT 명령어

- 사용자에게 시스템 권한 부여하기 위해서는 GRANT 명령어를 사용한다.

```
-- 형식
GRANT PRIVILEGE_NAME, ...
TO USER_NAME;
```

[실습] CREATE SESSION 권한 부여하기

- 새로 생성된 USER01에 데이터베이스에 접속할 수 있는 권한인 CREATE SESSION을 부여한다. 다시 USER01 사용자로 접속을 시도하면 이번에는 데이터베이스에 성공적으로 접속된다.

[LAB_17_2.SQL]

```
01 CONN SYSTEM/MANAGER
02 SHOW USER
03
04 DROP USER USER01 CASCADE;
05 CREATE USER USER01 IDENTIFIED BY TIGER;
06
07 GRANT CREATE SESSION TO USER01;
08
09 CONN USER01/TIGER
```

[실습] CREATE TABLE 권한 부여하기

- 새롭게 생성된 사용자 계정은 테이블을 생성할 권한을 부여받아야만 CREATE TABLE 명령을 사

용할 수 있다.

[LAB_17_3.SQL]

```
01 DROP TABLE EMP01;
02 CREATE TABLE EMP01(
03 EMPNO NUMBER(4),
04 ENAME VARCHAR2(10),
05 JOB VARCHAR2(9),
06 DEPTNO NUMBER(2)
07 );
08
09 CONN SYSTEM/MANAGER
10 SHOW USER
11
12 GRANT CREATE TABLE, RESOURCE TO USER01;
13
14 CONN USER01/TIGER
15 SHOW USER
16
17 DROP TABLE EMP01;
18 CREATE TABLE EMP01(
19 EMPNO NUMBER(4),
20 ENAME VARCHAR2(10),
21 JOB VARCHAR2(9),
22 DEPTNO NUMBER(2)
23 );
```

17.3.1 테이블 스페이스 확인하기

- 테이블스페이스(Tablespace)는 디스크 공간을 소비하는 테이블과 뷰 그리고 그 밖의 다른 데이터베이스 객체들이 저장되는 장소이다.

```
CONN SYSTEM/MANAGER
SHOW USER

SELECT USERNAME, DEFAULT_TABLESPACE
FROM DBA_USERS
WHERE USERNAME IN('USER01', 'SCOTT');

-- 테이블 스페이스 쿼터 할당하기
CONN SYSTEM/MANAGER
SHOW USER

ALTER USER USER01
QUOTA 2M ON USERS;

-- CREATE TABLE 생성하기
CONN USER01/TIGER;
SHOW USER;

DROP TABLE EMP01;
CREATE TABLE EMP01(
EMPNO NUMBER(4),
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
DEPTNO NUMBER(2)
);
```

17.3.2 WITH ADMIN OPTION

- 사용자에게 시스템 권한을 WITH ADMIN OPTION과 함께 부여하면 그 사용자는 데이터베이스 관리자가 아닌데도 불구하고 부여받은 시스템 권한을 다른 사용자에게 부여할 수 있는 권한도 함께 부여 받게 된다.

```
-- WITH ADMIN OPTION을 지정하여 권한 부여하기
CONN SYSTEM/MANAGER
SHOW USER

CREATE USER USER02 IDENTIFIED BY TIGER;
GRANT CREATE SESSION TO USER02 WITH ADMIN OPTION;

CONN USER02/TIGER;
SHOW USER;

GRANT CREATE SESSION TO USER01;

-- WITH ADMIN OPTION을 지정하지 않고 권한 부여하기
CONN SYSTEM/MANAGER
SHOW USER

CREATE USER USER03 IDENTIFIED BY TIGER;
GRANT CREATE SESSION TO USER03;

CONN USER03/TIGER;
SHOW USER;

GRANT CREATE SESSION TO USER01;
-- 자기가 받은 권한을 다른 사용자에게 부여할 수 없다.
```

17.4 객체 권한

17.4.1 객체와 권한 설정

- 객체 권한은 특정 객체에 조작을 할 수 있는 권한입니다. 객체의 소유자는 객체에 대한 모든 권한을 가진다.
- 객체 권한은 테이블이나 뷰나 시퀀스나 함수 등과 같은 객체별로 DML문(SELECT, INSERT, DELETE)을 사용할 수 있는 권한을 설정하는 것이다.

```
-- 형식
GRANT PRIVILEGE_NAME [(COLUMN_NAME)] | ALL
ON OBJECT_NAME | ROLE_NAME | PUBLIC
TO USER_NAME;
```

17.4.2 다른 유저의 객체 접근하기

```
CONN USER01/TIGER;
SHOW USER;
SELECT * FROM EMP; -- 조회할 수 없다.
```

```
-- 테이블 객체에 대한 SELECT 권한 부여하기
CONN SCOTT/TIGER;
SHOW USER;
GRANT SELECT ON EMP TO USER01;

CONN USER01/TIGER;
SHOW USER;
SELECT * FROM SCOTT.EMP;
```

17.4.3 스키마 알아보기

- 스키마(SCHEMA)란 객체를 소유한 사용자명을 의미한다. 객체 명 앞에 소속 사용자명을 기술한다.
- 자신이 소유한 객체를 언급할 때 객체 명 앞에 스키마를 생략할 수 있다.

```
SELECT * FROM SCOTT.EMP;
SELECT * FROM EMP;
```

17.4.4 사용자에게 부여된 권한 조회

- 사용자 권한과 관련된 데이터 디렉터리 중에서 USER_TAB_PRIVS_MADE 데이터 디렉터리는 현재 사용자가 다른 사용자에게 부여한 권한 정보를 알려준다.
- 만일 자신에게 부여된 사용자 권한을 알고 싶을 때에는 USER_TAB_PRIVS_RECD 데이터 디렉터리를 조회하면 된다.

```
CONN SCOTT/TIGER
SELECT * FROM USER_TAB_PRIVS_MADE;
SELECT * FROM USER_TAB_PRIVS_RECD;
```

17.4.5 사용자에게서 권한을 뺏기 위한 REVOKE 명령어

- 사용자에게 부여한 객체 권한을 데이터베이스 관리자나 객체 소유자로부터 철회하기 위해서는 REVOKE 명령어를 사용한다.

```
-- 형식
REVOKE {privilege_name | all}
ON object_name
FROM {user_name | role_name | public};

-- 예: 객체 권한 제거하기
SELECT * FROM USER_TAB_PRIVS_MADE;
REVOKE SELECT ON EMP FROM USER01;
SELECT * FROM USER_TAB_PRIVS_MADE;

CONN USER01/TIGER
SELECT * FROM SCOTT.EMP;
```

17.4.6 WITH GRANT OPTION

- 사용자에게 객체 권한을 WITH GRANT OPTION과 함께 부여하면 그 사용자는 그 객체를 접근할 권한을 부여 받으면서 그 권한을 다른 사용자에게 부여 할 수 있는 권한도 함께 부여받게 된다.

```
GRANT SELECT ON SCOTT.EMP TO USER02  
WITH GRANT OPTION;  
  
CONN USER02/TIGER;  
GRANT SELECT ON SCOTT.EMP TO USER01;
```

[과제] 과제-17-01.TXT

1. 전산실에 새로 입사한 직원에게 새로운 계정을 생성해 주려고 합니다.아래의 요구 사항을 만족하는 SQL문을 각각 작성 하세요 ?

[요구1] USER명 : woman, 패스워드 : tiger

[요구2] CREATE SESSION 이라는 시스템 권한을 부여해 줍니다.

(단, 또 다른 유저에게도 권한을 줄 수 있도록 WITH ADMIN OPTION을 부여합니다).

<정답>

2. user01 계정(비밀번호: tiger)을 생성하고 해당 계정에게 오라클 데이터 베이스에 접속해서, 테이블을 생성할 수 있는 권한을 부여하시오.

<정답>

18 데이터베이스 롤 권한 제어

18.1 롤의 정의와 종류

18.1.1 롤의 정의

- 롤은 사용자에게 보다 효율적으로 권한을 부여할 수 있도록 여러 개의 권한을 묶어 놓은 것이라고 생각하면 된다.
- 사용자를 생성했으면 그 사용자에게 각종 권한을 부여해야만 생성된 사용자가 데이터베이스를 사용할 수 있다.
- 데이터베이스의 접속 권한(CREATE SESSION), 테이블 생성 권한(CREATE TABLE), 테이블 수정(UPDATE), 삭제(DELETE), 조회(SELECT) 등과 같은 권한은 사용자에게 기본적으로 필요한 권한들인데 사용자를 생성할 때마다 일일이 이러한 권한을 부여하는 것은 번거롭다.
- 이 때문에 다수의 사용자에게 공통적으로 필요한 권한들을 롤에 하나의 그룹으로 묶어두고 사용자에게는 특정 롤에 대한 권한 부여를 함으로서 간단하게 권한 부여를 할 수 있도록 한다.
- 또한 여러 사용자에게 부여된 권한을 수정하고 싶을 때에도 일일이 사용자마다 권한을 수정하지 않고 롤만 수정하면 그 롤에 대한 권한 부여를 한 사용자들의 권한이 자동 수정된다. 이 밖에 롤을 활성화 비활성화 함으로서 일시적으로 권한을 부여했다 철회할 수 있으므로 사용자 관리를 간편하고 효율적으로 할 수 있다.

18.1.2 사전에 정의된 롤의 종류

- 롤은 오라클 데이터베이스를 설치하면 기본적으로 제공되는 사전 정의된 롤과 사용자가 정의한 롤로 구분된다. 다음과 사전 정의된 시스템에서 제공해주는 롤이다.
- CONNECT 롤
 - 사용자가 데이터베이스에 접속 가능하도록 하기 위해서 다음과 같이 가장 기본적인 시스템 권한 8가지를 묶어 놓았다.
 - ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW
- RESOURCE 롤
 - 사용자가 객체(테이블, 뷰, 인덱스)를 생성할 수 있도록 하기 위해서 시스템 권한을 묶어 놓았다.
 - CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER
- DBA 롤
 - 사용자들이 소유한 데이터베이스객체를 관리하고 사용자들을 작성하고 변경하고 제거할 수 있도록 하는 모든 권한을 가진다. 즉, 시스템 자원을 무제한적으로 사용하며 시스템 관리에 필요한 모든 권한을 부여할 수 있는 강력한 권한을 보유한 롤이다.

[실습] 롤 부여하기

- 일반적으로 데이터베이스 관리자는 새로운 사용자를 생성할 때 CONNECT롤과 RESOURCE롤을 부여한다. USER04 사용자를 생성하여 CONNECT롤과 RESOURCE롤을 부여해 보겠다.

[LAB_18_1.SQL] 롤 부여하기

```
01 CONN SYSTEM/MANAGER
02 DROP USER USER04 CASCADE;
03 CREATE USER USER04 IDENTIFIED BY TIGER;
04 CONN USER04/TIGER
```

```

05
06 CONN SYSTEM/MANAGER
07 GRANT CONNECT, RESOURCE TO USER04;
08 CONN USER04/TIGER

```

18.1.3 롤 관련 데이터 디렉터리

- 데이터를 디렉터리를 통해서 부여된 권한에 대한 정보를 확인할 수 있다.

```
SELECT * FROM DICT WHERE TABLE_NAME LIKE '%ROLE%';
```

```

CONN USER04/TIGER
SELECT * FROM USER_ROLE_PRIVS;

```

디렉터리 명	설 명
ORLE_SYS_PRIVS	롤에 부여된 시스템 권한 정보
ROLE_TAB_PRIVS	롤에 부여된 테이블 관련 권한 정보
USER_ROLE_PRIVS	접근 가능한 롤 정보
USER_TAB_PRIVS_MADE	해당 사용자 소유의 오브젝트에 대한 오브젝트 권한 정보
USER_TAB_PRIVS_RECD	사용자에게 부여된 오브젝트 권한 정보
USER_COL_PRIVS_MADE	사용자 소유의 오브젝트 중 칼럼에 부여된 오브젝트 권한 정보
USER_COL_PRIVS_REDC	사용자에게 부여된 특정 칼럼에 대한 오브젝트 권한 정보

18.2 사용자 롤 정의

- 사용자는 CREATE ROLE 명령어로 다음 형식에 따라 롤을 생성해야 한다.

```

-- 형식
CREATE ROLE ROLE_NAME;
GRANT PRIVILEGE_NAME TO ROLE_NAME;

```

[실습] 롤 생성하여 시스템 권한 할당하기

- 롤을 생성하여 할당하는 시스템 권한을 할당해 보겠다.

```

CONN SYSTEM/MANAGER
CREATE ROLE MROLE;

GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW TO MROLE;

DROP USER USER05 CASCADE;
CREATE USER USER05 IDENTIFIED BY TIGER;
GRANT MROLE TO USER05;

CONN USER05/TIGER
SELECT * FROM USER_ROLE_PRIVS;

```

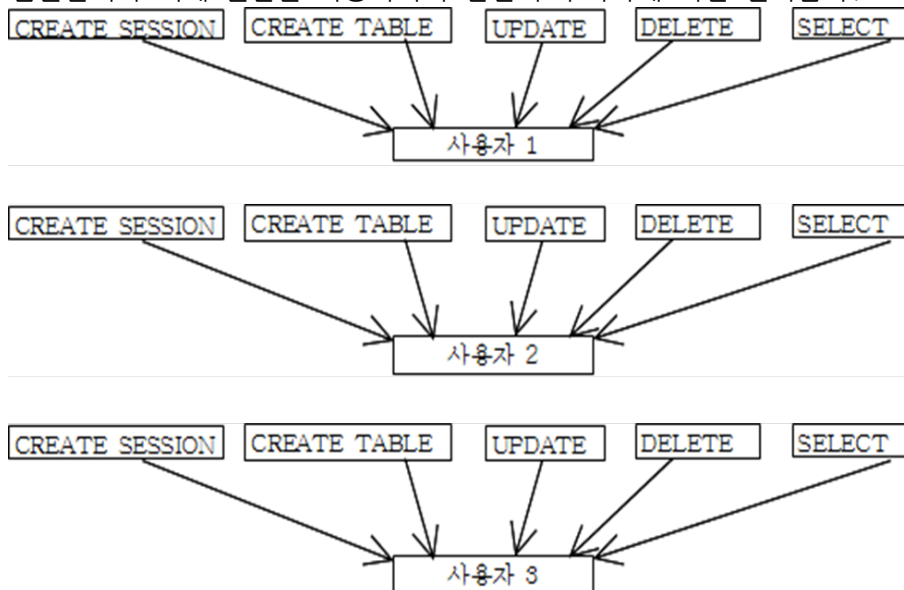
18.3 롤 회수하기

- 롤을 회수하기 위해 DROP ROLE 명령어를 사용한다.

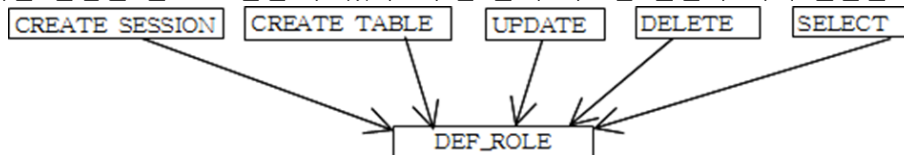
```
-- 형식  
REVOKE ROLE_NAME FROM USER_NAME;
```

18.4 롤의 장점

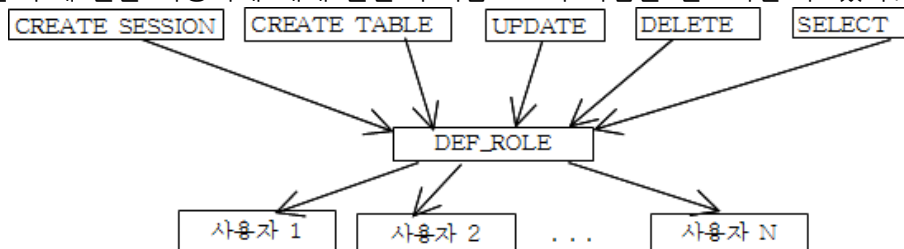
- 시스템권한이나 객체 권한을 사용자마다 일일이 부여하게 되면 번거롭다.



- 이러한 단점을 롤로 보완할 수 있다. 우선 롤에 시스템 권한과 객체 권한을 부여한다.



- 그런 후에 롤을 사용자에게 권한 부여함으로써 작업을 간소화할 수 있다.



[실습] 디폴트 롤을 생성하여 여러 사용자에게 부여하기

- 롤에 시스템 권한과 객체 권한을 부여한 후에 사용자에게 롤에 대한 권한을 부여하여 작업을 간소화해 보겠다.

```
CONN SYSTEM/MANAGER
CREATE ROLE DEF_ROLE;

GRANT CREATE SESSION TO DEF_ROLE;
GRANT CREATE TABLE TO DEF_ROLE;

CONN SCOTT/TIGER
GRANT UPDATE ON EMP TO DEF_ROLE;
GRANT DELETE ON EMP TO DEF_ROLE;
GRANT SELECT ON EMP TO DEF_ROLE;

CONN SYSTEM/MANAGER
CREATE USER USERA1 IDENTIFIED BY A1234;
CREATE USER USERA2 IDENTIFIED BY A1234;
CREATE USER USERA3 IDENTIFIED BY A1234;

SHOW USER
GRANT DEF_ROLE TO USERA1;
GRANT DEF_ROLE TO USERA2;
GRANT DEF_ROLE TO USERA3;

SHOW USER
SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE='DEF_ROLE';
SELECT * FROM ROLE_TAB_PRIVS WHERE ROLE='DEF_ROLE';
```

19 동의어

19.1 동의어의 개념과 종류

- 데이터베이스의 객체에 대한 소유권은 해당 객체를 생성한 사용자에게 있다. 따라서 다른 사용자가 객체에 접근하기 위해서는 소유자로부터 접근 권한을 부여받아야 한다. 또한 다른 사용자가 소유한 객체에 접근하기 위해서는 소유자의 이름을 객체 앞에 지정해야 한다.
- 이렇게 객체를 조회할 때마다 일일이 객체의 소유자를 지정하는 것이 번거로울 경우 동의어를 정의하면 긴 이름대신 간단한 이름으로 접근할 수 있게 된다.
- 동의어는 개별 사용자를 대상으로 하는 비공개 동의어와 전체 사용자를 대상으로 한 공개 동의어가 있다.
- 비공개 동의어
 - 객체에 대한 접근 권한을 부여받은 사용자가 정의한 동의어로 해당 사용자만 사용할 수 있다.
- 공개 동의어
 - 권한을 주는 사용자가 정의한 동의어로 누구나 사용할 수 있다. 공개 동의어는 DBA 권한을 가진 사용자만이 생성할 수 있다. SYNONYM 앞에 PUBLIC를 붙여서 정의한다.

```
-- 다른 사용자가 객체에 접근하려면 소유자의 이름을 객체 앞에 지정해야 한다.  
SELECT * FROM SCOTT.EMP;
```

```
-- DUAL이 공개 동의어로 지정되어 있다.  
SELECT * FROM SYS.DUAL;  
SELECT * FROM DUAL;
```

19.2 동의어 생성 및 제거

- synonym_name 은 user_name.object_name 에 대한 별칭이다.
- user_name 은 객체를 소유한 오라클 사용자이고 object_name 는 동의어를 만들려는 데이터베이스 객체 이름이다.

```
-- 형식  
CREATE [PUBLIC] SYNONYM synonym_name  
FOR user_name.object_name;
```

[실습] 테이블 생성 후 객체 권한 부여하기

- SYSTEM 사용자 계정으로 접속해서 테이블을 생성한 후 이를 SCOTT 사용자가 사용할 수 있도록 권한을 부여해 본다.

```
CONN system/manager  
SHOW USER  
  
DROP TABLE SYSTBL;  
CREATE TABLE SYSTBL (  
  ENAME VARCHAR2(20)  
);  
  
INSERT INTO SYSTBL VALUES('전수빈');
```

```
INSERT INTO SYSTBL VALUES('전원지');
```

```
GRANT SELECT ON SYSTBL TO SCOTT;
```

```
CONN scott/tiger  
SELECT * FROM SYSTBL;
```

```
SELECT * FROM system.SYSTBL;
```

[실습] 동의어 생성하기

```
CONN SCOTT/TIGER  
CREATE SYNONYM PRISYSTBL FOR SYSTEM.SYSTBL;  
SELECT * FROM PRISYSTBL;
```

```
CONN SYSTEM/MANAGER  
GRANT CREATE SYNONYM TO SCOTT;
```

```
CONN SCOTT/TIGER  
CREATE SYNONYM PRISYSTBL FOR SYSTEM.SYSTBL;  
SELECT * FROM PRISYSTBL;
```

20 PL/SQL 기초

20.1 PL/SQL의 구조

- PL/SQL 은 Oracle's Procedural Language extension to SQL의 약자이다. SQL문장에서 변수정의, 조건처리(IF), 반복처리(LOOP, WHILE, FOR)등을 지원하며, 오라클 자체에 내장되어 있는 절차적 언어(Procedure Language)로서 SQL의 단점을 보완해준다.
- PL/SQL은 SQL에 없는 다음과 같은 기능이 제공된다.
 - 변수 선언을 할 수 있다.
 - 비교 처리를 할 수 있다.
 - 반복 처리를 할 수 있다.
- PL/SQL은 블록(BLOCK) 구조의 언어로서 크게 3 부분으로 나눌 수 있다.
 - 선언부(DECLARE SECTION): PL/SQL에서 사용하는 모든 변수나 상수를 선언하는 부분으로서 DECLARE로 시작한다.
 - 실행부(EXECUTABLE SECTION): 절차적 형식으로 SQL문을 실행할 수 있도록 절차적 언어의 요소인 제어문, 반복문, 함수 정의 등 로직을 기술할 수 있는 부분으로 BEGIN으로 시작한다.
 - 예외 처리(EXCEPTION SECTION): PL/SQL 문이 실행되는 중에 예러가 발생할 수 있는데 이를 예외 사항이라고 한다. 이러한 예외 사항이 발생했을 때 이를 해결하기 위한 문장을 기술할 수 있는 부분으로 EXCEPTION 으로 시작한다.
- PL/SQL 프로그램의 작성 요령은 다음과 같다.
 - PL/SQL 블록내에서는 한 문장이 종료할 때마다 세미콜론(;)을 사용한다.
 - END뒤에 ;을 사용하여 하나의 블록이 끝났다는 것을 명시한다.
 - PL/SQL 블록의 작성은 편집기를 통해 파일로 작성할 수도 있고, 프롬프트에서 바로 작성할 수도 있다.
 - SQL*PLUS환경에서는 DELCLARE나 BEGIN이라는 키워드로 PL/SQL블럭이 시작하는 것을 알 수 있다.
 - 단일행 주석은 --이고 여러행 주석 /* */이다.
 - 쿼리문을 수행하기 위해서 /가 반드시 입력되어야 하며, PL/SQL 블록은 행에 / 가 있으면 종결된 것으로 간주한다.

```
-- 형식
DECLARE
    변수 선언;
BEGIN
    실행문;
END;
/

-- 예
SET SERVEROUTPUT ON -- 출력해 주는 내용을 화면에 보여주도록 설정한다.
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
/
```

20.2 변수 선언과 대입문

- PL/SQL의 선언부에서는 실행부에서 사용할 변수를 선언한다. 변수를 선언할 때 변수명 다음에 자료형을 기술해야 한다.

- PL/SQL에서 변수 선언할 때 사용되는 자료형은 SQL에서 사용하던 자료형과 거의 유사하다.

```
-- 형식
identifier [CONSTANT] datatype [NOT NULL]
[:= | DEFAULT expression];

-- identifier: 변수의 이름
-- CONSTANT: 변수의 값을 변경할 수 없도록 제약한다.
-- datatype: 자료형을 기술한다.
-- NOT NULL: 값을 반드시 포함하도록 하기 위해 변수를 제약한다.
-- Expression: Literal, 다른 변수, 연산자나 함수를 포함하는 표현식

-- 예
VEMPNO NUMBER(4);
VENAME VARCHAR2(10);
```

20.2.1 대입문으로 변수에 값 지정하기

- PL/SQL에서는 변수의 값을 지정하거나 재지정하기 위해서 :=를 사용한다. := 의 좌측에 새 값을 받기 위한 변수를 기술하고 우측에 저장할 값을 기술한다.

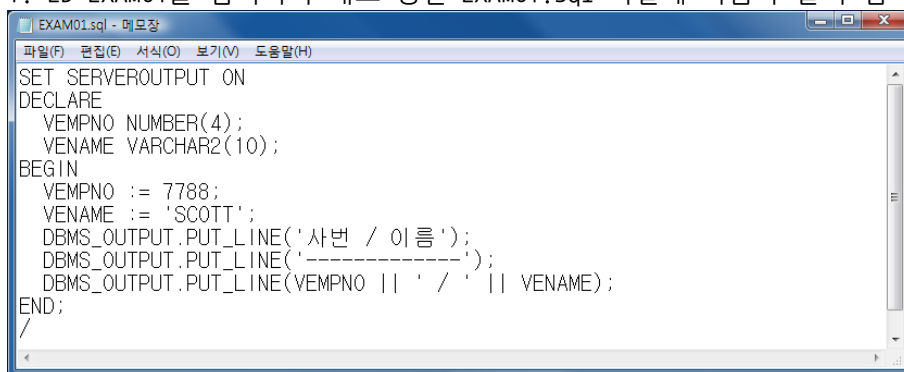
```
-- 형식
identifier := expression;

-- 예
VEMPNO := 7788;
VENAME := 'SCOTT';
```

[실습] 변수 사용하기

- 변수의 선언 및 할당을 하고 그 변수 값을 출력해 본다.

1. ED EXAM01을 입력하여 새로 생긴 EXAM01.sql 파일에 다음과 같이 입력하라.



```
SET SERVEROUTPUT ON
DECLARE
  VEMPNO NUMBER(4);
  VENAME VARCHAR2(10);
BEGIN
  VEMPNO := 7788;
  VENAME := 'SCOTT';
  DBMS_OUTPUT.PUT_LINE('사번 / 이름');
  DBMS_OUTPUT.PUT_LINE('-----');
  DBMS_OUTPUT.PUT_LINE(VEMPNO || ' / ' || VENAME);
END;
/
```

2. 작성이 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @EXAM01을 입력하면 EXAM01.sql 파일 내부에 기술한 PL/SQL 이 실행된 후 결과가 출력된다.


```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> ED EXAM01
SQL> @EXAM01
사번 / 이름
-----
7788 / SCOTT

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>
```

20.2.2 스칼라 변수/레퍼런스 변수

- PL/SQL 에서 변수를 선언하기 위해 사용할 수 있는 데이터형은 크게 스칼라(Scalar)와 레퍼런스(Reference)로 나눌 수 있다.
- 스칼라: PL/SQL 에서 변수를 선언할 때 사용되는 자료형은 SQL 에서 사용하던 자료형과 거의 유사하다. 숫자를 저장하려면 NUMBER 를 사용하고 문자를 저장하려면 VARCHAR2 를 사용해서 선언한다.

```
VEMPNO NUMBER(4);
VENAME VARCHAR2(10);
```

- 레퍼런스: 이전에 선언된 다른 변수 또는 데이터베이스 컬럼에 맞추어 변수를 선언하기 위해 %TYPE 속성을 사용할 수 있다.

```
VEMPNO EMP.EMPNO%TYPE;
VENAME EMP.ENAME%TYPE;
```

20.2.3 PL/SQL에서 SELECT INTO문

- 데이터베이스에서 정보를 추출할 필요가 있을 때 또는 데이터베이스로 변경된 내용을 적용할 필요가 있을 때 SQL 을 사용한다.
- PL/SQL 은 SQL 에 있는 DML 명령을 지원한다. 테이블의 행에서 질의된 값을 변수에 할당시키기 위해 SELECT 문장을 사용한다.
- PL/SQL 의 SELECT 문은 INTO 절이 필요한데, INTO 절에는 데이터를 저장할 변수를 기술한다.
- SELECT 절에 있는 컬럼은 INTO 절에 있는 변수와 1 대 1 대응을 하기에 개수와 데이터의 형, 길이가 일치하여야 한다.
- SELECT 문은 INTO 절에 의해 하나의 행만을 저장할 수 있다.

```
-- 형식
SELECT select_list
INTO {variable_name1[, variable_name2,...] | record_name}
FROM table_name
WHERE condition;

-- 예
SELECT EMPNO, ENAME INTO VEMPNO, VENAME
FROM EMP
WHERE ENAME='SCOTT';
```

[실습] 사번과 이름 검색하기

■ PL/SQL의 SELECT 문으로 EMP 테이블에서 사원번호와 이름을 조회한다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하라.

[EXAM02.SQL]

```
01 SET SERVEROUTPUT ON
02 DECLARE
03 -- %TYPE 속성으로 컬럼 단위 레퍼런스 변수 선언
04 VEMPNO EMP.EMPNO%TYPE;
05 VENAME EMP.ENAME%TYPE;
06 BEGIN
07 DBMS_OUTPUT.PUT_LINE('사번 / 이름');
08 DBMS_OUTPUT.PUT_LINE('-----');
09
10 SELECT EMPNO, ENAME INTO VEMPNO, VENAME
11 FROM EMP
12 WHERE ENAME='SCOTT';
13
14 -- 레퍼런스 변수에 저장된 값을 출력한다.
15 DBMS_OUTPUT.PUT_LINE(VEMPNO || ' / ' || VENAME);
16 END;
17 /
```

2. 작성이 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL 이 실행된 후 결과가 출력된다.

C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM02.SQL

SQL> @EXAM02.SQL

사번 / 이름

7788 / SCOTT

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> quit

20.2.4 PL/SQL 테이블 TYPE

- PL/SQL 테이블은 로우에 대해 배열처럼 액세스하기 위해 기본키를 사용한다.
- 배열과 유사하고 PL/SQL 테이블을 액세스하기 위해 BINARY_INTEGER 데이터형의 기본키와 PL/SQL 테이블 요소를 저장하는 스칼라 또는 레코드 데이터형의 컬럼을 포함해야 한다.

Primary key	Column
.....
1	SMITH
2	ALLEN
3	WARD
.....
BINARY_INTEGER	스칼라

```
TYPE table_type_name IS TABLE OF
{column_type | variable%TYPE | table.column%TYPE} [NOT NULL]
[INDEX BY BINARY_INTEGER];
identifier table_type_name;
```

[실습] TABLE 변수 사용하기

- TABLE 변수를 사용하여 EMP 테이블에서 이름과 업무를 출력해 본다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하라.

[EXAM03.SQL]

```
01 SET SERVEROUTPUT ON
02 DECLARE
03   -- 테이블 타입을 정의
04   TYPE ENAME_TABLE_TYPE IS TABLE OF EMP.ENAME%TYPE
05     INDEX BY BINARY_INTEGER;
06   TYPE JOB_TABLE_TYPE IS TABLE OF EMP.JOB%TYPE
07     INDEX BY BINARY_INTEGER;
08
09   -- 테이블 타입으로 변수 선언
10   ENAME_TABLE ENAME_TABLE_TYPE;
11   JOB_TABLE JOB_TABLE_TYPE;
12
13   I BINARY_INTEGER := 0;
14
15 BEGIN
16   -- EMP 테이블에서 사원이름과 직급을 얻어옴
17   FOR K IN (SELECT ENAME, JOB FROM EMP) LOOP
18     I := I + 1;           --인덱스 증가
19     ENAME_TABLE(I) := K.ENAME; --사원이름과
20     JOB_TABLE(I) := K.JOB;   --직급을 저장.
21   END LOOP;
22
23   --테이블에 저장된 내용을 출력
24   FOR J IN 1..I LOOP
25     DBMS_OUTPUT.PUT_LINE(RPAD(ENAME_TABLE(J),12)
26       || ' / ' || RPAD(JOB_TABLE(J),9));
27   END LOOP;
28 END;
29 /
```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에

기술한 PL/SQL이 실행된 후 결과가 출력된다.

```
C:\Temp>SQLPLUS SCOTT/TIGER
```

```
SQL> ed EXAM03.SQL
```

```
SQL> @EXAM03.SQL
```

```
SMITH      / CLERK
ALLEN      / SALESMAN
WARD       / SALESMAN
JONES      / MANAGER
MARTIN     / SALESMAN
BLAKE      / MANAGER
CLARK      / MANAGER
SCOTT      / ANALYST
KING       / PRESIDENT
TURNER     / SALESMAN
ADAMS      / CLERK
JAMES      / CLERK
FORD       / ANALYST
MILLER     / CLERK
```

PL/SQL 처리가 정상적으로 완료되었습니다.

```
SQL>
```

20.2.5 PL/SQL RECORD TYPE

- PL/SQL RECORD TYPE은 프로그램 언어의 구조체와 유사하다.
- PL/SQL RECORD는 FIELD(ITEM)들의 집합을 하나의 논리적 단위로 처리할 수 있게 해주므로 테이블의 ROW를 읽어올 때 편리하다.

[실습] RECORD TYPE 사용하기

- EMP 테이블에서 SCOTT 사원의 정보를 출력해 본다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하라.

```
[EXAM04.SQL]
```

```
01  SET SERVEROUTPUT ON
02  DECLARE
03      -- 레코드 타입을 정의
04      TYPE emp_record_type IS RECORD(
05          v_empno    emp.empno%TYPE,
06          v_ename    emp.ename%TYPE,
07          v_job      emp.job%TYPE,
08          v_deptno   emp.deptno%TYPE);
09
10      -- 레코드로 변수 선언
11      emp_record    emp_record_type;
12  BEGIN
13      -- SCOTT 사원의 정보를 레코드 변수에 저장
14      SELECT empno,ename, job, deptno
15          INTO emp_record
16          FROM emp
17          WHERE ename = UPPER('SCOTT');
```

```

18
19  -- 레코드 변수에 저장된 사원 정보를 출력
20  DBMS_OUTPUT.PUT_LINE('사원번호 : ' || TO_CHAR(emp_record.v_empno));
21  DBMS_OUTPUT.PUT_LINE('이      림: ' || emp_record.v_ename);
22  DBMS_OUTPUT.PUT_LINE('담당업무 : ' || emp_record.v_job);
23  DBMS_OUTPUT.PUT_LINE('부서번호 : ' || TO_CHAR(emp_record.v_deptno));
24  END;
25  /

```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```

C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM04.SQL

SQL> @EXAM04.SQL
사원번호 : 7788
이      림: SCOTT
담당업무 : ANALYST
부서번호 : 20

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>

```

20.3 PL/SQL의 제어문

- PL/SQL의 제어문은 어떤 조건에서 어떤 코드가 실행되어야 하는지를 제어하기 위한 문법으로, 절차적 언어의 구성요소를 포함함.

구문	의미	문법
BEGIN-END	<ul style="list-style-type: none"> • PL/SQL 문을 블록화시킴 • 중첩 가능 	BEGIN { SQL 문 } END
IF-ELSE	<ul style="list-style-type: none"> • 조건의 검사 결과에 따라 문장을 선택적으로 수행 	IF <조건> SQL 문 [ELSE SQL 문] END IF;
FOR	<ul style="list-style-type: none"> • counter 값이 범위 내에 있을 경우 FOR 문의 블록을 실행 	FOR counter IN <범위> {SQL 문} END LOOP
WHILE	<ul style="list-style-type: none"> • 조건이 참일 경우 WHILE 문의 블록을 실행 	WHILE <조건> { SQL 문 BREAK CONTINUE } END LOOP
RETURN	<ul style="list-style-type: none"> • 프로시저를 종료 • 상태값을 정수로 반환 가능 	RETURN [<정수>]

20.3.1 IF~THEN~END IF

```
IF condition THEN ..... 조건문
statements; ..... 조건에 만족할 경우 실행되는 문장
END IF
```

[실습] 부서 번호로 부서명 알아내기

- 다음은 사원 번호가 7788인 사원의 부서 번호를 얻어 와서 부서 번호에 따른 부서명을 구하는 예제이다. IF문이 끝났을 때에는 반드시 END IF를 기술해야 한다는 점에 주의해야 한다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하라.

[EXAM05.SQL]

```
01  SET SERVEROUTPUT ON
02  DECLARE
03      VEMPNO          NUMBER(4);
04      VENAME          VARCHAR2(20);
05      VDEPTNO         EMP.DEPTNO%TYPE;
06      VDNAME          VARCHAR2(20) := NULL;
07  BEGIN
08      SELECT EMPNO, ENAME, DEPTNO INTO VEMPNO, VENAME, VDEPTNO
09      FROM EMP
10      WHERE EMPNO=7788;
11
12      IF (VDEPTNO = 10) THEN
13          VDNAME := 'ACCOUNTING';
14      END IF;
15      IF (VDEPTNO = 20) THEN
16          VDNAME := 'RESEARCH';
17      END IF;
18      IF (VDEPTNO = 30) THEN
19          VDNAME := 'SALES';
20      END IF;
21      IF (VDEPTNO = 40) THEN
22          VDNAME := 'OPERATIONS';
23      END IF;
24
25      DBMS_OUTPUT.PUT_LINE('사번   이름   부서명');
26      DBMS_OUTPUT.PUT_LINE(VEMPNO||'   '||VENAME||'   '||VDNAME);
27  END;
28  /
```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```
C:\Temp>SQLPLUS SCOTT/TIGER
```

```
SQL> ed EXAM05.SQL
```

```
SQL> @EXAM05.SQL
```

```
사번   이름   부서명
7788   SCOTT   RESEARCH
```

```
PL/SQL 처리가 정상적으로 완료되었습니다.
```

```
SQL>
```

20.3.2 IF~THEN~ELSE~END IF

```
[문장1]
IF condition THEN ..... 조건문
statements; ..... 조건에 만족할 경우 실행되는 문장[문장2]
ELSE
statements; ..... 조건에 만족하지 않을 경우 실행되는 문장[문장3]
END IF
[문장4]
```

[실습하기] 직원의 연봉 구하기

- 다음은 연봉을 구하는 예제이다. 커미션을 받는 직원은 급여에 12를 곱한 후 커미션과 합산하여 연봉을 구하고 커미션을 받지 않는 직원은 급여에 12를 곱한 것으로만 연봉을 구한다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하라.

```
[EXAM06.SQL]

01  SET SERVEROUTPUT ON
02  DECLARE
03      VEMP EMP%ROWTYPE;
04      ANNSAL NUMBER(7,2);
05  BEGIN
06      -- SCOTT 사원의 전체 정보를 로우 단위로 얻어와 VEMP에 저장한다.
07      SELECT * INTO VEMP
08      FROM EMP
09      WHERE ENAME='SCOTT';
10
11      IF (VEMP.COMM IS NULL) THEN      -- 커미션이 NULL 이면
12          ANNSAL:=VEMP.SAL*12;        -- 급여에 12를 곱한다.
13      ELSE                            -- 커미션이 NULL이 아니면
14          ANNSAL:=VEMP.SAL*12+VEMP.COMM;-- 급여에 12를 곱한 후 커미션과 합산
15      END IF;
16
17      DBMS_OUTPUT.PUT_LINE('사번 / 이름 / 연봉');
18      DBMS_OUTPUT.PUT_LINE('-----');
19      DBMS_OUTPUT.PUT_LINE(VEMP.EMPNO||'/'||VEMP.ENAME||'/'||ANNSAL);
20  END;
21  /
```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```
C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM06.SQL

SQL> @EXAM06.SQL
사번 / 이름 / 연봉
-----
7788/SCOTT/36000

PL/SQL 처리가 정상적으로 완료되었습니다.
```

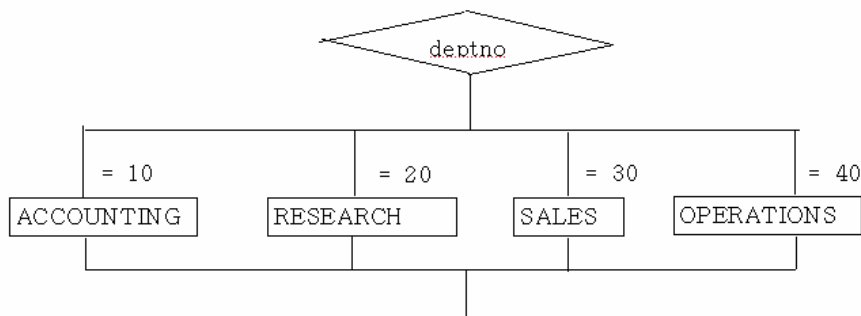
SQL>

20.3.3 IF~THEN~ELSEIF~ELSE~END IF

```
IF condition THEN
statements;
ELSIF condition THEN
statements;
ELSIF condition THEN
statements;
ELSE
statements;
END IF
```

[실습] 부서 번호로 부서명 알아내기

- SQL 함수에서 선택을 위한 DECODE 함수를 학습하면서 부서번호에 대해서 부서명을 지정해 보았다.



- 이곳 PL/SQL에서는 DECODE 함수 대신 IF ~ THEN ~ ELSEIF ~ ELSE ~ END IF 구문으로 부서번호에 대한 부서명을 구해 보자.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하라.

[EXAM07.SQL]

```
01 SET SERVEROUTPUT ON
02 DECLARE
03     VEMP EMP%ROWTYPE;
04     VNAME VARCHAR2(14);
05 BEGIN
06     DBMS_OUTPUT.PUT_LINE('사번 / 이름 / 부서명');
07     DBMS_OUTPUT.PUT_LINE('-----');
08
09     SELECT * INTO VEMP
10     FROM EMP
11     WHERE ENAME='SCOTT';
12
13     IF (VEMP.DEPTNO = 10) THEN
14         VNAME := 'ACCOUNTING';
15     ELSIF (VEMP.DEPTNO = 20) THEN
16         VNAME := 'RESEARCH';
17     ELSIF (VEMP.DEPTNO = 30) THEN
18         VNAME := 'SALES';
19     ELSIF (VEMP.DEPTNO = 40) THEN
20         VNAME := 'OPERATIONS';
```



```

21     END IF;
22
23     DBMS_OUTPUT.PUT_LINE(VEMP.EMPNO||'/'||VEMP.ENAME||'/'||VDNAME);
24     END;
25     /

```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```

C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM07.SQL

SQL> @EXAM07.SQL
사번 / 이름 / 부서명
-----
7788/SCOTT/RESEARCH

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>

```

20.3.4 BASIC LOOP문

- 지금 소개할 구문은 가장 간단한 루프로 구분 문자로 LOOP와 END LOOP가 사용된다.
- 실행 상의 흐름이 END LOOP에 도달할 때마다 그와 짝을 이루는 LOOP 문으로 제어가 되돌아간다.
- 이러한 루프를 무한 루프라 하며, 여기서 빠져나가려면 EXIT문을 사용한다.
- 기본 LOOP는 LOOP에 들어갈 때 조건이 이미 일치했다 할지라도 적어도 한번은 문장이 실행된다.

```

LOOP
statement1;
statement2;
. . . . .
EXIT [WHERE condition];
END LOOP

```

[실습] BASIC LOOP문으로 1부터 5까지 출력하기

- 다음은 BASIC LOOP 문으로 1부터 5까지 출력하는 예제이다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력한다.

```

[EXAM08.SQL]

01  SET SERVEROUTPUT ON
02  DECLARE
03      N  NUMBER := 1;
04  BEGIN
05      LOOP
06          DBMS_OUTPUT.PUT_LINE( N );

```

```

07      N := N + 1;
08      IF N > 5 THEN
09          EXIT;
10      END IF;
11      END LOOP;
12  END;
13  /

```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```

C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM08.SQL

SQL> @EXAM08.SQL
1
2
3
4
5

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>

```

20.3.5 FOR LOOP문

```

-- 형식
FOR index_counter
IN [REVERSE] lower_bound..upper_bound LOOP
    statement1;
    statement2;
    . . . . .
END LOOP

```

[실습] FOR LOOP문으로 1부터 5까지 출력하기

■ 다음은 FOR LOOP 문으로 1부터 5까지 출력하는 예제이다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력한다.

```

[EXAM09.SQL]

01      SET SERVEROUTPUT ON
02      DECLARE
03      BEGIN
04          FOR N IN 1..5 LOOP
05              DBMS_OUTPUT.PUT_LINE( N );
06          END LOOP;
07      END;
08      /

```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```
C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM09.SQL

SQL> @EXAM09.SQL
1
2
3
4
5

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>
```

20.3.6 WHILE LOOP문

```
WHILE condition LOOP
statement1;
statement2;
. . . . .
END LOOP
```

[실습] WHILE LOOP문으로 1부터 5까지 출력하기

■ 다음은 WHILE LOOP 문으로 1부터 5까지 출력하는 예제이다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력한다.

```
[EXAM10.SQL]

01  SET SERVEROUTPUT ON
02  DECLARE
03      N NUMBER := 1;
04  BEGIN
05      WHILE N <= 5 LOOP
06          DBMS_OUTPUT.PUT_LINE( N );
07          N := N + 1;
08      END LOOP;
09  END;
10  /
```

2. 작성을 완료한 후에 파일을 저장한다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL이 실행된 후 결과가 출력된다.

```
C:\Temp>SQLPLUS SCOTT/TIGER

SQL> ed EXAM10.SQL

SQL> @EXAM10.SQL
```

```
1  
2  
3  
4  
5
```

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>