ITNVALUE

SQL 프로그래밍

1

소개

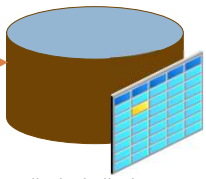
다양한 미디어의 데이터 저장 영역

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping		
4	60	IT		
5	80	Sales		
6	90	Executive		
7	110	Accounting		
8	190	Contracting		

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

전자
스프레드시트

서류함



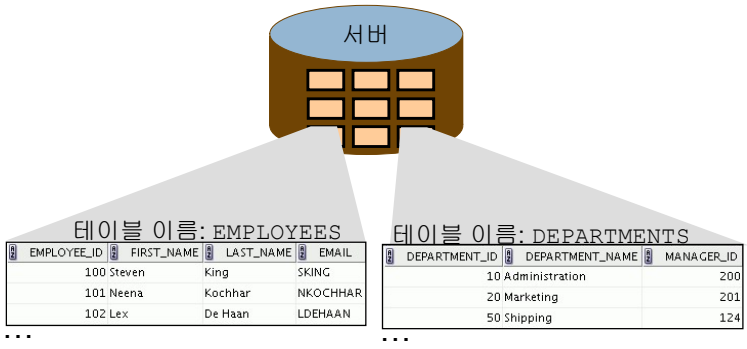
데이터베이스

관계형 데이터베이스 개념

- 데이터베이스 시스템의 관계형 모델은 1970년 E. F. Codd 박사가 처음 제안했습니다.
- 이 모델은 RDBMS(관계형 데이터베이스 관리 시스템)의 기초가 되었습니다.
- 관계형 모델은 다음과 같은 요소로 구성됩니다.
  - 객체 또는 관계 모음
  - 관계에서 실행되는 연산자 집합
  - 정확성 및 일관성을 보장하는 데이터 무결성

관계형 데이터베이스 정의

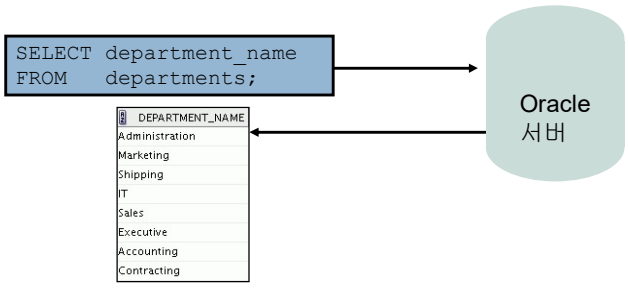
•관계형 데이터베이스는 Oracle 서버에서 제어되는 관계 또는2차원 테이블 모음입니다.



SQL을 사용하여 데이터베이스 query

•SQL(구조적 질의어)은 다음과 같은 특징이 있습니다.

- 관계형 데이터베이스 작동을 위한 ANSI 표준 언어
- 효율적이며 쉽게 배워 사용할 수 있음
- 완벽한 기능(SQL을 사용하면 테이블의 데이터를 정의, 검색 및 조작할 수 있습니다.)



SQL 문

7

2 SQL SELECT 문 작성

8

기본 SELECT 문

- SELECT는 표시할 열을 식별합니다.
- FROM은 이러한 열을 포함한 테이블을 식별합니다.

```
SELECT  *|{[DISTINCT] column [alias],...}
FROM    table;
```


모든 열 선택

```
SELECT *
FROM   departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

특정 열 선택

```
SELECT department_id, location_id
FROM   departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

산술식

•산술 연산자를 사용하여 숫자 및 날짜 데이터로 표현식을 작성합니다.

연산자	설명
+	더하기
-	빼기
*	곱하기
/	나누기

산술 연산자 사용

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...

연산자 우선 순위

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100
4	Hunold	9000	108100

...

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200
4	Hunold	9000	109200

...

날짜 작업

- 오라클 데이터베이스는 내부 숫자 형식(세기, 년, 월, 일, 시, 분, 초)으로 날짜를 저장합니다.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-2008';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-03
2	Kochhar	21-SEP-05

...

날짜 연산

- 날짜에 숫자를 더하거나 빼서 결과 날짜 값을 구합니다.
- 두 날짜 사이의 일 수를 알아내기 위해 빼기 연산을 합니다.
- 시간 수를 24로 나눠 날짜에 시간을 더합니다.

날짜에 산술 연산자 사용

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

	LAST_NAME	WEEKS
1	King	478.871917989417989417989417989417989418
2	Kochhar	360.729060846560846560846560846560846561
3	De Haan	605.300489417989417989417989417989417989

연결 연산자

- 연결 연산자:
  - 열이나 문자열을 다른 열에 연결합니다.
  - 두 개의 세로선(||)으로 나타냅니다.
  - 결과 열로 문자식을 생성합니다.

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP
6	GietzAC_ACCOUNT
7	GrantSA_REP
8	HartsteinMK_MAN

...

리터럴 문자열

- 리터럴은 SELECT 문에 포함된 문자, 숫자 또는 날짜입니다.
- 날짜 및 문자 리터럴 값은 작은따옴표로 묶어야 합니다.
- 각 문자열은 반환되는 각 행에 한 번 출력됩니다.

리터럴 문자열 사용

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
6	Gietz is a AC_ACCOUNT
7	Grant is a SA_REP
8	Hartstein is a MK_MAN
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG
11	King is a AD_PRES

...

Null 값 정의

- Null은 사용할 수 없거나, 할당되지 않았거나, 알 수 없거나, 적용할 수 없는 값입니다.
- Null은 0이나 공백과는 다릅니다.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	De Haan	AD_VP	17000	(null)
...				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15
...				
18	Fay	MK_REP	6000	(null)
19	Higgins	AC_MGR	12008	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

열 alias 정의

- 열 alias:
  - 열 머리글의 이름을 바꿉니다.
  - 계산에서 유용합니다.
  - 열 이름 바로 뒤에 나옵니다. 열 이름과 alias 사이에 선택 사항인 AS 키워드가 올 수도 있습니다.
  - 공백이나 특수 문자를 포함하거나 대소문자를 구분하는 경우 큰따옴표가 필요합니다.

열 alias 사용

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)
4	Hunold	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000
4	Hunold	108000

...

중복 행

•기본적으로 query 결과에는 중복 행을 포함한 모든 행이 표시됩니다.

1

```
SELECT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60
6	60
7	50
8	50

...

2

```
SELECT DISTINCT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110
5	50
6	80
7	60
8	10

선택되는 행 제한

- WHERE 절을 사용하여 반환되는 행을 제한합니다.

```
SELECT  *|{[DISTINCT] column [alias],...}
FROM    table
[WHERE logical expression(s)];
```

- WHERE 절은 FROM 절 다음에 나옵니다.

WHERE 절 사용

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

문자열 및 날짜

- 문자열 및 날짜 값은 작은따옴표로 묶습니다.
- 문자 값은 대소문자를 구분하고 날짜 값은 형식을 구분합니다.
- 기본 날짜 표시 형식은 DD-MON-RR입니다.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen' ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	Whalen	AD_ASST	10

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-OCT-03' ;
```

	LAST_NAME
1	Rajs

비교 연산자

연산자	의미
=	같음
>	보다 큼
>=	보다 크거나 같음
<	보다 작음
<=	보다 작거나 같음
<>	같지 않음
BETWEEN ...AND...	두 값 사이(경계값 포함)
IN(set)	값 리스트 중 일치하는 값 검색
LIKE	일치하는 문자 패턴 검색
IS NULL	null 값인지 여부

비교 연산자 사용

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

BETWEEN 연산자를 사용하는 범위 조건

- BETWEEN 연산자를 사용하여 값의 범위를 기반으로 행을 표시합니다.

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

IN 연산자 사용

•IN 연산자를 사용하여 리스트의 값을 테스트합니다.

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201

LIKE 연산자를 사용하여 패턴 일치

- LIKE 연산자를 사용하여 유효한 검색 문자열 값의 대체 문자 검색을 수행합니다.
- 검색 조건에는 리터럴 문자나 숫자가 포함될 수 있습니다.
  - %는 0개 이상의 문자를 나타냅니다.
  - _은 한 문자를 나타냅니다.

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%' ;
```

	FIRST_NAME
1	Shelley
2	Steven

대체 문자 결합

- 패턴 일치를 위해 두 개의 대체 문자(%, _)를 리터럴 문자와 결합할 수 있습니다.

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

NULL 조건 사용

- IS NULL 연산자로 null을 테스트합니다.

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

	LAST_NAME	MANAGER_ID
1	King	(null)

논리 연산자를 사용하여 조건 정의

연산자	의미
AND	*두* 구성 요소 조건이 모두 참인 경우 TRUE를 반환합니다.
OR	구성 요소 중 *하나*가 참인 경우 TRUE를 반환합니다.
NOT	조건이 거짓인 경우 TRUE를 반환합니다.

AND 연산자 사용

- AND 연산에서는 두 구성 요소 조건이 모두 참이어야 합니다.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
AND    job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

OR 연산자 사용

•OR 연산에서는 두 구성 요소 조건 중 하나가 참이어야 합니다.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12008

NOT 연산자 사용

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

우선 순위 규칙

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  department_id = 60
OR     department_id = 80
AND    salary > 10000;
```

1

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Hunold	60	9000
2	Ernst	60	6000
3	Lorentz	60	4200
4	Zlotkey	80	10500
5	Abel	80	11000

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (department_id = 60
OR     department_id = 80)
AND    salary > 10000;
```

2

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Zlotkey	80	10500
2	Abel	80	11000

ORDER BY 절 사용

- ORDER BY 절을 사용하여 다음과 같이 검색된 행을 정렬합니다.
  - ASC: 오름차순, 기본값
  - DESC: 내림차순

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	De Haan	AD_VP	90	13-JAN-01
2	Gietz	AC_ACCOUNT	110	07-JUN-02
3	Higgins	AC_MGR	110	07-JUN-02
4	King	AD_PRES	90	17-JUN-03
5	Whalen	AD_ASST	10	17-SEP-03
6	Rajs	ST_CLERK	50	17-OCT-03

...

정렬

- 내림차순으로 정렬:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY department_id DESC ;
```

1

- 열 alias를 기준으로 정렬:

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

2

정렬

- 열의 숫자 위치를 사용하여 정렬

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY 3;
```

3

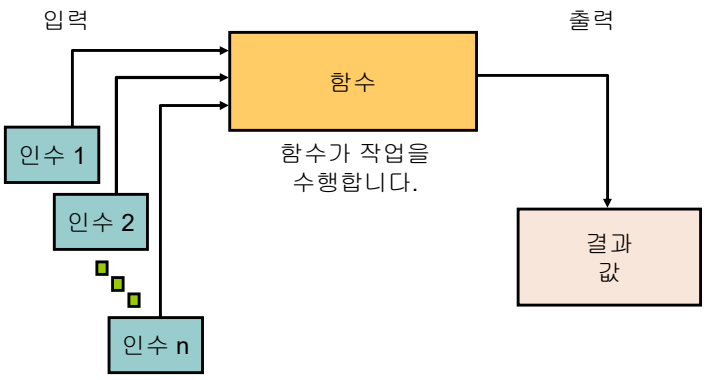
- 여러 열을 기준으로 정렬

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

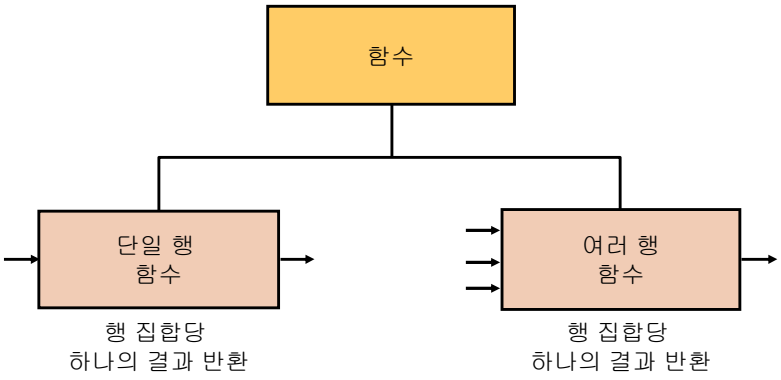
4

3 단일 행 함수 사용

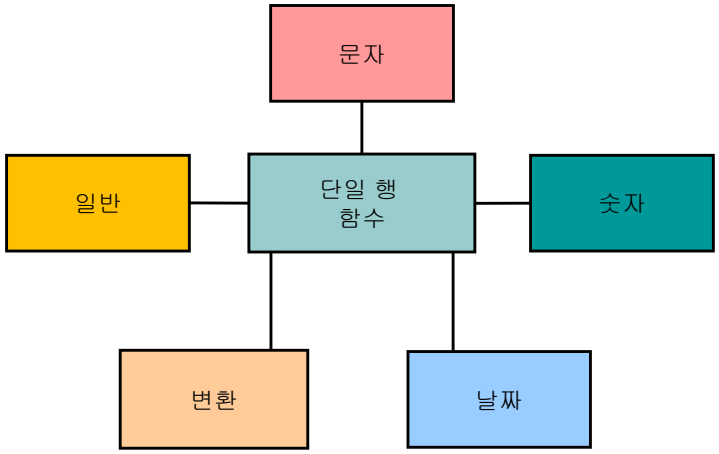
SQL 함수



SQL 함수의 두 가지 유형



단일 행 함수 종류



대소문자 변환 함수

•다음은 문자열의 대소문자를 변환하는 함수입니다.

함수	결과
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

문자 조작 함수

•다음은 문자열을 조작하는 함수입니다.

함수	결과
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6

숫자 함수

- ROUND: 지정된 소수점 자릿수로 값을 반올림합니다.
- TRUNC: 지정된 소수점 자릿수로 값을 truncate합니다.

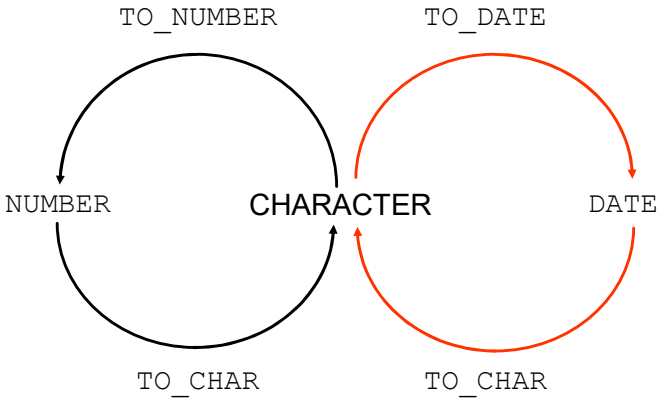
함수	결과
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92

날짜 조작 함수

함수	결과
MONTHS_BETWEEN	두 날짜 간의 월 수
ADD_MONTHS	날짜에 월 추가
LAST_DAY	월의 마지막 날

함수	결과
MONTHS_BETWEEN ('01-SEP-05','11-JAN-04')	19.6774194
ADD_MONTHS ('31-JAN-04',1)	'29-FEB-04'
LAST_DAY ('01-FEB-05')	'28-FEB-05'

명시적 데이터 유형 변환



날짜에 TO_CHAR 함수 사용

```
TO_CHAR(date[,'format_model'])
```

- 형식 모델:
  - 작은따옴표로 묶어야 합니다.
  - 대소문자를 구분합니다.
  - 임의의 유효한 날짜 형식 요소를 포함할 수 있습니다.
  - 쉼표로 날짜 값과 구분됩니다.

날짜 형식 모델의 요소

요소	결과
YYYY	숫자로 된 전체 연도
YEAR	영어 철자로 표기된 연도
MM	월의 2자리 값
MONTH	전체 월 이름
MON	월의 3자 약어
DY	요일의 3자 약어
DAY	요일의 전체 이름
DD	숫자 형식의 월간 일

날짜에 TO_CHAR 함수 사용

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	17 June 2003
2	Kochhar	21 September 2005
3	De Haan	13 January 2001
4	Hunold	3 January 2006
5	Ernst	21 May 2007
6	Lorentz	7 February 2007
7	Mourgos	16 November 2007
8	Rajs	17 October 2003

...

숫자에 TO_CHAR 함수 사용

```
TO_CHAR(number[, 'format_model'])
```

•다음은 TO_CHAR 함수와 함께 사용하여 숫자 값을 문자로 표시할 수 있는 몇 가지 형식 요소입니다.

요소	결과
9	숫자를 나타냄
0	0이 표시되도록 강제 적용
$	부동 달러 기호 배치
L	부동 로컬 통화 기호 사용
.	소수점 출력
,	천 단위 표시자로 쉼표 출력

숫자에 TO_CHAR 함수 사용

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

	SALARY
1	$6,000.00

TO_NUMBER 및 TO_DATE 함수 사용

- TO_NUMBER 함수를 사용하여 문자열을 숫자 형식으로 변환합니다.

```
TO_NUMBER(char[, 'format_model'])
```

- TO_DATE 함수를 사용하여 문자열을 날짜 형식으로 변환합니다.

```
TO_DATE(char[, 'format_model'])
```


일반 함수

•다음 함수는 임의의 데이터 유형을 사용하며 null 사용과 관련이 있습니다.

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)

NVL 함수

- null 인 행을 실제 값으로 변환합니다.
  - 사용할 수 있는 데이터 유형은 날짜, 문자 및 숫자입니다.
  - 데이터 유형이 일치해야 합니다.
    - NVL(commission_pct,0)
    - NVL(hire_date,'01-JAN-97')
    - NVL(job_id,'No Job Yet')

NVL 함수 사용

```
SELECT last_name, salary, NVL(commission_pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200

...

NVL2 함수 사용

```
SELECT last_name, salary, commission_pct,
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

조건부 표현식

- SQL 문에서 IF-THEN-ELSE 논리 사용 가능
- 다음 메소드를 사용:
  - CASE 식
  - 검색된 CASE 표현식
  - DECODE 함수

CASE 식

•IF-THEN-ELSE 문 작업을 수행하여 조건부 조회를 편리하게 수행하도록 합니다.

```
CASE expr WHEN comparison_expr1 THEN return_expr1
         [WHEN comparison_expr2 THEN return_expr2
          WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
END
```


CASE 식 사용

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE       salary END     "REVISED_SALARY"
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
1	King	AD_PRES	24000	24000
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

DECODE 함수

•CASE 식 또는 IF-THEN-ELSE 문의 작업을 수행하여 조건부 조회를 편리하게 수행합니다.

```
DECODE(col|expression, search1, result1
                [, search2, result2,...,]
                [, default])
```


DECODE 함수 사용

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
12	Zlotkey	SA_MAN	10500	10500
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

검색된 CASE 표현식

```
CASE
    WHEN condition1 THEN use_expression1
    WHEN condition2 THEN use_expression2
    WHEN condition3 THEN use_expression3
    ELSE default_use_expression
END
```

```
   SELECT last_name,salary,
   (CASE WHEN salary<5000 THEN 'Low'
         WHEN salary<10000 THEN 'Medium'
         WHEN salary<20000 THEN 'Good'
         ELSE 'Excellent'
   END) qualified_salary
   FROM employees;
```


4 그룹 함수 사용

Group Functions

•그룹 함수는 행 집합 연산을 수행하여 그룹별로 하나의 결과를 산출합니다.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

EMPLOYEES
테이블의 최고 급여

MAX(SALARY)
24000

그룹 함수 유형

- AVG
- COUNT
- MAX
- MIN
- SUM

AVG 및 SUM 함수 사용

•숫자 데이터에 대해 AVG 및 SUM 함수를 사용할 수 있습니다.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

MIN 및 MAX 함수 사용

• 숫자, 문자 및 날짜 데이터 유형에 대해 MIN 및 MAX 함수를 사용할 수 있습니다.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM       employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	13-JAN-01	29-JAN-08

COUNT 함수 사용

•COUNT(*)는 테이블의 행 수를 반환합니다.

1
```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

•COUNT(*expr*)은 *expr*에 대해 null이 아닌 값을 가진 행의 수를 반환합니다.

2
```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(COMMISSION_PCT)
1	0

DISTINCT 키워드 사용

- COUNT(DISTINCT expr)은 *expr*의 null이 아닌 구분 값의 수를 반환합니다.
- 다음은 EMPLOYEES 테이블에서 부서의 구분 값 수를 표시합니다.

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

그룹 함수 및 Null 값

•그룹 함수는 열에 있는 null 값을 무시합니다.

1
```
SELECT AVG(commission_pct)
FROM   employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

•NVL 함수는 강제로 그룹 함수에 null 값이 포함되도록 합니다.

2
```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

데이터 그룹 생성

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

4400
9500
3500
6400
10033

EMPLOYEES 테이블의
각 부서에 대한
평균 급여

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

데이터 그룹 생성: GROUP BY 절 구문

- GROUP BY 절을 사용하여 테이블의 행을 더 작은 그룹으로 나눌 수 있습니다.

```
SELECT    column, group_function(column)
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```


GROUP BY 절 사용

- 그룹 함수에 속하지 않는 SELECT list의 모든 열은 GROUP BY 절에 있어야 합니다.

```
SELECT   department_id, AVG(salary)
FROM     employees
GROUP BY department_id;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.333333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.333333333333333333333333333333333
7	60	6400
8	10	4400

GROUP BY 절 사용

- GROUP BY 열은 SELECT list에 없어도 됩니다.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id;
```

	AVG(SALARY)
1	7000
2	19333.333333333333333333333333333333333
3	9500
4	10154
5	3500
6	10033.333333333333333333333333333333333
7	6400
8	4400

두 개 이상의 열로 그룹화

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

EMPLOYEES 테이블에서 부서별로 그룹화한 각 직무에 대해 급여를 더합니다.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12008
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

다중 열에서 GROUP BY 절 사용

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

그룹 함수를 사용한 잘못된 query

•집계 함수가 아닌 SELECT list의 열이나 표현식은 GROUP BY 절에 있어야 합니다.

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"

그룹 함수를 사용한 잘못된 query

- WHERE 절은 그룹을 제한하는 데 사용할 수 없습니다.
- 그룹을 제한하려면 HAVING 절을 사용합니다.
- WHERE 절에서 그룹 함수를 사용할 수 없습니다.

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```


그룹 결과 제한

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

최고 급여가 $10,000가
넘는각 부서의
최고 급여

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

HAVING 절을 사용하여 그룹 결과 제한

- HAVING 절을 사용할 경우 Oracle 서버는 다음과 같이 그룹을 제한합니다.
  1. 행이 그룹화됩니다.
  2. 그룹 함수가 적용됩니다.
  3. HAVING 절과 일치하는 그룹이 표시됩니다.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```


HAVING 절 사용

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000

HAVING 절 사용

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

그룹 함수 중첩

•최고 평균 급여를 표시합니다.

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333

5 조인문 작성

다중 테이블에서 데이터 가져오기

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	100	Steven	King	AD_PRES
2	101	Neena	Kochhar	AD_VP
3	102	Lex	De Haan	AD_VP
4	103	Alexander	Hunold	IT_PROG
5	104	Bruce	Ernst	IT_PROG
6	105	David	Austin	IT_PROG
7	106	Valli	Pataballa	IT_PROG
8	107	Diana	Lorentz	IT_PROG
9	108	Nancy	Greenberg	FI_MGR
10	109	Daniel	Faviet	FI_ACCOUNT

JOBS

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative

	EMPLOYEE_ID	JOB_ID	JOB_TITLE
1	206	AC_ACCOUNT	Public Accountant
2	205	AC_MGR	Accounting Manager
3	200	AD_ASST	Administration Assistant
4	100	AD_PRES	President
5	101	AD_VP	Administration Vice President
6	102	AD_VP	Administration Vice President
7	109	FI_ACCOUNT	Accountant

...

조인 유형

- SQL:1999 표준과 호환되는 조인에는 다음이 포함됩니다.
  - NATURAL JOIN 절을 사용하여 Natural Join
  - USING 절을 사용하여 조인
  - ON 절을 사용하여 조인
  - OUTER join:
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
  - Cross Join

Cartesian Product

- Cartesian Product는 한 테이블의 모든 행을 다른 테이블의 모든 행과 조인합니다.
- Cartesian Product는 다수의 행을 생성하므로 결과는 그다지 유용하지 않습니다.

Cartesian Product 생성

EMPLOYEES(20행)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS(8행)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian product:
20 x 8 = 160행

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700
...			
21	200	10	1800
22	201	20	1800
...			
159	176	80	1700
160	178	(null)	1700

Cross Join 생성

- CROSS JOIN은 두 테이블의 Cartesian Product를 생성하는 JOIN 작업입니다.
- Cartesian Product를 생성하려면 SELECT 문에 CROSS JOIN을 지정합니다.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

ON 절을 사용하여 조인 생성

- ON 절을 사용하여 임의 조건을 지정하거나 조인할 열을 지정합니다.
- 조인 조건은 다른 검색 조건과는 별개입니다.
- ON 절을 사용하면 코드를 이해하기 쉽습니다.

ON 절을 사용하여 레코드 검색

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400

...

3-Way 조인 생성

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

...

조인에 추가 조건 적용

- AND 절 또는 WHERE 절을 사용하여 추가 조건을 적용합니다.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON    (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

또는

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON    (e.department_id = d.department_id)
WHERE  e.manager_id = 149 ;
```

테이블 자체 조인

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...

WORKER 테이블의 MANAGER_ID는 MANAGER 테이블의 EMPLOYEE_ID와 같습니다.

ON 절을 사용하는 Self-Join

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King

...

Nonequijoin

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000

...

19	Taylor	8600
20	Grant	7000

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

JOB_GRADES 테이블은
각 GRADE_LEVEL의 LOWEST_SAL 및
HIGHEST_SAL 값 범위를 정의합니다.
따라서 GRADE_LEVEL 열을 사용하여
각 사원에 등급을 지정할 수 있습니다.

Nonequijoin을 사용하여 레코드 검색

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

OUTER Join으로 직접 일치되지 않는 레코드 반환

DEPARTMENTS

	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

부서 190에는 사원이 없습니다.

EMPLOYEES와 Equijoin

	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst
...		
18	80	Abel
19	80	Taylor

사원 "Grant"에게는 부서 ID가 할당되지 않았습니다.

INNER Join과 OUTER Join 비교

- SQL:1999에서 일치하는 행만 반환하는 두 테이블의 조인을 INNER join이라고 합니다.
- INNER join의 결과는 물론, 왼쪽(또는 오른쪽) 테이블의 일치하지 않는 행도 반환하는 두 테이블 간의 조인을 Left(또는 Right) OUTER join이라고 합니다.
- INNER join의 결과는 물론, Left 및 Right OUTER join의 결과를 반환하는 두 테이블 간의 조인을 Full outer join이라고 합니다.

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...

15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

6 Subquery

Subquery를 사용하여 문제 해결

•Davies 이후로 채용된 사원은 누구입니까?

Main query:

Davies 이후로 채용된 모든 사원의 이름을 식별하십시오.

Subquery:

Davies가 채용된 시기는 언제입니까?

Subquery 구문

- subquery는 main query*전에* 실행됩니다.
- Subquery 결과는 main query에서 사용됩니다.

```
SELECT   select_list
FROM     table
WHERE    expr operator
             (SELECT   select_list
                 FROM     table);
```

Subquery 사용

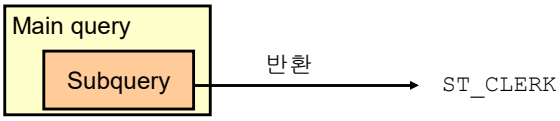
```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date > (SELECT hire_date
                    FROM   employees
                    WHERE  last_name = 'Davies');
```


Subquery 사용 규칙과 지침

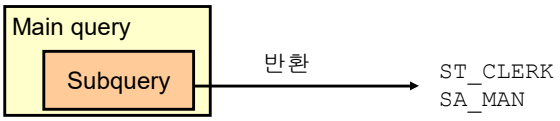
- subquery는 괄호로 묶습니다.
- 가독성을 위해 비교 조건의 오른쪽에 subquery를 배치합니다. 그러나 subquery는 비교 연산자의 양쪽 어디에나 사용할 수 있습니다.
- single-row subquery에는 단일 행 연산자를 사용하고 multiple-row subquery에는 다중 행 연산자를 사용합니다.

Subquery 유형

- Single-row subquery



- Multiple-row subquery



Single-Row Subquery

- 한 행만 반환합니다.
- 단일 행 비교 연산자를 사용합니다.

연산자	의미
=	같음
>	보다 큼
>=	보다 크거나 같음
<	보다 작음
<=	보다 작거나 같음
<>	같지 않음

Single-Row Subquery 실행

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary > (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Taylor');
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

Subquery에서 그룹 함수 사용

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary = (SELECT MIN(salary)
                 FROM   employees);
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

Subquery가 있는 HAVING 절

- Oracle 서버는 subquery를 먼저 실행합니다.
- Oracle 서버는 main query의 HAVING 절로 결과를 반환합니다.

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) > (SELECT MIN(salary)
                        FROM   employees
                        WHERE  department_id = 30);
```

	DEPARTMENT_ID	MIN(SALARY)
1	100	6900
2	(null)	7000
3	90	17000
4	20	6000
5	70	10000
6	110	8300
7	80	6100
8	40	6500
9	60	4200
10	10	4400

이 명령문에서 잘못된 점은?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary = (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
```

```
ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:
```

Multiple-Row Subquery

- 두 개 이상의 행을 반환합니다.
- 다중 행 비교 연산자를 사용합니다.

연산자	의미
IN	리스트의 임의 멤버와 같음
ANY	=, !=, >, <, <=, >= 연산자가 앞에 있어야 합니다. 관계가 TRUE인 subquery의 결과 집합에 요소가 1개 이상 있는 경우 TRUE를 반환합니다.
ALL	=, !=, >, <, <=, >= 연산자가 앞에 있어야 합니다. Subquery 결과 집합의 모든 요소에 대한 관계가 TRUE인 경우 TRUE를 반환합니다.

Multiple-Row subquery에서 ANY 연산자 사용

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

Multiple-Row subquery에서 ALL 연산자 사용

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

Multiple-Column Subquery

- multiple-column subquery는 outer query에 두 개 이상의 열을 반환합니다.
- multiple-column subquery는 SELECT 문의 FROM 절에 사용될 수도 있습니다.

Multiple-Column Subquery: 예제

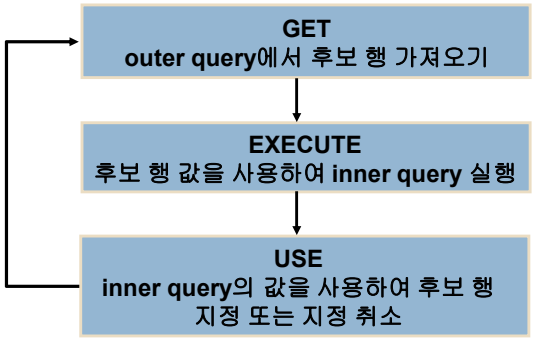
- 각 부서에서 급여가 가장 낮은 모든 사원 표시

```
SELECT first_name, department_id, salary
FROM   employees
WHERE (salary, department_id) IN
      (SELECT min(salary), department_id
       FROM   employees
       GROUP BY department_id)
ORDER BY department_id;
```

	FIRST_NAME	DEPARTMENT_ID	SALARY
1	Jennifer	10	4400
2	Pat	20	6000
3	Peter	50	2500
4	Diana	60	4200
5	Jonathon	80	8600
6	Neena	90	17000
7	Lex	90	17000
8	William	110	8300

Correlated Subquery

- Correlated subquery는 행 단위 처리에 사용됩니다. 각 subquery는 outer query의 모든 행에 대해 한 번씩 실행됩니다.



Correlated Subquery

- Subquery는 상위 query에 있는 테이블의 열을 참조합니다.

```
SELECT column1, column2, ...
FROM   table1   Outer_table
WHERE  column1 operator
           (SELECT  column1, column2
                    FROM    table2
                    WHERE   expr1 =
            .expr2);        Outer_table
```


Correlated Subquery 사용

- 자신의 부서의 평균 급여보다 많은 급여를 받는 사원을 모두 찾습니다.

```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
               (SELECT AVG(salary)
                 FROM   employees inner_table
                 WHERE inner_table.department_id =
                 outer_table.department_id);
```

	LAST_NAME	SALARY	DEPARTMENT_ID
1	King	24000	90
2	Hunold	9000	60
3	Ernst	6000	60
4	Greenberg	12008	100
5	Faviet	9000	100
6	Raphaely	11000	30

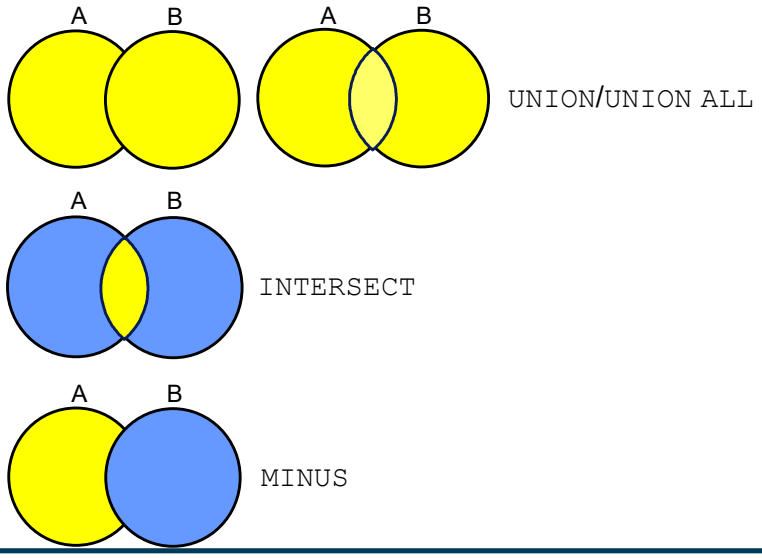
....

outer query의 행이 처리될 때마다 inner query가 평가됩니다.

7 집합 연산자

127

집합 연산자

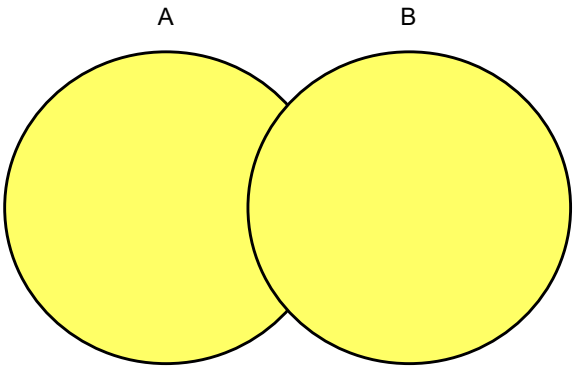


128

집합 연산자 규칙

- SELECT 리스트의 표현식은 개수가 일치해야 합니다.
- 후속 query에 있는 각 열의 데이터 유형은 첫번째 query에 있는 상응하는 열의 데이터 유형과 일치해야 합니다.
- ORDER BY 절은 명령문의 맨 끝에만 올 수 있습니다.
- 중복 행은 UNION ALL 외에는 자동으로 제거됩니다.
- 첫번째 query의 열 이름이 결과에 나타납니다.
- UNION ALL의 경우를 제외하고 출력은 기본적으로 오름차순으로 정렬됩니다.

UNION 연산자



UNION 연산자는 중복 행을 제거한 후 양쪽 query에서 행을 반환합니다.

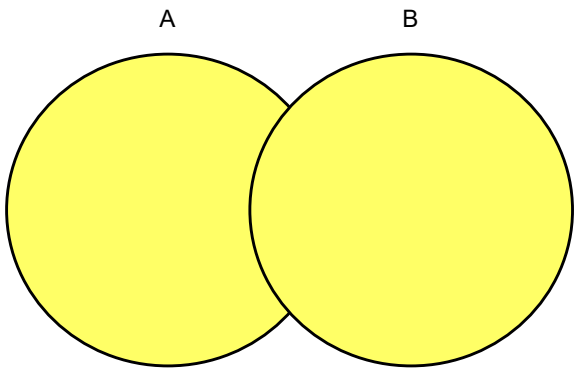
UNION 연산자 사용

•현재 사원 및 은퇴한 사원 모두의 직무 세부 정보를 표시합니다. 각 직무를 한 번만 표시합니다.

```
SELECT job_id
FROM   employees
UNION
SELECT job_id
FROM   retired_employees
```

	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	FI_ACCOUNT
7	FI_MGR
8	IT_PROG
9	MK_MAN
10	MK_REP
11	PU_CLERK
12	PU_MAN
13	SA_MAN
14	SA_REP
15	ST_CLERK
16	ST_MAN

UNION ALL 연산자



UNION ALL 연산자는 모든 중복 행을 포함하여 양쪽 query의 결과를 반환합니다.

UNION ALL 연산자 사용

•현재 사원 및 이전 사원 모두의 직무와 부서를 표시합니다.

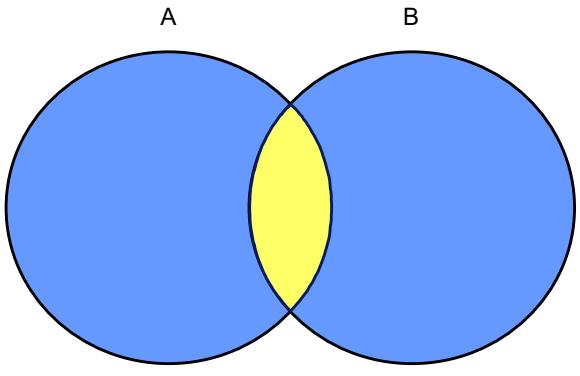
```
SELECT job_id, department_id
FROM   employees
UNION ALL
SELECT job_id, department_id
FROM   retired_employees
ORDER BY job_id;
```

	JOB_ID	DEPARTMENT_ID
1	AC_ACCOUNT	110
2	AC_MGR	110
3	AD_ASST	10
4	AD_PRES	90
5	AD_PRES	90
6	AD_VP	90
7	AD_VP	80
8	AD_VP	90
9	AD_VP	90

28	SA_REP	80
29	SA_REP	80
30	SA_REP	(null)
31	ST_CLERK	50
32	ST_CLERK	50
33	ST_CLERK	50
34	ST_CLERK	50
35	ST_MAN	50

…

INTERSECT 연산자



INTERSECT 연산자는 양쪽 query에 공통되는 행을 반환합니다.

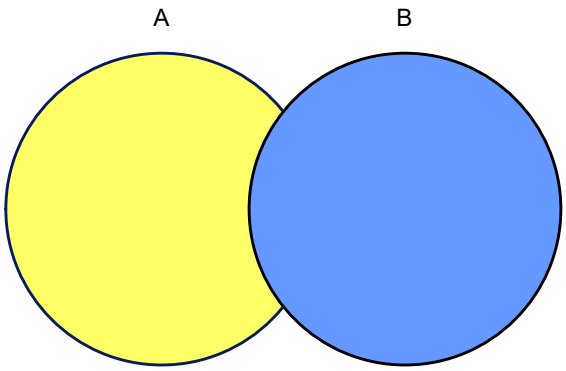
INTERSECT 연산자 사용

•현재 사원과 이전 사원의 공통된 관리자 ID 및 부서 ID를 표시합니다.

```
SELECT  manager_id,department_id
FROM    employees
INTERSECT
SELECT  manager_id,department_id
FROM    retired_employees
```

	MANAGER_ID	DEPARTMENT_ID
1	149	80

MINUS 연산자



MINUS 연산자는 첫번째 query에 의해 선택되지만 두번째 query 결과 집합에는 없는 모든 구분 행을 반환합니다.

MINUS 연산자 사용

•영업부에서 근무하는 사원의 사원 ID 및 직무 ID를 표시합니다.

```
SELECT employee_id, job_id
FROM   employees
WHERE  department_id = 80
MINUS
SELECT employee_id, job_id
FROM   retired_employees
WHERE  department_id = 90;
```

	EMPLOYEE_ID	JOB_ID
1	149	SA_MAN
2	174	SA_REP
3	176	SA_REP

SELECT 문 일치

• 열이 두 테이블 중 하나에 없는 경우 TO_CHAR 함수 또는 기타 변환 함수를 사용하여 데이터 유형을 일치시켜야 합니다.

```
SELECT location_id, department_name "Department",
   TO_CHAR(NULL) "Warehouse location"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
   state_province
FROM locations;
```

8 DML 명령문

DML(데이터 조작어)

- DML 문은 다음과 같은 경우에 실행합니다.
  - 테이블에 새 행 추가
  - 테이블의 기존 행 수정
  - 테이블에서 기존 행 제거

INSERT 문 구문

- INSERT 문을 사용하여 테이블에 새 행을 추가합니다.

```
INSERT INTO  table [(column [, column...])]
VALUES     (value [, value...]);
```

- 이 구문을 사용할 경우 한 번에 한 행만 삽입됩니다.

새 행 삽입

- 각 열에 대한 값을 포함하는 새 행을 삽입합니다.
- 테이블에 있는 열의 기본 순서로 값을 나열합니다.
- 선택적으로 INSERT 절에 열을 나열합니다.
- 문자와 날짜 값은 작은따옴표로 묶습니다.

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 rows inserted
```

Null 값을 가진 행 삽입

- 암시적 방법: 열 리스트에서 열을 생략합니다.

```
INSERT INTO  departments (department_id,
                          department_name)
VALUES    (30, 'Purchasing');
1 rows inserted
```

- 명시적 방법: VALUES 절에서 NULL 키워드를 지정합니다.

```
INSERT INTO  departments
VALUES    (100, 'Finance', NULL, NULL);
1 rows inserted
```


특수 값 삽입

- SYSDATE 함수는 현재 날짜와 시간을 기록합니다.

```
INSERT INTO employees (employee_id,
                       first_name, last_name,
                       email, phone_number,
                       hire_date, job_id, salary,
                       commission_pct, manager_id,
                       department_id)
VALUES          (113,
                       'Louis', 'Popp',
                       'LPOPP', '515.124.4567',
                       CURRENT_DATE, 'AC_ACCOUNT', 6900,
                       NULL, 205, 110);
1 rows inserted
```

특정 날짜 및 시간 값 삽입

- 새 사원을 추가합니다.

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 2003', 'MON DD, YYYY'),
             'SA_REP', 11000, 0.2, 100, 60);
1 rows inserted
```

- 추가한 내용을 확인합니다.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-03	SA_REP	11000	0.2	100

다른 테이블에서 행 복사

- INSERT 문을 subquery로 작성합니다.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

5 rows inserted.
```

- VALUES 절을 사용하지 마십시오.
- INSERT 절의 열 개수를 subquery의 열 개수와 일치시킵니다.

UPDATE 문 구문

- UPDATE 문을 사용하여 테이블의 기존 값을 수정합니다.

```
UPDATE    table
SET       column = value [, column = value, ...]
[WHERE    condition];
```

- 필요한 경우 한 번에 두 개 이상의 행을 갱신합니다.

테이블의 행 갱신

- WHERE 절을 지정하면 특정 행에서 값이 수정됩니다.

```
UPDATE employees
SET    department_id = 50
WHERE  employee_id = 113;
1 rows updated
```

- WHERE 절을 생략하면 테이블의 모든 행에서 값이 수정됩니다.

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated
```

Subquery를 사용하여 두 개의 열 갱신

- 사원 103의 직무와 급여를 사원 205와 일치하도록 갱신합니다.

```
UPDATE employees
SET      (job_id,salary)  = (SELECT  job_id,salary
                    FROM    employees
                    WHERE employee_id = 205)
         WHERE  employee_id = 103;
```

```
1 rows updated
```


다른 테이블을 기반으로 행 갱신

- UPDATE 문에서 subquery를 사용하여 다른 테이블의 값을 기반으로 테이블의 행 값을 갱신합니다.

```
UPDATE  copy_emp
SET     department_id  =  (SELECT department_id
                          FROM employees
                          WHERE employee_id = 100)
WHERE   job_id         =  (SELECT job_id
                          FROM employees
                          WHERE employee_id = 200);
```

```
1 rows updated
```

DELETE 문

•DELETE 문을 사용하여 테이블에서 기존 행을 제거할 수 있습니다.

```
DELETE [FROM]   table
[WHERE          condition];
```


테이블에서 행 삭제

- WHERE 절을 지정하면 특정 행이 삭제됩니다.

```
DELETE FROM departments
WHERE  department_name = 'Finance';
```

```
1 rows deleted
```

- WHERE 절을 생략하면 테이블의 모든 행이 삭제됩니다.

```
DELETE FROM  copy_emp;
```

```
22 rows deleted
```

다른 테이블을 기반으로 행 삭제

•DELETE 문에서 subquery를 사용하여 다른 테이블의 값을 기반으로 테이블에서 행을 제거합니다.

```
DELETE FROM employees
WHERE  department_id IN
                (SELECT department_id
                 FROM   departments
                 WHERE  department_name
                        LIKE '%Public%');
1 rows deleted
```


COMMIT 후의 데이터 상태

- 데이터 변경 사항이 데이터베이스에 저장됩니다.
- 이전의 데이터 상태를 겹쳐 씁니다.
- 모든 세션이 결과를 확인할 수 있습니다.
- 영향을 받는 행의 잠금이 해제되어 이러한 행을 다른 세션에서 조작할 수 있습니다.
- 모든 저장점이 지워집니다.

데이터 커밋

- 데이터를 변경합니다.

```
DELETE FROM EMPLOYEES
WHERE employee_id=113;
1 rows deleted
INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

- 변경 사항을 커밋합니다.

```
COMMIT;
commited.
```


ROLLBACK 후의 데이터 상태

•ROLLBACK 문을 사용하여 보류 중인 모든 변경 사항을 폐기합니다.

- 데이터 변경 사항이 실행 취소됩니다.
- 이전의 데이터 상태가 복원됩니다.
- 영향받는 행의 잠금이 해제됩니다.

```
DELETE FROM copy_emp;
ROLLBACK;
```

9 DDL 명령문

157

데이터베이스 객체

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
색인	일부 query의 성능을 향상시킵니다.
동의어	객체에 다른 이름을 부여합니다.

158

이름 지정 규칙

•테이블 이름 및 열 이름은 다음 규칙을 따라야 합니다.

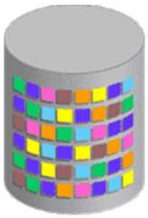
- 문자로 시작해야 합니다.
- 길이는 1-30자 사이여야 합니다.
- A-Z, a-z, 0-9, _, $, #만 포함할 수 있습니다.
- 동일한 유저가 소유한 다른 객체의 이름과 중복되면 안 됩니다.
- Oracle 서버 예약어는 사용할 수 없습니다.

CREATE TABLE 문

- 다음이 필요합니다.
  - CREATE TABLE 권한
  - 저장 영역

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

- 다음을 지정합니다.
  - 테이블 이름
  - 열 이름, 열 데이터 유형 및 열 크기



테이블 생성

- 테이블 생성:

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
table DEPT created.
```

- 테이블 생성 확인:

```
DESCRIBE dept
```

```
DESCRIBE dept
Name        Null Type
----------- ---- ------------
DEPTNO           NUMBER(2)
DNAME            VARCHAR2(14)
LOC              VARCHAR2(13)
CREATE_DATE      DATE
```


데이터 유형

데이터 유형	설명
VARCHAR2(*size*)	가변 길이 문자 데이터
CHAR(*size*)	고정 길이 문자 데이터
NUMBER(p, s)	가변 길이 숫자 데이터
DATE	날짜 및 시간 값
LONG	가변 길이 문자 데이터(최대 2GB)
CLOB	최대 크기는 (4GB - 1) * (DB_BLOCK_SIZE) 입니다.
RAW 및 LONG RAW	Raw binary data
BLOB	최대 크기는 (4GB - 1) * (DB_BLOCK_SIZE 초기화 파라미터(8TB - 128TB))입니다.
BFILE	외부 파일에 저장된 바이너리 데이터(최대 4GB)
ROWID	테이블에 있는 행의 고유한 주소를 나타내는 base-64 숫자 체계

Subquery를 사용하여 테이블 생성

- CREATE TABLE 문과 AS *subquery* 옵션을 결합하여 테이블을 생성하고 행을 삽입합니다.

```
CREATE TABLE table
          [(column, column...)]
AS subquery;
```

- 지정된 열 개수와 subquery 열 개수를 일치시킵니다.
- 열 이름과 기본값을 가진 열을 정의합니다.

Subquery를 사용하여 테이블 생성

```
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department_id = 80;
```

```
table DEPT80 created.
```

```
DESCRIBE dept80
```

```
Name                          Null     Type
----------------------------- -------- -------------
EMPLOYEE_ID                            NUMBER(6)
LAST_NAME                     NOT NULL VARCHAR2(25)
ANNSAL                                 NUMBER
HIRE_DATE                     NOT NULL DATE
```

ALTER TABLE 문

- ALTER TABLE 문을 사용하여 다음을 수행합니다.
  - 새 열 추가
  - 기존 열 정의 수정
  - 새 열에 기본값 정의
  - 열 삭제
  - 열 이름 바꾸기
  - 읽기 전용 상태로 테이블 변경

ALTER TABLE 문

- ALTER TABLE 문을 사용하여 열을 추가, 수정 또는 삭제할 수 있습니다.

```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP (column [, column] ...);
```

열 추가

- ADD 절을 사용하여 열을 추가합니다.

```
ALTER TABLE dept80
ADD         (job_id VARCHAR2(9));
```

```
table DEPT80 altered.
```

- 새 열은 마지막 열이 됩니다.

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	149	Zlotkey	10500	29-JAN-08	(null)
2	174	Abel	11000	11-MAY-04	(null)
3	176	Taylor	8600	24-MAR-06	(null)

열 수정

- 열의 데이터 유형, 크기 및 기본값을 변경할 수 있습니다.

```
ALTER TABLE dept80
MODIFY    (last_name VARCHAR2(30));
```

```
table DEPT80 altered.
```

- 기본값을 변경하면 이후에 테이블에 삽입하는 항목에만 적용됩니다.

열 삭제

- DROP COLUMN 절을 사용하여 테이블에서 더 이상 필요 없는 열을 삭제할 수 있습니다.

```
ALTER TABLE  dept80
DROP (job_id);
```

```
table DEPT80 altered.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	10500	29-JAN-08
2	174	Abel	11000	11-MAY-04
3	176	Taylor	8600	24-MAR-06

읽기 전용 테이블

- ALTER TABLE 구문을 사용하여 다음을 수행할 수 있습니다.
  - 테이블을 읽기 전용 모드로 설정하여 테이블을 유지 관리하는 동안 DDL 문 또는 DML 문에 의한 변경을 방지합니다.
  - 테이블을 읽기/쓰기 모드로 다시 되돌립니다.

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

테이블 삭제

- 테이블을 Recycle bin으로 이동
- PURGE 절이 지정되면 테이블 및 해당 데이터를 완전히 제거
- 종속 객체 무효화 및 테이블의 객체 권한 제거

```
DROP TABLE dept80;
table DEPT80 dropped.
```


10

제약 조건 소개

제약 조건 포함

- 제약 조건은 테이블 레벨에서 규칙을 강제 적용합니다.
- 제약 조건은 데이터베이스의 일관성 및 무결성을 보장합니다.
- 유효한 제약 조건 유형은 다음과 같습니다.
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

NOT NULL 제약 조건

- 열에 null 값이 허용되지 않도록 보장합니다.

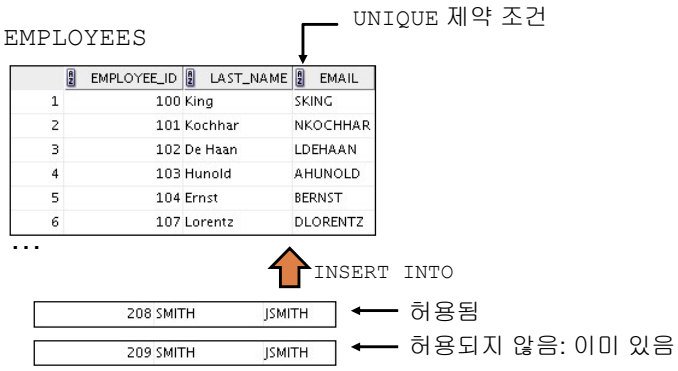
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	2600	(null)	50	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	2500	(null)	50	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	10500	0.2	80	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	11000	0.3	80	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	8600	0.2	80	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	7000	0.15	(null)	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	4400	(null)	10	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	13000	(null)	20	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	6000	(null)	20	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	12000	(null)	110	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	8300	(null)	110	WGIETZ	515.123.8181	07-JUN-94

NOT NULL 제약 조건
(Primary Key는 NOT NULL
제약 조건을 적용합니다.)

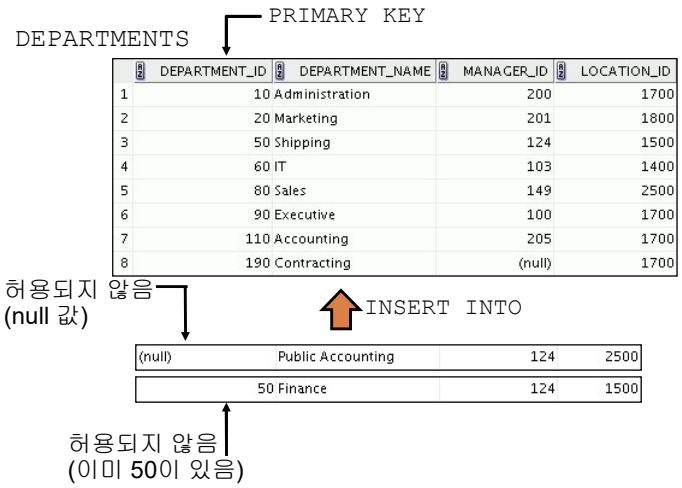
NOT NULL
제약 조건

NOT NULL 제약 조건 없음(모든
행에서 이 열에 대해 null 값을
포함할 수 있습니다.)

UNIQUE 제약 조건



PRIMARY KEY 제약 조건



FOREIGN KEY 제약 조건

177

CHECK 제약 조건

- 각 행이 충족해야 하는 조건을 정의합니다.
- 다른 테이블의 열을 참조할 수 없습니다.

```
..., salary   NUMBER(2)
     CONSTRAINT emp_salary_min
            CHECK (salary > 0),...
```


178

CREATE TABLE: 예제

```
CREATE TABLE teach_emp (
         empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         photo      BLOB,
         sal        NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL
                     CONSTRAINT admin_dept_fkey REFERENCES
                      departments (department_id));
```


제약 조건 위반

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
Error starting at line 1 in command:
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110
Error report:
SQL Error: ORA-02291: integrity constraint (ORA1.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```

- 부서 55가 존재하지 않습니다.

제약 조건 위반

- 다른 테이블에서 Foreign key로 사용되는 Primary key를 포함한 행은 삭제할 수 없습니다.

```
DELETE FROM departments
WHERE department_id = 60;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA1.JHIST_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

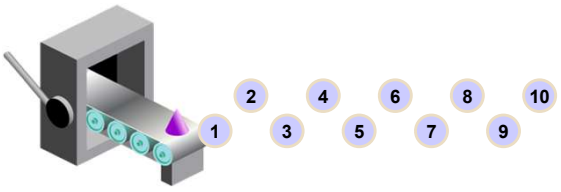

11 기타 객체 관리

데이터베이스 객체

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
색인	데이터 검색 query의 성능을 향상시킵니다.
동의어	객체에 대체 이름을 부여합니다.

시퀀스

- 시퀀스:
  - 고유 번호를 자동으로 생성할 수 있습니다.
  - 공유할 수 있는 객체입니다.
  - Primary key 값을 생성하는 데 사용할 수 있습니다.
  - 응용 프로그램 코드를 대체합니다.
  - 시퀀스 값이 메모리에서 캐시된 경우 액세스 속도가 향상됩니다.



CREATE SEQUENCE 문: 구문

- 자동으로 일련 번호를 생성하도록 시퀀스를 정의합니다.

```
CREATE SEQUENCE [ schema. ] sequence
  [ { START WITH|INCREMENT BY } integer
  | { MAXVALUE integer | NOMAXVALUE }
  | { MINVALUE integer | NOMINVALUE }
  | { CYCLE | NOCYCLE }
  | { CACHE integer | NOCACHE }
  | { ORDER | NOORDER }
];
```


시퀀스 생성

- DEPARTMENTS 테이블의 Primary key에 사용할 DEPT_DEPTID_SEQ라는 시퀀스를 생성합니다.
- CYCLE 옵션을 사용하면 숫자값의 중복이 발생할 수 있습니다.

```
CREATE SEQUENCE dept_deptid_seq
                START WITH 280
                INCREMENT BY 10
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
  sequence DEPT_DEPTID_SEQ created.
```

NEXTVAL 및 CURRVAL Pseudocolumn

- NEXTVAL은 사용 가능한 다음 시퀀스 값을 반환합니다. 다른 유저인 경우도 포함하여 참조될 때마다 고유 값을 반환합니다.
- CURRVAL은 가장 최근에 발생된 숫자값을 보여줍니다.
- CURRVAL을 통해 확인된 숫자 이후에 숫자가 NEXTVAL을 통해 발생됩니다.

시퀀스 사용

- 위치 ID 2500에 "Support"라는 새 부서를 삽입합니다.

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
```

```
1 rows inserted
```

- DEPT_DEPTID_SEQ 시퀀스의 현재 값을 확인합니다.

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

시퀀스 수정

- 증분값, 최대값, 최소값, 순환 옵션 또는 캐시 옵션을 변경합니다.

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ altered.
```


시퀀스 수정 지침

- 후속 시퀀스 번호에만 적용됩니다.
- 다른 번호로 시퀀스를 재시작하려면 시퀀스를 삭제하고 다시 생성해야 합니다.
- 일부 유효성 검사가 수행됩니다.
- 시퀀스를 제거하려면 DROP 문을 사용합니다.

```
DROP SEQUENCE dept_deptid_seq;
```

```
sequence DEPT_DEPTID_SEQ dropped.
```

시퀀스 정보

- USER_SEQUENCES 뷰는 유저가 소유한 모든 시퀀스를 설명합니다.

```
DESCRIBE user_sequences
```

```
DESCRIBE user_sequences
Name            Null     Type
--------------- -------- ------------
SEQUENCE_NAME   NOT NULL VARCHAR2(128)
MIN_VALUE                NUMBER
MAX_VALUE                NUMBER
INCREMENT_BY    NOT NULL NUMBER
CYCLE_FLAG               VARCHAR2(1)
ORDER_FLAG               VARCHAR2(1)
CACHE_SIZE      NOT NULL NUMBER
LAST_NUMBER     NOT NULL NUMBER
PARTITION_COUNT          NUMBER
SESSION_FLAG             VARCHAR2(1)
KEEP_VALUE               VARCHAR2(1)
```

- USER_SEQUENCES 데이터 딕셔너리 테이블에서 시퀀스 값을 확인합니다.

```
SELECT   sequence_name, min_value, max_value,
         increment_by, last_number
FROM     user_sequences;
```


동의어

- 동의어:
  - 데이터베이스 객체입니다.
  - 테이블 또는 기타 데이터베이스 객체에 대체 이름을 제공하기 위해 생성할 수 있습니다.
  - 데이터 딕셔너리에서 정의 이외의 저장 영역이 필요하지 않습니다.
  - 기본 스키마 객체의 ID 및 위치를 숨기는 데 유용합니다.

객체의 동의어 생성

•동의어(객체의 또 다른 이름)를 생성하면 객체에 쉽게 액세스할 수 있습니다. 동의어를 사용하여 다음을 수행할 수 있습니다.

- 다른 유저가 소유한 테이블을 쉽게 참조할 수 있습니다.
- 긴 객체 이름을 짧게 만듭니다.

```
CREATE [PUBLIC] SYNONYM synonym
FOR    object;
```


동의어 생성 및 제거

- DEPT_SUM_VU 뷰의 짧은 이름을 생성합니다.

```
CREATE SYNONYM d_sum
FOR  dept_sum_vu;
  synonym D_SUM created.
```

- 동의어를 삭제합니다.

```
DROP SYNONYM d_sum;
  synonym D_SUM dropped.
```

동의어 정보

```
DESCRIBE user_synonyms
```

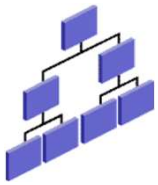
```
DESCRIBE user_synonyms
Name         Null     Type
------------ -------- -------------
SYNONYM_NAME NOT NULL VARCHAR2(128)
TABLE_OWNER           VARCHAR2(128)
TABLE_NAME   NOT NULL VARCHAR2(128)
DB_LINK               VARCHAR2(128)
```

```
SELECT *
FROM   user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	D_SUM	ORA21	DEPT_SUM_VU	(null)

인덱스

- 인덱스:
  - 스키마 객체입니다.
  - 오라클 서버에서 포인터를 사용하여 행 검색 속도를 높이는 데 사용할 수 있습니다.
  - 신속한 경로 액세스 방식을 사용하여 데이터를 빠르게 찾아 디스크 I/O(입/출력)를 줄일 수 있습니다.
  - 인덱스의 대상인 테이블에 종속적입니다.
  - 오라클 서버에서 자동으로 사용되고 유지 관리됩니다.



인덱스가 생성되는 방식

- 자동으로: 테이블 정의에서 PRIMARY KEY 또는 UNIQUE 제약 조건을 정의하면 고유 인덱스가 자동으로 생성됩니다.

- 수동으로: 행에 액세스하는 속도를 높이기 위해 유저가 열의 고유 또는 비고유 인덱스를 생성할 수 있습니다.

인덱스 생성

- 하나 이상의 열에 인덱스를 생성합니다.

```
CREATE [UNIQUE]INDEX index
ON table (column[, column]...);
```

- EMPLOYEES 테이블의 LAST_NAME 열에 대한 query 액세스 속도를 향상시킵니다.

```
CREATE INDEX  emp_last_name_idx
ON employees(last_name);

index EMP_LAST_NAME_IDX created.
```

인덱스 정보

- USER_INDEXES는 인덱스에 대한 정보를 제공합니다.
- USER_IND_COLUMNS는 유저가 소유한 인덱스 열과 테이블에 있는 인덱스 열에 대해 설명합니다.

```
DESCRIBE user_indexes
```

```
DESCRIBE user_indexes
Name                  Null     Type
--------------------- -------- -------------
INDEX_NAME            NOT NULL VARCHAR2(128)
INDEX_TYPE                     VARCHAR2(27)
TABLE_OWNER           NOT NULL VARCHAR2(128)
TABLE_NAME            NOT NULL VARCHAR2(128)
TABLE_TYPE                     VARCHAR2(11)
UNIQUENESS                     VARCHAR2(9)
```

...

USER_INDEXES: 예제

a

```
SELECT index_name, table_name, uniqueness
FROM   user_indexes
WHERE  table_name = 'EMPLOYEES';
```

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_NAME_IX	EMPLOYEES	NONUNIQUE
2	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
3	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
4	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
5	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
6	EMP_EMAIL_UK	EMPLOYEES	UNIQUE

...

b

```
SELECT index_name, table_name
FROM   user_indexes
WHERE  table_name = 'EMP_LIB';
```

	INDEX_NAME	TABLE_NAME
1	SYS_C0010979	EMP_LIB

USER_IND_COLUMNS Query

```
DESCRIBE user_ind_columns
```

```
DESCRIBE user_ind_columns
Name            Null Type
--------------- ---- --------------
INDEX_NAME           VARCHAR2(128)
TABLE_NAME           VARCHAR2(128)
COLUMN_NAME          VARCHAR2(4000)
COLUMN_POSITION      NUMBER
COLUMN_LENGTH        NUMBER
CHAR_LENGTH          NUMBER
DESCEND              VARCHAR2(4)
```

```
SELECT index_name, column_name,table_name
FROM   user_ind_columns
WHERE  index_name = 'LNAME_IDX';
```

	INDEX_NAME	COLUMN_NAME	TABLE_NAME
1	LNAME_IDX	LAST_NAME	EMP_TEST

인덱스 제거

- DROP INDEX 명령을 사용하여 데이터 딕셔너리에서 인덱스를 제거합니다.

```
DROP INDEX index;
```

- 데이터 딕셔너리에서 emp_last_name_idx 인덱스를 제거합니다.

```
DROP INDEX emp_last_name_idx;
  index EMP_LAST_NAME_IDX dropped.
```

- 인덱스를 삭제하려면 인덱스의 소유자이거나 DROP ANY INDEX 권한이 있어야 합니다.

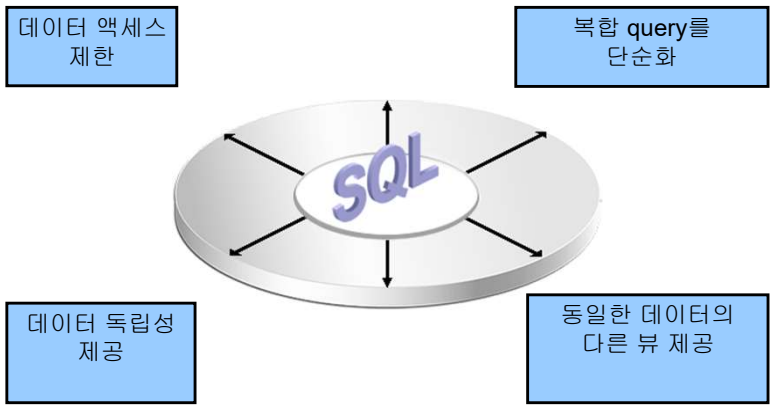
뷰란?

- **EMPLOYEES** 테이블

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104	Bru					ROG	6000
107						ROG	4200
						ACCOUNT	6900
						MAN	5800
						CLERK	3500
						CLERK	3100
						CLERK	2600
						CLERK	2500
						A_MAN	10500
						SA_REP	11000
					-98	SA_REP	8600
					-MAY-99	SA_REP	7000
					17-SEP-87	AD_ASST	4400
					17-FEB-96	MK_MAN	13000
					17-AUG-97	MK_REP	6000
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
101	Neena	Kochhar	17000
102	Lex	De Haan	17000
103	Alexander	Hunold	9000
104	Bruce	Ernst	6000

뷰의 이점



뷰 생성

- CREATE VIEW 문에 subquery를 포함시킵니다.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- 이 subquery에 복합 SELECT 구문을 포함할 수 있습니다.

뷰 생성

- 부서 80의 사원에 대한 세부 정보를 포함하는 EMPVU80 뷰를 생성합니다.

```
CREATE VIEW   empvu80
 AS SELECT  employee_id, last_name, salary
    FROM    employees
    WHERE   department_id = 80;
view EMPVU80 created.
```

- DESCRIBE 명령을 사용하여 뷰의 구조를 설명합니다.

```
DESCRIBE empvu80;
```

뷰 생성

- subquery에서 열 alias를 사용하여 뷰를 생성합니다.

```
CREATE VIEW  salvu50
 AS SELECT  employee_id ID_NUMBER, last_name NAME,
            salary*12 ANN_SALARY
    FROM    employees
    WHERE   department_id = 50;
```

```
view SALVU50 created.
```

- 제공된 alias 이름으로 이 뷰에서 열을 선택합니다.

뷰에서 데이터 검색

```
SELECT *
FROM   salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	120	Weiss	96000
2	121	Fripp	98400
3	122	Kaufling	94800
4	123	Vollman	78000
5	124	Mourgos	69600
6	125	Nayer	38400
7	126	Mikkilineni	32400

...

뷰 수정

- CREATE OR REPLACE VIEW 절을 사용하여 EMPVU80 뷰를 수정합니다. 각 열 이름에 alias를 추가합니다.

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
   FROM    employees
   WHERE   department_id = 80;
view EMPVU80 created.
```

- CREATE OR REPLACE VIEW 절의 열 alias는 subquery의 열과 동일한 순서로 나열되어야 합니다.

복합 뷰 생성

- 그룹 함수를 포함하는 복합 뷰를 생성하여 두 테이블의 값을 표시합니다.

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
            MAX(e.salary),AVG(e.salary)
   FROM     employees e JOIN departments d
   USING    (department_id)
   GROUP BY d.department_name;
view DEPT_SUM_VU created.
```

뷰 정보

1
```
DESCRIBE user_views
```

Name	Null	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG()

...

2
```
SELECT view_name FROM user_views;
```

	VIEW_NAME
1	EMP_DETAILS_VIEW
2	SALVU50
3	EMPVU80
4	DEPT_SUM_VU

3
```
SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';
```

	TEXT
1	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.co

...

```
AND c.region_id = r.region_id  AND j.job_id = e.job_idWITH READ ONLY
```


뷰 제거

- 뷰는 데이터베이스의 기본 테이블을 기반으로 하기 때문에 뷰를 제거해도 데이터는 손실되지 않습니다.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
```

```
view EMPVU80 dropped.
```