

# Oracle Database 11g: SQL Fundamentals I(한글판)

학생용 • 볼륨 I

D49996KR10

Edition 1.0

2007년 8월

D54000

**ORACLE®**



Copyright © 2007, Oracle. All rights reserved.

ORACLE®

## 목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- **equijoin** 및 **nonequijoin**을 사용하여 두 개 이상의 테이블에서 데이터에 액세스하는 **SELECT** 문 작성
- **Self-join**을 사용하여 테이블을 자체 조인
- **Outer join**을 사용하여 일반적으로 조인 조건을 충족하지 않는 데이터 보기
- 두 개 이상의 테이블에서 모든 행의 **Cartesian Product** 생성

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### 목표

이 단원에서는 두 개 이상의 테이블에서 데이터를 가져오는 방법에 대해 설명합니다. 조인은 여러 테이블의 정보를 보는 데 사용됩니다. 따라서 테이블을 조인하여 두 개 이상의 테이블에 있는 정보를 볼 수 있습니다.

주: 조인에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "SQL Queries and Subqueries: Joins" 관련 섹션을 참조하십시오.

## 여러 테이블에서 데이터 가져오기

**EMPLOYEES**

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

**DEPARTMENTS**

	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

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### 여러 테이블에서 데이터 가져오기

때때로 두 개 이상의 테이블에서 데이터를 사용해야 할 경우가 있습니다. 슬라이드 예제에서는 별도의 두 테이블에서 가져온 데이터가 보고서에 표시됩니다.

- 사원 ID는 EMPLOYEES 테이블에 있습니다.
- 부서 ID는 EMPLOYEES 테이블과 DEPARTMENTS 테이블에 모두 있습니다.
- 부서 이름은 DEPARTMENTS 테이블에 있습니다.

이 보고서를 작성하려면 EMPLOYEES 및 DEPARTMENTS 테이블을 연결하고 두 테이블에서 데이터에 액세스해야 합니다.

# Cartesian Product

- 다음과 같은 경우에 **Cartesian Product**가 형성됩니다.
  - 조인 조건이 생략된 경우
  - 조인 조건이 잘못된 경우
  - 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행에 조인되는 경우
- **Cartesian Product**를 피하려면 항상 **WHERE** 절에 유효한 조인 조건을 포함하십시오.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## Cartesian Product

조인 조건이 잘못되거나 완전히 생략된 경우 결과는 모든 행 조합이 표시되는 *Cartesian Product*로 나타납니다. 즉, 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행에 조인됩니다.

**Cartesian Product**는 많은 행을 생성하므로 결과는 그다지 유용하지 않습니다. 따라서 특별히 모든 테이블에서 모든 행을 조합해야 하는 경우가 아니면 항상 유효한 조인 조건을 포함해야 합니다.

그러나 **Cartesian Product**는 일부 테스트에서 적당량의 데이터를 시뮬레이트하기 위해 많은 행을 생성해야 하는 경우에 유용합니다.

# Cartesian Product 생성

**EMPLOYEES(20행)**

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60

...

19	205	Higgins	110
20	206	Gietz	110

**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 = 160 행**

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	90	1700
2	101	90	1700
3	102	90	1700
4	103	60	1700

...

159	205	110	1700
160	206	110	1700

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## Cartesian Product 생성

조인 조건을 생략하면 Cartesian Product가 생성됩니다. 슬라이드의 예제에서는 EMPLOYEES 및 DEPARTMENTS 테이블에서 가져온 사원의 성과 부서 이름을 표시합니다. 조인 조건이 지정되지 않았기 때문에 EMPLOYEES 테이블의 모든 행(20행)이 DEPARTMENTS 테이블의 모든 행(8행)과 조인되어 출력에 160행이 생성됩니다.

```
SELECT last_name, department_name dept_name
FROM employees, departments;
```

	LAST_NAME	DEPT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

159	Whalen	Contracting
160	Zlotkey	Contracting

# Oracle 고유의 조인 유형

- **Equijoin**
- **Nonequijoin**
- **Outer join**
- **Self-join**

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## 조인 유형

테이블을 조인하기 위해 Oracle의 조인 구문을 사용할 수 있습니다.

주: Oracle9i 릴리스 이전에는 고유 조인 구문이 사용되었습니다. SQL:1999 호환 조인 구문은 Oracle 고유의 조인 구문과 비교하여 별다른 성능 이점을 제공하지 않습니다.

Oracle에는 SQL:1999 호환 조인 구문의 FULL OUTER JOIN을 지원하는 해당 구문이 없습니다.

# Oracle 구문을 사용하여 테이블 조인

조인을 사용하여 둘 이상의 테이블에서 데이터를 **query**합니다.

```
SELECT    table1.column, table2.column
FROM      table1, table2
WHERE     table1.column1 = table2.column2;
```

- **WHERE** 절에 조인 조건을 작성합니다.
- 동일한 열 이름이 둘 이상의 테이블에 있을 경우에는 테이블 이름을 열 이름 앞에 붙입니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## Oracle 구문을 사용하여 테이블 조인

데이터베이스에 있는 둘 이상의 테이블에서 데이터가 필요한 경우 **조인** 조건을 사용합니다. 해당 열(일반적으로 Primary Key 또는 Foreign Key 열)에 존재하는 공통 값에 따라 한 테이블의 행을 다른 테이블의 행에 조인할 수 있습니다.

둘 이상의 관련 테이블에서 데이터를 표시하려면 WHERE 절에 간단한 조인 조건을 작성합니다.

구문 설명:

*table1.column*            데이터가 검색되는 테이블과 열을 나타냅니다.

*table1.column1* =        테이블을 함께 조인 또는 연관시키는 조건입니다.

*table2.column2*

지침

- 테이블을 조인하는 SELECT 문을 작성할 때는 열 이름 앞에 테이블 이름을 붙여서 표시를 명확히 하고 데이터베이스 액세스를 향상합니다.
- 둘 이상의 테이블에서 동일한 열 이름이 나타나면 열 이름 앞에 테이블 이름을 붙여야 합니다.
- *n*개의 테이블을 함께 조인하려면 최소 *n-1*개의 조인 조건이 필요합니다. 예를 들어, 네 개의 테이블을 조인하려면 최소 세 개의 조인이 필요합니다. 이 규칙은 테이블에 연결된 Primary Key가 있을 때는 적용되지 않습니다. 이 경우에는 각 행을 고유하게 식별하기 위해 둘 이상의 열이 필요합니다.



## 모호한 열 이름 한정

- 테이블 접두어를 사용하여 여러 테이블에 있는 열 이름을 한정합니다.
- 테이블 접두어를 사용하여 성능을 향상합니다.
- 전체 테이블 이름 접두어 대신 테이블 **alias**를 사용합니다.
- 테이블 **alias**로 테이블에 짧은 이름을 지정합니다.
  - **SQL** 코드 크기를 줄여 메모리를 적게 사용합니다.
- 열 **alias**를 사용하여 이름은 같지만 서로 다른 테이블에 상주하는 열을 구분합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### 모호한 열 이름 한정

두 개 이상의 테이블을 조인하는 경우 모호성을 피하기 위해 열 이름을 테이블 이름으로 한정해야 합니다. 테이블 접두어를 사용하지 않으면 `SELECT` 리스트의 `DEPARTMENT_ID` 열을 `DEPARTMENTS` 테이블이나 `EMPLOYEES` 테이블에서 가져올 수 있습니다. 따라서 query를 실행하려면 테이블 접두어를 추가해야 합니다. 두 테이블 간에 공통되는 열 이름이 없으면 열을 한정할 필요가 없습니다. 하지만 테이블 접두어를 사용하면 Oracle 서버에 열을 찾을 위치를 정확히 알려줄 수 있으므로 성능이 향상됩니다.

테이블 이름으로 열 이름을 한정하는 것은 시간이 많이 소요되는 작업이 될 수 있으며 테이블 이름이 길 경우 더욱 그렇습니다. 따라서 테이블 이름 대신 테이블 **alias**를 사용할 수 있습니다. 열 **alias**로 열에 다른 이름을 지정할 수 있는 것처럼 테이블에도 테이블 **alias**로 다른 이름을 지정합니다. 테이블 **alias**를 사용하면 SQL 코드를 더 작게 유지할 수 있으므로 메모리 사용량도 줄어듭니다.

테이블의 전체 이름을 지정하고 그 뒤에 공백과 테이블 **alias**를 차례로 배치합니다. 예를 들어 `EMPLOYEES` 테이블에는 `e`라는 **alias**를 지정하고 `DEPARTMENTS` 테이블에는 `d`라는 **alias**를 지정할 수 있습니다.

## 모호한 열 이름 한정(계속)

### 지침

- 테이블 alias는 30자까지 사용할 수 있지만 길이는 짧을수록 좋습니다.
- FROM 절에서 특정 테이블 이름에 테이블 alias를 사용하는 경우 SELECT 문 전체에서 테이블 이름 대신 테이블 alias를 사용해야 합니다.
- 테이블 alias는 의미 있는 단어여야 합니다.
- 테이블 alias는 현재 SELECT 문에 대해서만 유효합니다.

# Equijoin

**EMPLOYEES**

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80

...

**DEPARTMENTS**

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
50	Shipping
60	IT
80	Sales
90	Executive
110	Accounting
190	Contracting

Foreign Key

Primary Key

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## Equijoin

사원의 부서 이름을 확인하려면 EMPLOYEES 테이블의 DEPARTMENT\_ID 열의 값을 DEPARTMENTS 테이블의 DEPARTMENT\_ID 값과 비교합니다. EMPLOYEES 테이블과 DEPARTMENTS 테이블 사이의 관계는 *Equijoin*입니다. 즉, 두 테이블의 DEPARTMENT\_ID 열에 있는 값이 같아야 합니다. 대개 이러한 유형의 조인에는 Primary Key 및 Foreign Key가 보완 요소로 포함됩니다.

주: Equijoin은 *simple join* 또는 *inner join*이라고도 합니다.

## Equijoin을 사용하여 레코드 검색

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  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

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Equijoin을 사용하여 레코드 검색

슬라이드의 예제는 다음과 같습니다.

- **SELECT** 절은 다음과 같이 검색할 열 이름을 지정합니다.
  - 사원의 성, 사원 번호 및 부서 번호(EMPLOYEES 테이블의 열)
  - 부서 번호, 부서 이름 및 위치 ID(DEPARTMENTS 테이블의 열)
- **FROM** 절은 데이터베이스에서 액세스해야 하는 다음의 두 테이블을 지정합니다.
  - EMPLOYEES 테이블
  - DEPARTMENTS 테이블
- **WHERE** 절은 테이블이 조인되는 방식을 지정합니다.
  - e.department\_id = d.department\_id

DEPARTMENT\_ID 열이 두 테이블에 공통되므로 명확하게 구분하기 위해서는 열 이름 앞에 테이블 alias를 붙여야 합니다. 두 테이블 모두에 있지 않은 다른 열은 테이블 alias로 한정할 필요는 없지만 더 나은 성능을 위해 이렇게 하는 것이 좋습니다.

주: SQL Developer에서는 접미어 "\_1"을 사용하여 두 DEPARTMENT\_ID를 구분합니다.

## Equijoin을 사용하여 레코드 검색: 예제

```
SELECT d.department_id, d.department_name,  
       d.location_id, l.city  
FROM   departments d, locations l  
WHERE  d.location_id = l.location_id;
```

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

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Equijoin을 사용하여 레코드 검색: 예제

슬라이드의 예제에서는 LOCATIONS 테이블과 DEPARTMENT 테이블을 두 테이블에서 유일하게 이름이 같은 열인 LOCATION\_ID 열로 조인합니다. 열을 한정하고 모호성을 방지하기 위해 테이블 alias를 사용합니다.

## AND 연산자를 사용한 추가 검색 조건

```
SELECT d.department_id, d.department_name, l.city
FROM departments d, locations l
WHERE d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### AND 연산자를 사용한 추가 검색 조건

조인과 함께 WHERE 절에 대한 조건을 설정하여 조인에서 하나 이상의 테이블에 대해 고려 대상인 행을 제한할 수 있습니다. 슬라이드의 예제에서는 출력 행을 부서 ID가 20 또는 50인 행으로 제한합니다.

예를 들어, Matos라는 사원의 부서 번호와 부서 이름을 표시하려면 WHERE 절에서 다음과 같은 추가 조건이 필요합니다.

```
SELECT e.last_name, e.department_id,
       d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND last_name = 'Matos';
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

## 세 개 이상의 테이블 조인

**EMPLOYEES**

	LAST_NAME	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60
6	Lorentz	60
7	Mourgos	50
8	Rajs	50
9	Davies	50
10	Matos	50

...

**DEPARTMENTS**

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

**LOCATIONS**

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

$n$ 개의 테이블을 함께 조인하려면 최소  $n-1$ 개의 조인 조건이 필요합니다. 예를 들어, 세 개의 테이블을 조인하려면 최소 두 개의 조인이 필요합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### 세 개 이상의 테이블 조인

때때로 세 개 이상의 테이블을 조인해야 할 경우가 있습니다. 예를 들어, 각 사원의 성, 부서 이름 및 근무지를 표시하려면 EMPLOYEES, DEPARTMENTS 및 LOCATIONS 테이블을 조인해야 합니다.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

	LAST_NAME	DEPARTMENT_NAME	CITY
1	Abel	Sales	Oxford
2	Davies	Shipping	South San Francisco
3	De Haan	Executive	Seattle
4	Ernst	IT	Southlake
5	Fay	Marketing	Toronto

...

# Nonequijoin

**EMPLOYEES**

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Davies	3100
10	Matos	2600
...		
19	Higgins	12000
20	Gietz	8300

**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** 테이블에서는 각 **GRADE\_LEVEL**에 대해 **LOWEST\_SAL** 및 **HIGHEST\_SAL** 값의 범위를 정의합니다. 따라서 **GRADE\_LEVEL** 열을 사용하여 각 사원에 등급을 지정할 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## Nonequijoin

Nonequijoin은 등호 연산자 외의 다른 연산자를 포함하는 조인 조건입니다.

Nonequijoin의 한 예로 **EMPLOYEES** 테이블과 **JOB\_GRADES** 테이블 사이의 관계를 들 수 있습니다. **EMPLOYEES** 테이블의 **SALARY** 열은 **JOB\_GRADES** 테이블의 **LOWEST\_SAL** 열에 있는 값과 **HIGHEST\_SAL** 열에 있는 값 사이의 범위에 해당하는 값을 가집니다. 따라서 급여를 기준으로 각 사원의 등급을 지정할 수 있습니다. 이때 등호(=) 연산자가 아닌 다른 연산자를 사용하여 관계를 구합니다.



## Nonequijoin을 사용하여 레코드 검색

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  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

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Nonequijoin을 사용하여 레코드 검색

슬라이드의 예제에서는 사원의 급여 등급을 평가하는 nonequijoin을 생성합니다. 급여는 낮은 급여 범위와 높은 급여 범위 *사이에* 있어야 합니다.

이 query를 실행하면 모든 사원이 정확히 한 번만 나타납니다. 리스트에서 반복되는 사원은 없습니다. 여기에는 다음과 같은 두 가지 이유가 있습니다.

- 직무 등급 테이블의 어떠한 행도 중복되는 등급을 포함하지 않습니다. 즉, 한 사원에 대한 급여 값은 급여 등급 테이블의 한 행에서 낮은 급여 값과 높은 급여 값 범위에만 속할 수 있습니다.
- 모든 사원의 급여는 직무 등급 테이블에서 제공하는 제한 범위 내에 놓입니다. 즉, LOWEST\_SAL 열에 포함된 최저값보다 적은 급여를 받거나 HIGHEST\_SAL 열에 포함된 최고값보다 많은 급여를 받는 사원은 없습니다.

주: <= 및 >=와 같은 다른 조건을 사용할 수 있지만 BETWEEN이 가장 간단합니다. BETWEEN 조건을 사용할 때는 맨 처음에 낮은 값을 지정하고 마지막에 높은 값을 지정해야 합니다. Oracle 서버는 BETWEEN 조건을 AND 조건 쌍으로 변환합니다. 따라서 BETWEEN을 사용해도 성능상이점이 전혀 없으므로 논리적으로 간결한 SQL 문을 작성하는 경우에만 BETWEEN을 사용해야 합니다.

슬라이드의 예제에서는 모호성을 피하기 위해서가 아니라 성능상의 이유로 테이블 alias를 지정했습니다.

## Outer Join을 사용하여 직접 일치되지 않는 레코드 반환

**DEPARTMENTS**

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

**EMPLOYEES**

	DEPARTMENT_ID	LAST_NAME
1	90	King
2	90	Kochhar
3	90	De Haan
4	60	Hunold
5	60	Ernst
6	60	Lorentz
7	50	Mourgos
8	50	Rajs
9	50	Davies
10	50	Matos
...		
19	110	Higgins
20	110	Gietz

부서 **190**에는 사원이 없습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Outer join을 사용하여 직접 일치되지 않는 레코드 반환

조인 조건을 충족하지 못하는 행은 query 결과에 나타나지 않습니다. 예를 들어, EMPLOYEES 테이블과 DEPARTMENTS 테이블의 Equijoin 조건에서 부서 ID 190은 나타나지 않습니다. 해당 부서 ID를 가진 사원이 EMPLOYEES 테이블에 기록되어 있지 않기 때문입니다. 또한 DEPARTMENT\_ID가 NULL로 설정된 사원이 있으므로 해당 행도 Equijoin의 query 결과에 표시되지 않습니다. 사원이 없는 부서 레코드를 반환하거나 부서에 속하지 않는 사원 레코드를 반환하려면 Outer join을 사용할 수 있습니다.

## Outer Join: 구문

- **Outer join**을 사용하면 조인 조건을 만족하지 않는 행을 볼 수 있습니다.
- **Outer join** 연산자는 더하기 기호를 괄호로 묶은 (+)입니다.

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column(+) = table2.column;
```

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column(+);
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Outer Join: 구문

조인 조건에서 *Outer join* 연산자를 사용하면 누락된 행이 반환될 수 있습니다. 이 연산자는 더하기 기호를 괄호로 묶은 (+)이며, 누락된 정보가 있는 조인의 "한쪽"에 놓입니다. 이 연산자는 하나 이상의 null 행이 생성되는 결과를 가져오며 누락되지 않은 테이블에 있는 하나 이상의 행을 이 null 행에 조인할 수 있습니다.

구문 설명:

`table1.column =`      테이블을 조인 또는 연관시키는 조건입니다.

`table2.column (+)` Outer join 기호입니다. WHERE 절 조건의 양쪽 중 한쪽에 놓을 수 있습니다. 일치하는 행이 없는 테이블의 열 이름 다음에 Outer join 기호를 놓습니다.

# Outer Join 사용

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  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
9	Hunold	60	IT
10	Ernst	60	IT
...			
19	Gietz	110	Accounting
20	(null)	(null)	Contracting

ORACLE

Copyright © 2007, Oracle. All rights reserved.

## Outer Join 사용

슬라이드의 예제에서는 사원의 성, 부서 ID 및 부서 이름을 표시합니다. Contracting 부서에는 사원이 없습니다. 따라서 출력에 빈 값이 표시됩니다.

### Outer join 제한 사항

- Outer join 연산자는 표현식의 한쪽, 즉 정보가 누락된 쪽에만 나타날 수 있습니다. 이 연산자는 테이블에서 다른 테이블과 직접 일치되는 항목이 없는 행을 반환합니다.
- Outer join을 포함하는 조건에서는 IN 연산자를 사용할 수 없으며 OR 연산자를 통해 다른 조건에 연결할 수 없습니다.

주: Oracle에는 SQL:1999 호환 조인 구문의 FULL OUTER JOIN에 해당하는 조인 구문이 없습니다.

## Outer Join: 다른 예제

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  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
...			
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Outer Join: 다른 예제

위 예제의 query에서는 DEPARTMENTS 테이블에 일치하는 행이 없어도 EMPLOYEES 테이블의 모든 행을 검색합니다.

## 테이블 자체 조인

**EMPLOYEES (WORKER)**

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

...

**EMPLOYEES (MANAGER)**

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
141	Rajs
142	Davies
143	Matos

...

**WORKER** 테이블의 **MANAGER\_ID**는 **MANAGER** 테이블의 **EMPLOYEE\_ID**와 같습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

### 테이블 자체 조인

때때로 테이블을 자체 조인해야 할 경우가 있습니다. 각 사원의 관리자 이름을 찾으려면 EMPLOYEES 테이블을 자체 조인하거나 Self-join을 수행해야 합니다. 예를 들어, Lorentz의 관리자 이름을 찾으려면 다음을 수행해야 합니다.

- EMPLOYEES 테이블에서 LAST\_NAME 열을 조사하여 Lorentz를 찾습니다.
- MANAGER\_ID 열을 조사하여 Lorentz의 관리자 번호를 찾습니다. Lorentz의 관리자 번호는 103입니다.
- LAST\_NAME 열을 조사하여 EMPLOYEE\_ID가 103인 관리자의 이름을 찾습니다. Hunold의 사원 번호가 103이므로 Hunold가 Lorentz의 관리자입니다.

이 과정에서 EMPLOYEES 테이블을 두 번 조사하게 됩니다. 먼저 테이블을 조사하여 LAST\_NAME 열에서 Lorentz를 찾고 MANAGER\_ID 값 103을 찾습니다. 이어서 EMPLOYEE\_ID 열을 조사하여 103을 찾고 LAST\_NAME 열에서 Hunold를 찾습니다.

## Self-Join: 예제

```
SELECT worker.last_name || ' works for ' || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

	WORKER.LAST_NAME  'WORKSFOR'  MANAGER.LAST_NAME
1	Hunold works for De Haan
2	Fay works for Hartstein
3	Gietz works for Higgins
4	Lorentz works for Hunold
5	Ernst works for Hunold
6	Zlotkey works for King
7	Mourgos works for King
8	Kochhar works for King
9	Hartstein works for King
10	De Haan works for King

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### Self-Join: 예제

슬라이드의 예제에서는 EMPLOYEES 테이블을 자체 조인합니다. FROM 절의 두 테이블을 시뮬레이트하기 위해 동일한 테이블 EMPLOYEES에 대해 두 개의 alias worker 및 manager를 사용합니다.

이 예제에서 WHERE 절에는 "작업자의 관리자 번호가 해당 관리자에 대한 사원 번호와 일치함"을 의미하는 조인이 포함됩니다.

## 요약

이 부록에서는 **Oracle** 고유의 구문을 통해 조인을 사용하여 여러 테이블의 데이터를 표시하는 방법을 배웁니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

### 요약

다양한 방법으로 테이블을 조인할 수 있습니다.

#### 조인 유형

- Cartesian Product
- Equijoin
- Nonequijoin
- Outer join
- Self-join

#### Cartesian Product

Cartesian Product는 결과에 모든 행 조합을 표시합니다. WHERE 절을 생략하면 이러한 결과가 나타납니다.

#### 테이블 **alias**

- 테이블 **alias**는 데이터베이스 액세스 속도를 높입니다.
- 테이블 **alias**를 사용하면 SQL 코드를 더 작게 유지할 수 있으므로 메모리 사용량도 줄어듭니다.



## 연습 C: 개요

이 연습에서는 다음 내용을 다룹니다.

- **Equijoin**을 사용하여 테이블 조인
- **Outer join** 및 **Self-join** 수행
- 조건 추가

ORACLE

Copyright © 2007, Oracle. All rights reserved.

### 연습 C: 개요

이 연습은 Oracle 조인 구문을 사용하여 두 개 이상의 테이블에서 데이터를 추출하는 과정을 실습할 수 있도록 구성되었습니다.

## 연습 C

1. HR 부서를 위해 모든 부서의 주소를 생성하는 query를 작성합니다. LOCATIONS 및 COUNTRIES 테이블을 사용합니다. 출력에 번지, 동/리, 구/군, 시/도 및 국가를 표시합니다. query를 실행합니다.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
2	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
3	1700	2004 Charade Rd	Seattle	Washington	United States of America
4	1800	460 Bloor St. W.	Toronto	Ontario	Canada
5	2500	Magdalen Centre, The ...	Oxford	Oxford	United Kingdom

2. HR 부서에서 모든 사원에 대한 보고서를 요구합니다. 모든 사원의 성, 부서 번호 및 부서 이름을 표시하는 query를 작성합니다. query를 실행합니다.

	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
9	Hunold	60	IT
10	Ernst	60	IT
...			
18	Higgins	110	Accounting
19	Gietz	110	Accounting

## 연습 C(계속)

- HR 부서에서 Toronto에 근무하는 사원에 대한 보고서를 요구합니다. Toronto에서 근무하는 모든 사원의 성, 직무, 부서 번호 및 부서 이름을 표시합니다.

	A Z LAST_NAME	A Z JOB_ID	A Z DEPARTMENT_ID	A Z DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

- 사원의 성 및 사원 번호를 해당 관리자의 성 및 관리자 번호와 함께 표시하는 보고서를 작성합니다. 열 레이블을 각각 Employee, Emp#, Manager 및 Mgr#으로 지정합니다. SQL 문을 lab\_c\_04.sql로 저장합니다.

	A Z Employee	A Z EMP#	A Z Manager	A Z Mgr#
1	Kochhar	101	King	100
2	De Haan	102	King	100
3	Hunold	103	De Haan	102
4	Ernst	104	Hunold	103
5	Lorentz	107	Hunold	103
6	Mourgos	124	King	100
7	Rajs	141	Mourgos	124
8	Davies	142	Mourgos	124
9	Matos	143	Mourgos	124
10	Vargas	144	Mourgos	124

■ ■ ■

15	Whalen	200	Kochhar	101
16	Hartstein	201	King	100
17	Fay	202	Hartstein	201
18	Higgins	205	Kochhar	101
19	Gietz	206	Higgins	205

## 연습 C(계속)

5. King을 비롯하여 관리자가 지정되지 않은 모든 사원을 표시하도록 lab\_c\_04.sql을 수정합니다. 사원 번호순으로 결과를 정렬합니다. SQL 문을 lab\_c\_05.sql로 저장합니다. lab\_c\_05.sql의 query를 실행합니다.

	A Z Employee	A Z EMP#	A Z Manager	A Z Mgr#
1	Hunold	103	De Haan	102
2	Fay	202	Hartstein	201
3	Gietz	206	Higgins	205
4	Lorentz	107	Hunold	103
5	Ernst	104	Hunold	103
6	Hartstein	201	King	100
7	Zlotkey	149	King	100
8	Mourgos	124	King	100
9	De Haan	102	King	100
10	Kochhar	101	King	100

...

17	Grant	178	Zlotkey	149
18	Taylor	176	Zlotkey	149
19	Abel	174	Zlotkey	149
20	King	100	(null)	(null)

6. HR 부서를 위해 사원의 성과 부서 번호 및 해당 사원과 동일한 부서에 근무하는 모든 사원을 표시하는 보고서를 작성합니다. 각 열에 적절한 레이블을 지정합니다. 이 스크립트를 lab\_c\_06.sql이라는 파일에 저장합니다.

	A Z DEPARTMENT	A Z EMPLOYEE	A Z COLLEAGUE
1	20	Fay	Hartstein
2	20	Hartstein	Fay
3	50	Davies	Matos
4	50	Davies	Mourgos
5	50	Davies	Rajs
6	50	Davies	Vargas
7	50	Matos	Davies
8	50	Matos	Mourgos
9	50	Matos	Rajs
10	50	Matos	Vargas

...

42	110	Higgins	Gietz
----	-----	---------	-------






## 연습 C(계속)

7. HR 부서에서 직책 등급 및 급여에 대한 보고서를 요구합니다. JOB\_GRADES 테이블에 익숙해지도록 먼저 JOB\_GRADES 테이블의 구조를 표시합니다. 그런 다음 모든 사원의 이름, 직무, 부서 이름, 급여 및 등급을 표시하는 query를 작성합니다.

```
DESC JOB_GRADES
```

Name	Null	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

3 rows selected

	 LAST_NAME	 JOB_ID	 DEPARTMENT_NAME	 SALARY	 GRADE_LEVEL
1	Vargas	ST_CLERK	Shipping	2500	A
2	Matos	ST_CLERK	Shipping	2600	A
3	Davies	ST_CLERK	Shipping	3100	B
4	Rajs	ST_CLERK	Shipping	3500	B
5	Lorentz	IT_PROG	IT	4200	B
6	Whalen	AD_ASST	Administration	4400	B
7	Mourgos	ST_MAN	Shipping	5800	B
8	Ernst	IT_PROG	IT	6000	C
9	Fay	MK_REP	Marketing	6000	C
10	Gietz	AC_ACCOUNT	Accounting	8300	C
■ ■ ■					
18	De Haan	AD_VP	Executive	17000	E
19	King	AD PRES	Executive	24000	E

## 연습 C(계속)

심화 연습에 도전하려면 다음 연습을 완료하십시오.

- HR 부서에서 Davies 이후에 채용된 모든 사원의 이름을 파악하려고 합니다. 사원 Davies 이후로 채용된 모든 사원의 이름과 채용 날짜를 표시하기 위한 query를 작성합니다.

	A2	LAST_NAME	HIRE_DATE
1		Lorentz	07-FEB-99
2		Mourgos	16-NOV-99
3		Matos	15-MAR-98
4		Vargas	09-JUL-98
5		Zlotkey	29-JAN-00
6		Taylor	24-MAR-98
7		Grant	24-MAY-99
8		Fay	17-AUG-97

- HR 부서에서 관리자보다 먼저 채용된 모든 사원의 이름과 채용 날짜 및 해당 관리자의 이름과 채용 날짜를 찾으려고 합니다. 이 스크립트를 lab\_c\_09.sql이라는 파일에 저장합니다.

	A2	LAST_NAME	HIRE_DATE	A2	LAST_NAME_1	HIRE_DATE_1
1		Whalen	17-SEP-87		Kochhar	21-SEP-89
2		Hunold	03-JAN-90		De Haan	13-JAN-93
3		Vargas	09-JUL-98		Mourgos	16-NOV-99
4		Matos	15-MAR-98		Mourgos	16-NOV-99
5		Davies	29-JAN-97		Mourgos	16-NOV-99
6		Rajs	17-OCT-95		Mourgos	16-NOV-99
7		Grant	24-MAY-99		Zlotkey	29-JAN-00
8		Taylor	24-MAR-98		Zlotkey	29-JAN-00
9		Abel	11-MAY-96		Zlotkey	29-JAN-00

**관련 데이터를 그룹화하여 보고서 생성**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 부록을 마치면 다음을 수행할 수 있습니다.**

- **ROLLUP 연산을 사용하여 소계 값 생성**
- **CUBE 연산을 사용하여 교차 분석 값 생성**
- **GROUPING 함수를 사용하여 ROLLUP 또는 CUBE에 의해 생성된 행 값 식별**
- **GROUPING SETS를 사용하여 단일 결과 집합 생성**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 부록에서는 다음 작업을 수행하는 방법을 설명합니다.

- ROLLUP 연산자로 데이터를 그룹화하여 소계 값 구하기
- CUBE 연산자로 데이터를 그룹화하여 교차 분석 값 구하기
- GROUPING 함수를 사용하여 ROLLUP 또는 CUBE 연산자에 의해 생성된 결과 집합의 집계 레벨 식별
- GROUPING SETS를 사용하여 UNION ALL 방식과 같은 단일 결과 집합 생성



## 그룹 함수 검토

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

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

- 예제:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct), MAX(hire_date)
FROM   employees
WHERE  job_id LIKE 'SA%';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 그룹 함수

GROUP BY 절을 사용하여 테이블의 행을 그룹으로 나눌 수 있습니다. 그런 다음 그룹 함수를 사용하여 각 그룹에 대한 요약 정보를 반환할 수 있습니다. 그룹 함수는 SELECT 리스트와 ORDER BY 및 HAVING 절에 나타날 수 있습니다. Oracle 서버는 각 행 그룹에 그룹 함수를 적용하고 각 그룹에 대해 단일 결과 행을 반환합니다.

**그룹 함수 유형:** 그룹 함수 AVG, SUM, MAX, MIN, COUNT, STDDEV 및 VARIANCE는 각각 하나의 인수만 허용합니다. AVG, SUM, STDDEV 및 VARIANCE 함수는 숫자 값에 대해서만 실행됩니다. MAX 및 MIN은 숫자, 문자 또는 날짜 데이터 값에 대해 실행될 수 있습니다. COUNT는 제공된 표현식에 대해 널이 아닌 행 수를 반환합니다. 슬라이드의 예제는 JOB\_ID가 SA로 시작하는 사원의 평균 급여, 급여의 표준 편차, 커미션을 받는 사원 수 및 최대 채용 날짜를 계산합니다.

### 그룹 함수 사용 지침

- 인수에 대한 데이터 유형은 CHAR, VARCHAR2, NUMBER 또는 DATE일 수 있습니다.
- COUNT(\*)를 제외한 모든 그룹 함수는 널 값을 무시합니다. 널 값을 다른 값으로 치환하려면 NVL 함수를 사용합니다. COUNT는 숫자 또는 0을 반환합니다.
- Oracle 서버는 GROUP BY 절이 사용될 때 암시적으로 지정된 그룹화 열의 결과 집합을 오름차순으로 정렬합니다. 이 기본 정렬 방식 대신 내림차순으로 정렬하려면 ORDER BY 절에 DESC를 사용하면 됩니다.

## GROUP BY 절 검토

- 구문:

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

- 예제:

```
SELECT  department_id, job_id, SUM(salary),
        COUNT(employee_id)
FROM    employees
GROUP BY department_id, job_id ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### GROUP BY 절 검토

슬라이드에 표시된 예제는 Oracle 서버에 의해 다음과 같이 평가됩니다.

- SELECT 절은 다음 열을 검색하도록 지정합니다.
  - EMPLOYEES 테이블의 부서 ID 열과 직무 ID 열
  - GROUP BY 절에 지정된 각 그룹의 모든 급여 합계 및 사원 수
- GROUP BY 절은 테이블에서 행이 그룹화되는 방법을 지정합니다. 사원 수 및 급여 합계는 각 부서에서 직무 ID별로 계산됩니다. 행은 먼저 부서 ID로 그룹화된 다음 각 부서 내의 직무별로 그룹화됩니다.

## HAVING 절 검토

- HAVING 절을 사용하여 표시할 그룹을 지정합니다.
- 제한 조건을 기준으로 추가로 그룹을 제한합니다.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### HAVING 절

그룹이 형성되고 그룹 함수가 계산된 다음 HAVING 절이 그룹에 적용됩니다. HAVING 절이 GROUP BY 절 앞에 올 수 있지만 GROUP BY 절을 먼저 두는 것이 더 논리적이므로 이렇게 하는 것이 좋습니다.

Oracle 서버는 사용자가 HAVING 절을 사용할 때 다음 단계를 수행합니다.

1. 행을 그룹화합니다.
2. 그룹에 그룹 함수를 적용하고 HAVING 절의 조건과 일치하는 그룹을 표시합니다.

## GROUP BY에 ROLLUP 및 CUBE 연산자 사용

- GROUP BY에 ROLLUP 또는 CUBE를 사용하여 상호 참조 열별로 대집계 행을 생성합니다.
- ROLLUP 그룹화는 일반 그룹화 행과 소계 값을 포함한 결과 집합을 생성합니다.
- CUBE 그룹화는 ROLLUP에 따른 행과 교차 분석 행이 포함된 결과 집합을 생성합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### GROUP BY에 ROLLUP 및 CUBE 연산자 사용

Query의 GROUP BY 절에 ROLLUP 및 CUBE 연산자를 지정하십시오. ROLLUP 그룹화는 일반 그룹화 행과 소계 값을 포함한 결과 집합을 생성합니다. ROLLUP 연산자는 총계도 계산합니다. GROUP BY 절의 CUBE 연산은 사양에서 가능한 모든 표현식의 조합 값에 준하여 선택한 행을 그룹화하고 각 그룹에 대한 요약 정보를 단일 행으로 반환합니다. CUBE 연산자를 사용하여 교차 분석 행을 생성할 수 있습니다.

**참고:** ROLLUP 및 CUBE를 사용할 경우에는 GROUP BY 절 다음의 열이 서로 의미 있는 실제 관계에 있어야 합니다. 그렇지 않은 경우 이 연산자는 부적절한 정보를 반환합니다.

## ROLLUP 연산자

- ROLLUP은 GROUP BY 절의 확장입니다.
- ROLLUP 연산을 사용하여 소계와 같은 누적 집계를 생성합니다.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ROLLUP 연산자

ROLLUP 연산자는 GROUP BY 문 내의 표현식에 대해 집계 및 대집계를 전달합니다. 보고서 작성자는 ROLLUP 연산자를 사용하여 결과 집합에서 통계 및 요약 정보를 추출할 수 있습니다. 누적 집계는 보고서, 차트 및 그래프에 사용할 수 있습니다.

ROLLUP 연산자는 GROUP BY 절에 지정된 열 리스트를 한 방향으로(오른쪽에서 왼쪽으로) 이동하여 그룹을 생성합니다. 그런 다음 이 그룹화에 집계 함수를 적용합니다.

#### 참고

- ROLLUP 연산자 없이  $n$ 차원(즉, GROUP BY 절에 있는  $n$ 개의 열)에서 소계를 생성하려면  $n+1$ 개의 SELECT 문을 UNION ALL과 연결해야 합니다. 이렇게 하면 SELECT 문마다 테이블에 액세스하게 되므로 query가 비효율적으로 실행됩니다. ROLLUP 연산자는 테이블에 한 번만 액세스하여 해당 결과를 수집합니다. 소계 생성에 관련된 열이 많은 경우에는 ROLLUP 연산자가 유용합니다.
- 소계와 총계는 ROLLUP을 통해 생성됩니다. CUBE 역시 총계를 생성하며 가능한 각 방향으로 효과적으로 롤업하여 교차 분석 데이터를 생성합니다.

## ROLLUP 연산자: 예제

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	10	(null)	4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20	(null)	19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30	(null)	24900
9	40	HR_REP	6500
10	40	(null)	6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50	(null)	156400
15	(null)	(null)	211200

1

2

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ROLLUP 연산자 예제

슬라이드의 예제는 다음과 같습니다.

- GROUP BY 절은 부서 ID가 60보다 작은 부서에 대해 부서 내의 모든 직무 ID에 대한 총 급여를 표시합니다.
- ROLLUP 연산자는 다음을 표시합니다.
  - 부서 ID가 60보다 작은 각 부서의 총 급여
  - 직무 ID와 관계없이 부서 ID가 60보다 작은 모든 부서의 총 급여

이 예제에서 1은 DEPARTMENT\_ID와 JOB\_ID로 집계된 그룹을 나타내고, 2는 DEPARTMENT\_ID로만 집계된 그룹을 나타내며, 3은 총계를 나타냅니다.

ROLLUP 연산자는 GROUP BY 절에 지정된 그룹화 리스트를 따라 세부 레벨에서 전체 총계까지 롤업하는 소계를 생성합니다. 먼저 GROUP BY 절에 지정된 그룹에 대한 표준 집계 값을 계산합니다(예제에서는 부서 내의 각 직무로 그룹화된 급여 합계). 그런 다음 그룹 열 리스트에 따라 오른쪽에서 왼쪽으로 이동하여 점차 높은 레벨의 소계를 생성합니다. 예제에서는 각 부서의 급여 합계가 계산되고 그 다음에 모든 부서의 급여 합계가 계산됩니다.

- GROUP BY 절의 ROLLUP 연산자에서  $n$  표현식을 사용할 경우 연산 결과는  $n+1$ (이 경우  $2+1=3$ )개의 그룹이 됩니다.
- 첫번째  $n$  표현식의 값을 기반으로 하는 행을 행 또는 일반 행이라고 하며 나머지 행을 대집계 행이라고 합니다.

## CUBE 연산자

- CUBE는 GROUP BY 절의 확장입니다.
- CUBE 연산자를 사용하여 단일 SELECT 문으로 교차 분석 값을 생성할 수 있습니다.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CUBE 연산자

CUBE 연산자는 SELECT 문의 GROUP BY 절에 있는 추가 스위치입니다. CUBE 연산자는 AVG, SUM, MAX, MIN 및 COUNT를 포함하여 모든 집계 함수에 적용할 수 있습니다. CUBE 연산자는 일반적으로 교차 분석 보고서에 사용되는 결과 집합을 생성하는 데 사용됩니다. ROLLUP이 가능한 소계 조합의 일부만 생성하는 반면, CUBE는 GROUP BY 절에 지정된 가능한 모든 그룹화 조합의 소계와 총계를 생성합니다.

CUBE 연산자는 집계 함수와 함께 사용되어 결과 집합에 추가 행을 생성합니다. GROUP BY 절에 포함된 열은 상호 참조되어 대집합 그룹을 생성합니다. 선택 리스트에 지정된 집계 함수는 이러한 그룹에 적용되어 추가 대집계 행의 요약 값을 생성합니다. 결과 집합의 추가 그룹 수는 GROUP BY 절에 포함된 열 수에 의해 결정됩니다.

실제로 GROUP BY 절에서 가능한 열이나 표현식의 모든 조합을 사용하여 대집계를 생성할 수 있습니다. GROUP BY 절에  $n$  열이나 표현식이 있을 경우  $2^n$ 개의 가능한 대집계 조합이 있습니다. 수학적으로 이러한 조합은  $n$ 차원 큐브를 형성하며 연산자는 이러한 방법으로 해당 이름을 얻습니다.

응용 프로그램이나 프로그래밍 도구를 사용하여 이러한 대집계 값을 차트와 그래프에 제공하여 결과와 관계를 시각적이고 효율적으로 전달할 수 있습니다.

## CUBE 연산자: 예제

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 60
GROUP BY CUBE (department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)	
1	(null)	(null)	211200	1
2	(null)	HR_REP	6500	
3	(null)	MK_MAN	13000	
4	(null)	MK_REP	6000	
5	(null)	PU_MAN	11000	
6	(null)	ST_MAN	36400	2
7	(null)	AD_ASST	4400	
8	(null)	PU_CLERK	13900	
9	(null)	SH_CLERK	64300	
10	(null)	ST_CLERK	55700	
11	10	(null)	4400	3
12	10	AD_ASST	4400	
13	20	(null)	19000	
14	20	MK_MAN	13000	
15	20	MK_REP	6000	
16	30	(null)	24900	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CUBE 연산자 예제

이 예제에서 SELECT 문의 출력은 다음과 같이 해석될 수 있습니다.

- 부서 내의 모든 직무에 대한 총 급여(부서 ID가 60보다 작은 부서의 경우)
- 부서 ID가 60보다 작은 각 부서의 총 급여
- 부서와 관계없이 각 직무에 대한 총 급여
- 직책과 관계없이 부서 ID가 60보다 작은 부서의 총 급여

이 예제에서 1은 총계를 나타내고, 2는 JOB\_ID만으로 집계된 행을 나타내며, 3은 DEPARTMENT\_ID와 JOB\_ID로 집계된 일부 행을 나타내고, 4는 DEPARTMENT\_ID만으로 집계된 일부 행을 나타냅니다.

CUBE 연산자도 ROLLUP 연산을 수행하여 직위와 관계없이 부서 ID가 60보다 작은 부서의 소계와 부서 ID가 60보다 작은 부서의 총 급여를 표시합니다. 또한 CUBE 연산자는 부서와 관계없이 모든 직무에 대한 총 급여를 표시합니다.

**참고:** ROLLUP 연산자와 마찬가지로 CUBE 연산자 없이  $n$ 차원(즉, GROUP BY 절에 있는  $n$ 개의 열)에서 소계를 생성하려면  $2^n$ 개의 SELECT 문을 UNION ALL과 연결해야 합니다. 따라서 3차원의 보고서를 생성하기 위해서는  $2^3 = 8$ , 즉 8개의 SELECT 문을 UNION ALL로 연결해야 합니다.



## GROUPING 함수

### GROUPING 함수:

- CUBE 또는 ROLLUP 연산자와 함께 사용됩니다.
- 행에서 소계를 형성하는 그룹을 찾는 데 사용됩니다.
- ROLLUP 또는 CUBE로 생성된 NULL 값과 저장된 NULL 값을 구분하는 데 사용됩니다.
- 0 또는 1을 반환합니다.

```
SELECT      [column,] group_function(column) .. ,  
            GROUPING(expr)  
FROM        table  
[WHERE      condition]  
[GROUP BY  [ROLLUP][CUBE] group_by_expression]  
[HAVING    having_expression]  
[ORDER BY  column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### GROUPING 함수

GROUPING 함수를 CUBE 또는 ROLLUP 연산자와 함께 사용하면 요약 값을 구하는 방식을 이해하는 데 도움이 됩니다.

GROUPING 함수는 단일 열을 인수로 사용합니다. GROUPING 함수의 *expr*은 GROUP BY 절의 표현식 중 하나와 일치해야 합니다. 이 함수는 0 또는 값 1을 반환합니다.

GROUPING 함수에서 반환된 값은 다음 용도로 사용됩니다.

- 제공된 소계의 집계 레벨, 즉 소계의 기반이 되는 그룹을 결정합니다.
- 결과 집합의 행에 대한 표현식 열에 있는 NULL 값이 다음을 나타내는지 확인합니다.
  - 기본 테이블의 NULL 값(저장된 NULL 값)
  - ROLLUP 또는 CUBE로 생성한 NULL 값(해당 표현식의 그룹 함수의 결과)

표현식에 준하여 GROUPING 함수에 의해 반환된 값 0은 다음 중 하나를 나타냅니다.

- 집계 값을 계산하는 데 표현식이 사용되었습니다.
- 표현식 열의 NULL 값은 저장된 NULL 값입니다.

표현식에 준하여 GROUPING 함수에 의해 반환된 값 1은 다음 중 하나를 나타냅니다.

- 집계 값을 계산하는 데 표현식이 사용되지 않았습니다.
- 표현식 열의 NULL 값은 ROLLUP 또는 CUBE의 그룹화 결과 생성되었습니다.

## GROUPING 함수: 예제

```
SELECT    department_id DEPTID, job_id JOB,
          SUM(salary),
          GROUPING(department_id) GRP_DEPT,
          GROUPING(job_id) GRP_JOB
FROM      employees
WHERE     department_id < 50
GROUP BY  ROLLUP(department_id, job_id);
```

	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10	(null)	4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20	(null)	19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30	(null)	24900	0	1
9	40	HR_REP	6500	0	0
10	40	(null)	6500	0	1
11	(null)	(null)	54800	1	1

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### GROUPING 함수 예제

슬라이드 예제에서 첫번째 행의 요약 값 4400(레이블 1)을 살펴 보십시오. 이 요약 값은 부서 10 내의 직무 ID AD\_ASST에 대한 총 급여입니다. 이 요약 값을 계산하려면 DEPARTMENT\_ID와 JOB\_ID 열을 모두 고려해야 합니다. 따라서 GROUPING(department\_id) 및 GROUPING(job\_id) 표현식 모두에 대해 0 값이 반환됩니다.

두번째 행의 요약 값 4400을 살펴 보십시오(레이블 2). 이 값은 부서 10의 총 급여이며 DEPARTMENT\_ID 열을 고려하여 계산되었습니다. 따라서 GROUPING(department\_id)에 의해 0 값이 반환되었습니다. 이 값을 계산할 때 JOB\_ID 열은 고려되지 않았기 때문에 GROUPING(job\_id)에 대해 값 1이 반환되었습니다. 다섯번째 행에서 비슷한 결과를 볼 수 있습니다.

마지막 행에서는 요약 값 54800(레이블 3)을 살펴 보십시오. 이 값은 부서 ID가 50 보다 작은 부서 및 모든 직무에 대한 총 급여입니다. 이 요약 값을 계산하는 데 DEPARTMENT\_ID와 JOB\_ID 열을 모두 고려하지 않았습니다. 따라서 GROUPING(department\_id) 및 GROUPING(job\_id) 표현식 모두에 대해 값 1이 반환됩니다.

## GROUPING SETS

- GROUPING SETS 구문은 동일한 query에서 여러 개의 그룹화를 정의하는 데 사용됩니다.
- GROUPING SETS 절에 지정된 모든 그룹화를 계산하며 개별 그룹화 결과를 UNION ALL 연산과 결합합니다.
- 그룹화 집합의 효율성:
  - 기본 테이블에 대해 하나의 패스만 필요합니다.
  - 복잡한 UNION 문을 작성하지 않아도 됩니다.
  - GROUPING SETS 요소가 많을수록 성능이 향상됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### GROUPING SETS

GROUPING SETS는 GROUP BY 절이 추가로 확장된 것으로 데이터에 대해 여러 그룹화를 지정하는 데 사용할 수 있습니다. 이렇게 하면 효율적인 집계가 이루어지고 그에 따라 쉽게 여러 차원(Dimension)의 데이터를 분석할 수 있습니다.

이제 UNION ALL 연산자로 여러 SELECT 문을 결합하지 않고 GROUPING SETS를 사용하여 여러 그룹화(ROLLUP 또는 CUBE 연산자 포함 가능)를 지정하도록 단일 SELECT 문을 작성할 수 있습니다. 예를 들면 다음과 같습니다.

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY
GROUPING SETS
((department_id, job_id, manager_id),
 (department_id, manager_id),(job_id, manager_id));
```

이 명령문은 다음 세 그룹에 대한 집계를 계산합니다.

```
(department_id, job_id, manager_id), (department_id,
manager_id)and (job_id, manager_id)
```

이러한 기능이 없는 경우 위와 같은 SELECT 문의 출력 결과를 얻기 위해서는 여러 query를 UNION ALL과 결합해야 합니다. 다중 query 접근 방법은 동일한 데이터를 여러 번 스캔해야 하므로 비효율적입니다.

## GROUPING SETS(계속)

앞의 예제와 다음 방법을 비교해 보십시오.

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

이 명령문은 (department\_id, job\_id, manager\_id), (department\_id, manager\_id) 및 (job\_id, manager\_id) 그룹만 필요한 경우에도 8개(2\*2\*2)의 그룹화를 모두 계산합니다.

다음은 또 다른 대체 명령문입니다.

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

이 명령문은 기본 테이블을 세 번 스캔해야 하므로 비효율적입니다.

CUBE 및 ROLLUP은 매우 특수한 의미와 결과를 지닌 그룹화 집합으로 간주할 수 있습니다.

다음은 이러한 사실을 보여줍니다.

CUBE(a, b, c) 는 다음과 동일합니다.	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ( ))
ROLLUP(a, b, c) 는 다음과 동일합니다.	GROUPING SETS ((a, b, c), (a, b), (a), ( ))

## GROUPING SETS: 예제

```
SELECT    department_id, job_id,
          manager_id, AVG(salary)
FROM      employees
GROUP BY  GROUPING SETS
          ((department_id, job_id), (job_id, manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
1	(null)	SH_CLERK	122	3200
2	(null)	AC_MGR	101	12000
3	(null)	ST_MAN	100	7280
4	...	(null)	ST_CLERK	2675

1

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
39	110	AC_MGR	(null)	12000
40	90	AD_PRES	(null)	24000
41	60	IT_PROG	(null)	5760
42	100	FL_MGR	(null)	12000

2

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### GROUPING SETS: 예제

슬라이드의 query는 두 그룹에 대한 집계를 계산합니다. 테이블은 다음의 그룹으로 나뉩니다.

- 부서 ID, 직무 ID
- 직무 ID, 관리자 ID

이러한 각 그룹의 평균 급여가 계산됩니다. 결과 집합에는 두 그룹 각각에 대한 평균 급여가 표시됩니다.

출력에서 1로 표시된 그룹은 다음과 같이 해석될 수 있습니다.

- 관리자 122 휘하의 직무 ID가 SH\_CLERK인 모든 사원의 평균 급여는 3,200입니다.
  - 관리자 101 휘하의 직무 ID가 AC\_MGR인 모든 사원의 평균 급여는 12,000입니다.
- 나머지 행도 이러한 방식으로 해석됩니다.

출력에서 2로 표시된 그룹은 다음과 같이 해석됩니다.

- 부서 110에서 직무 ID가 AC\_MGR인 모든 사원의 평균 급여는 12,000입니다.
  - 부서 90에서 직무 ID가 AD\_PRES인 모든 사원의 평균 급여는 24,000입니다.
- 나머지 행도 이러한 방식으로 해석됩니다.

## GROUPING SETS: 예제(계속)

슬라이드의 예제를 다음과 같이 작성할 수도 있습니다.

```
SELECT department_id, job_id, NULL as manager_id,  
       AVG(salary) as AVGSAL  
FROM employees  
GROUP BY department_id, job_id  
UNION ALL  
SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL  
FROM employees  
GROUP BY job_id, manager_id;
```

query 블록을 찾아서 실행 계획을 생성하는 옵티마이저가 없을 경우 이전 query는 기본 테이블 EMPLOYEES를 두 번 스캔해야 합니다. 이는 매우 비효율적일 수 있습니다. 따라서 GROUPING SETS 문을 사용하는 것이 좋습니다.

## 조합 열

- 조합 열은 한 단위로 처리되는 열의 모음입니다.

`ROLLUP (a, (b, c), d)`

- GROUP BY 절에서 괄호를 사용하여 열을 그룹화합니다. 이렇게 하면 ROLLUP 또는 CUBE 연산을 계산할 때 괄호로 묶인 열이 하나의 단위로 처리됩니다.
- 조합 열을 ROLLUP 또는 CUBE와 함께 사용할 경우 특정 레벨에서 집계를 생략해야 합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 조합 열

조합 열은 그룹화 계산 중에 하나의 단위로 처리되는 열 모음입니다. `ROLLUP (a, (b, c), d)` 와 같이 열을 괄호 안에 지정하십시오.

여기서 (b, c)는 조합 열을 형성하고 하나의 단위로 처리됩니다. 일반적으로 조합 열은 ROLLUP, CUBE 및 GROUPING SETS에서 유용합니다. 예를 들어, CUBE 또는 ROLLUP에서 조합 열을 사용할 경우 특정 레벨에서 집계를 생략해야 합니다.

즉, `GROUP BY ROLLUP(a, (b, c))`는 다음과 같습니다.

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ( )
```

여기서 (b, c)는 하나의 단위로 처리되며 ROLLUP은 (b, c)에 적용되지 않습니다. 이것은 마치 (b, c)에 alias(예: z)가 있는 것과 같으며 GROUP BY 표현식은 `GROUP BY ROLLUP(a, z)`로 줄어듭니다.

**참고:** `GROUP BY ( )`는 일반적으로 a 및 b열에 대한 NULL 값과 집계 함수로만 구성된 SELECT 문입니다. 일반적으로 총계를 생성하는 데 사용됩니다.

```
SELECT NULL, NULL, aggregate_col
FROM <table_name>
GROUP BY ( );
```

## 조합 열(계속)

이 명령문을 다음과 같은 일반 ROLLUP과 비교해 보십시오.

```
GROUP BY ROLLUP(a, b, c)
```

이것은 다음을 의미합니다.

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

마찬가지로

```
GROUP BY CUBE((a, b), c)
```

이것은 다음과 동일합니다.

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP BY ()
```

다음 표는 GROUPING SETS 사양 및 해당 GROUP BY 사양을 보여줍니다.

GROUPING SETS 문	동일한 GROUP BY 문
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b, (b, c)) (GROUPING SETS 식은 조합 열을 갖습니다.)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS (a, ROLLUP(b, c)) (GROUPING SETS 식은 조합 열을 갖습니다.)	GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)



## 조합 열: 예제

```
SELECT    department_id, job_id, manager_id,
          SUM(salary)
FROM      employees
GROUP BY ROLLUP( department_id, (job_id, manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	(null)	(null)	(null)	7000
3	10	AD_ASST	101	4400
4	10	(null)	(null)	4400
5	20	MK_MAN	100	13000
6	20	MK_REP	201	6000
7	20	(null)	(null)	19000
...				
40	100	FI_MGR	101	12000
41	100	FI_ACCOUNT	108	39600
42	100	(null)	(null)	51600
43	110	AC_MGR	101	12000
44	110	AC_ACCOUNT	205	8300
45	110	(null)	(null)	20300
46	(null)	(null)	(null)	691400

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 조합 열: 예제

다음 예제를 살펴 보십시오.

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
GROUP BY ROLLUP( department_id, job_id, manager_id);
```

이 query에서 Oracle 서버는 다음 그룹화를 계산합니다.

- (job\_id, manager\_id)
- (department\_id, job\_id, manager\_id)
- (department\_id)
- 총계

특정 그룹에만 관심이 있는 경우 조합 열을 사용하지 않으면 이러한 그룹화로 계산을 제한할 수 없습니다. 조합 열의 경우에는 롤업 시 JOB\_ID 및 MANAGER\_ID 열을 하나의 단위로 처리함으로써 그룹화 제한이 가능합니다. 괄호로 묶인 열은 ROLLUP 및 CUBE 계산 시 하나의 단위로 처리됩니다. 이러한 내용이 슬라이드 예제에 설명되어 있습니다. JOB\_ID 열과 MANAGER\_ID 열을 괄호로 묶어 Oracle 서버에서 JOB\_ID 및 MANAGER\_ID를 단일 단위, 즉 조합 열로 처리하도록 지시하십시오.

## 조합 열: 예제(계속)

슬라이드의 예제는 다음 그룹화를 계산합니다.

- (department\_id, job\_id, manager\_id)
- (department\_id)
- ( )

슬라이드의 예제는 다음을 표시합니다.

- 모든 직무 및 관리자에 대한 총 급여(레이블 1)
- 모든 부서, 직무 및 관리자에 대한 총 급여(레이블 2)
- 모든 부서에 대한 총 급여(레이블 3)
- 총계(레이블 4)

슬라이드의 예제를 다음과 같이 작성할 수도 있습니다.

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM   employees
GROUP  BY department_id, job_id, manager_id
UNION  ALL
SELECT  department_id, TO_CHAR(NULL), TO_NUMBER(NULL),
        SUM(salary)
FROM    employees
GROUP BY department_id
UNION ALL
SELECT  TO_NUMBER(NULL), TO_CHAR(NULL), TO_NUMBER(NULL),
        SUM(salary)
FROM    employees
GROUP BY ( );
```

Query 블록을 찾아서 실행 계획을 생성하는 옵티마이저가 없을 경우 이전 query는 기본 테이블 EMPLOYEES를 세 번 스캔해야 합니다. 이는 매우 비효율적일 수 있습니다. 따라서 조합 열을 사용하는 것이 좋습니다.

## 연결된 그룹화

- 연결된 그룹화는 유용한 그룹화 조합을 생성하는 간단한 방법을 제공합니다.
- 연결된 그룹화 집합을 지정하려면 Oracle 서버에서 여러 그룹화 집합, ROLLUP 및 CUBE 연산을 단일 GROUP BY 절에 결합하도록 이들을 쉼표로 구분하십시오.
- 결과는 각 GROUPING SET에서 가져온 그룹화의 cross-product입니다.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연결된 그룹화

연결된 그룹화는 유용한 그룹화 조합을 생성하는 간단한 방법을 제공합니다. 여러 그룹화 집합, CUBE 및 ROLLUP을 나열하고 이들을 콤마로 구분하여 지정됩니다. 다음은 연결된 그룹화 집합의 예입니다.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

이 SQL 예제는 다음 그룹화를 정의합니다.

```
(a, c), (a, d), (b, c), (b, d)
```

그룹화 집합을 연결하면 다음과 같은 이점이 있습니다.

- **Query 개발의 용이성:** 모든 그룹화를 수동으로 열거하지 않아도 됩니다.
- **응용 프로그램에서 사용:** OLAP(Online Analytic Processing) 응용 프로그램에서 생성한 SQL에는 종종 그룹화 집합에 대한 연결이 포함됩니다. 각 GROUPING SET는 한 차원에 필요한 그룹화를 정의합니다.

## 연결된 그룹화: 예제

```
SELECT  department_id, job_id, manager_id,
        SUM(salary)
FROM    employees
GROUP BY department_id,
        ROLLUP(job_id),
        CUBE(manager_id);
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	10	AD_ASST	101	4400
3	20	MK_MAN	100	13000
4	20	MK_REP	201	6000
...				
1	90	AD_VP	100	34000
	90	AD_PRES	(null)	24000
...				
	(null)	SA_REP	(null)	7000
	10	AD_ASST	(null)	4400
...				
2	110	(null)	101	12000
	110	(null)	205	8300
	110	(null)	(null)	20300

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연결된 그룹화: 예제

슬라이드 예제를 실행하면 다음과 같은 그룹화가 생성됩니다.

- (department\_id, job\_id, ) (1)
- (department\_id, manager\_id) (2)
- (department\_id) (3)

각 그룹에 대한 총 급여가 계산됩니다.

다음은 연결된 그룹화의 또 다른 예입니다.

```
SELECT department_id, job_id, manager_id, SUM(salary) totalsal
FROM employees
WHERE department_id < 60
GROUP BY GROUPING SETS (department_id),
        GROUPING SETS (job_id, manager_id);
```

## 요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- ROLLUP 연산을 사용하여 소계 값 생성
- CUBE 연산을 사용하여 교차 분석 값 생성
- GROUPING 함수를 사용하여 ROLLUP 또는 CUBE에 의해 생성된 행 값 식별
- GROUPING SETS 구문을 사용하여 동일한 query에서 여러 그룹화 정의
- GROUP BY 절을 사용하여 다음과 같은 방법으로 표현식 결합
  - 조합 열
  - 연결된 그룹화 집합

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

- ROLLUP과 CUBE는 GROUP BY 절의 확장입니다.
- ROLLUP은 소계와 총계 값을 표시하는 데 사용됩니다.
- CUBE는 교차 분석 값을 표시하는 데 사용됩니다.
- GROUPING 함수를 사용하면 행이 CUBE 또는 ROLLUP 연산자로 생성된 집계인지 여부를 판별할 수 있습니다.
- GROUPING SETS 구문을 사용하여 동일한 query에서 여러 그룹화를 정의할 수 있습니다. GROUP BY는 지정된 그룹화를 모두 계산하고 이들을 UNION ALL과 결합합니다.
- 여러 가지 방법으로 GROUP BY 절 내에 표현식을 결합할 수 있습니다.
  - 조합 열을 지정하려면 열을 괄호로 묶어서 그룹화합니다. 이렇게 하면 Oracle 서버에서 ROLLUP 또는 CUBE 연산을 계산할 때 괄호로 묶인 열을 하나의 단위로 처리합니다.
  - 연결된 그룹화 집합을 지정하려면 여러 그룹화 집합, ROLLUP 및 CUBE 연산을 콤마로 구분하여 Oracle 서버가 이들을 단일 GROUP BY 절로 결합할 수 있도록 합니다. 결과는 각 그룹화 집합에서 가져온 그룹화의 cross-product입니다.





계층적 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 부록을 마치면 다음을 수행할 수 있습니다.**

- Hierarchical query의 개념 이해
- 트리 구조의 보고서 생성
- 계층 구조의 데이터 서식 지정
- 트리 구조에서 분기 제거

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 부록에서는 hierarchical query를 사용하여 트리 구조의 보고서를 생성하는 방법을 설명합니다.



## EMPLOYEES 테이블의 예제 데이터

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	100	King	AD_PRES	(null)
2	101	Kochhar	AD_VP	100
3	102	De Haan	AD_VP	100
4	103	Hunold	IT_PROG	102
5	104	Ernst	IT_PROG	103
6	107	Lorentz	IT_PROG	103

...

16	200	Whalen	AD_ASST	101
17	201	Hartstein	MK_MAN	100
18	202	Fay	MK_REP	201
19	205	Higgins	AC_MGR	101
20	206	Gietz	AC_ACCOUNT	205

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### EMPLOYEES 테이블의 예제 데이터

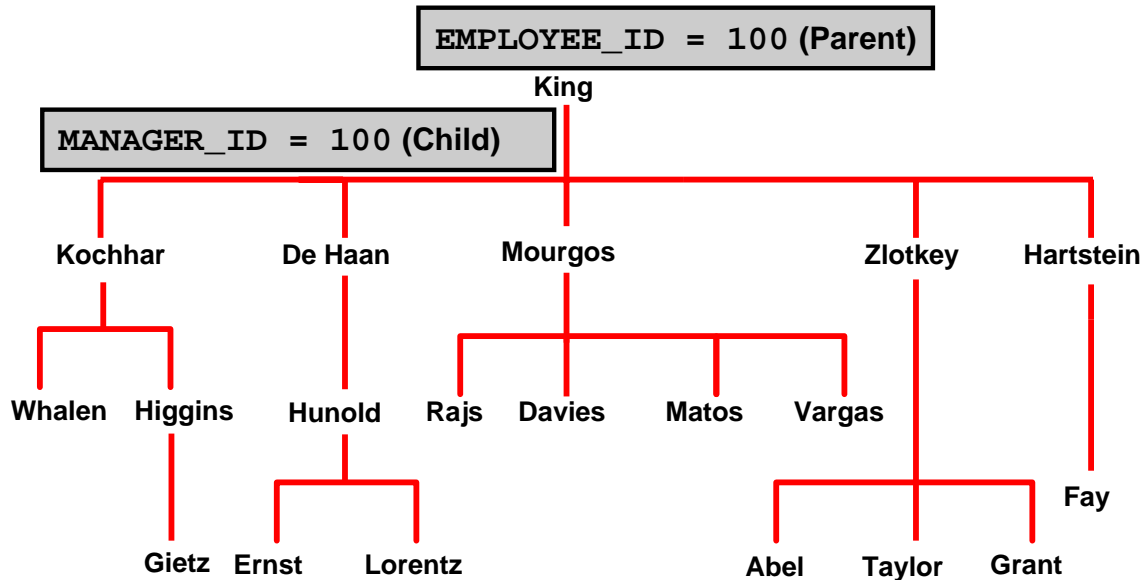
Hierarchical query를 사용하면 테이블에 있는 행 간의 자연적 계층 관계에 준하여 데이터를 검색할 수 있습니다. 관계형 데이터베이스는 레코드를 계층 방식으로 저장하지 않습니다. 그러나 한 테이블의 행 사이에 계층 관계가 존재하면 트리 탐색이라는 프로세스를 사용하여 계층을 구성할 수 있습니다. Hierarchical query는 보고의 한 방법으로, 특정 순서로 된 트리 분기가 있습니다.

가장 오래된 계열 멤버는 트리의 기부 또는 몸체 가까이에 있고 가장 최신 멤버는 트리의 분기를 나타내는 계열 트리를 가정해 봅시다. 분기에는 자체 분기가 있을 수 있습니다.

테이블의 행 사이에 관계가 있으면 hierarchical query가 가능합니다. 예를 들어, 위 슬라이드에서 Kochhar, De Haan, Hartstein은 MANAGER\_ID 100에게 보고하게 되어 있는데, 이 ID는 King의 EMPLOYEE\_ID입니다.

**참고:** 계층 트리는 계보(가계도), 가축류(품종 개량 목적), 기업 관리(관리 계층), 제조(제품 어셈블리), 진화론 연구(종의 진화), 과학 연구 등의 여러 분야에 사용됩니다.

## 일반 트리 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 일반 트리 구조

EMPLOYEES 테이블에는 관리 보고 라인을 나타내는 트리 구조가 있습니다. EMPLOYEE\_ID 열과 MANAGER\_ID 열의 해당 값 간의 관계를 보고 계층을 생성할 수 있습니다. 테이블을 자체에 조인하여 이 관계를 이용할 수 있습니다. MANAGER\_ID 열은 해당 사원의 관리자의 사원 번호를 포함합니다.

트리 구조의 상/하위 관계를 사용하여 다음을 제어할 수 있습니다.

- 계층 탐색 방향
- 계층 내의 시작점

**참고:** 슬라이드는 EMPLOYEES 테이블에 있는 사원 관리 계층의 역 트리 구조를 표시한 것입니다.

# Hierarchical Query

```
SELECT [LEVEL], column, expr...  
FROM table  
[WHERE condition(s)]  
[START WITH condition(s)]  
[CONNECT BY PRIOR condition(s)] ;
```

## 조건:

```
expr comparison_operator expr
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 키워드와 절

Hierarchical query는 CONNECT BY 절과 START WITH 절의 존재 여부로 식별할 수 있습니다. 이 구문에서 다음이 적용됩니다.

SELECT	표준 SELECT 절입니다.
LEVEL	hierarchical query에 의해 반환되는 각 행의 경우 LEVEL pseudocolumn은 루트 행에 대해 1을 반환하고 루트의 하위 행에 대해 2를 반환합니다(이하 같은 방식).
FROM table	열을 포함하는 테이블, 뷰 또는 스냅샷을 지정합니다. 한 테이블에서만 선택할 수 있습니다.
WHERE	계층 구조의 다른 행에 영향을 주지 않으면서 query에 의해 반환되는 행을 제한합니다.
condition	표현식을 사용한 비교입니다.
START WITH	계층의 루트 행(시작 위치)을 지정합니다. 이 절은 실제 hierarchical query에 필요합니다.
CONNECT BY	상위 및 하위 PRIOR 행 사이에 관계가 있는 열을 지정합니다. 이 절은 hierarchical query에 필요합니다.

# 트리 탐색

## 시작점

- 충족해야 하는 조건을 지정합니다.
- 유효한 조건을 받아들입니다.

```
START WITH column1 = value
```

EMPLOYEES 테이블을 사용하여 성이 Kochhar인 사원부터 시작합니다.

```
...START WITH last_name = 'Kochhar'
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 트리 탐색

트리의 루트로 사용할 행은 START WITH 절에 의해 결정됩니다. START WITH 절은 유효한 조건을 포함할 수 있습니다.

### 예제

EMPLOYEES 테이블을 사용하여 회사 대표인 King부터 시작합니다.

```
... START WITH manager_id IS NULL
```

EMPLOYEES 테이블을 사용하여 사원 Kochhar부터 시작합니다. START WITH 조건은 subquery를 포함할 수 있습니다.

```
... START WITH employee_id = (SELECT employee_id  
                                FROM employees  
                                WHERE last_name = 'Kochhar')
```

START WITH 절을 생략하면 테이블의 모든 행을 루트 행으로 간주하고 트리 탐색이 시작됩니다.

**참고:** CONNECT BY 및 START WITH 절은 ANSI(American National Standards Institute) SQL 표준이 아닙니다.

## 트리 탐색

```
CONNECT BY PRIOR column1 = column2
```

**EMPLOYEES 테이블을 사용하여 하향식으로 탐색합니다.**

```
... CONNECT BY PRIOR employee_id = manager_id
```

### 방향

하향식      →      Column1 = 상위 키  
                                Column2 = 하위 키

상향식      →      Column1 = 하위 키  
                                Column2 = 상위 키

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 트리 탐색

Query 방향은 CONNECT BY PRIOR 절의 위치에 의해 결정됩니다. 하향식의 경우 PRIOR 연산자를 상위 행을 참조하고, 하향식의 경우 PRIOR 연산자를 하위 행을 참조합니다. 상위 행의 하위 행을 찾기 위해 Oracle 서버는 상위 행에 대해 PRIOR 표현식을 평가하고 테이블의 각 행에 대해 다른 표현식을 평가합니다. 조건이 참인 행은 상위 행의 하위 행입니다. Oracle 서버는 항상 현재 상위 행에 대해 CONNECT BY 조건을 평가하여 하위 행을 선택합니다.

### 예제

EMPLOYEES 테이블을 사용하여 하향식으로 탐색합니다. 상위 행의 EMPLOYEE\_ID 값이 하위 행의 MANAGER\_ID 값과 같은 계층 관계를 정의합니다.

```
... CONNECT BY PRIOR employee_id = manager_id
```

EMPLOYEES 테이블을 사용하여 상향식으로 탐색합니다.

```
... CONNECT BY PRIOR manager_id = employee_id
```

PRIOR 연산자를 반드시 CONNECT BY 바로 뒤에 코딩할 필요는 없습니다. 따라서 다음 CONNECT BY PRIOR 절은 이전 예제의 절과 동일한 결과를 나타냅니다.

```
... CONNECT BY employee_id = PRIOR manager_id
```

**참고:** CONNECT BY 절은 subquery를 포함할 수 없습니다.

## 트리 탐색: 상향식

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START WITH employee_id = 101
CONNECT BY PRIOR manager_id = employee_id ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	101	Kochhar	AD_VP	100
2	100	King	AD_PRES	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 트리 탐색: 상향식

슬라이드의 예제는 사원 ID가 101인 사원부터 시작되는 관리자 리스트를 표시합니다.

## 트리 탐색: 하향식

```
SELECT last_name || ' reports to ' ||  
PRIOR last_name "Walk Top Down"  
FROM employees  
START WITH last_name = 'King'  
CONNECT BY PRIOR employee_id = manager_id ;
```

Walk Top Down
1 King reports to
2 King reports to
3 Kochhar reports to King
4 Greenberg reports to Kochhar
5 Fawiet reports to Greenberg

...

105 Grant reports to Zlotkey
106 Johnson reports to Zlotkey
107 Hartstein reports to King
108 Fay reports to Hartstein

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 트리 탐색: 하향식

하향식으로 탐색하면서 사원과 관리자의 이름을 표시합니다. King 사원을 시작점으로 사용하고 한 열만 인쇄합니다.

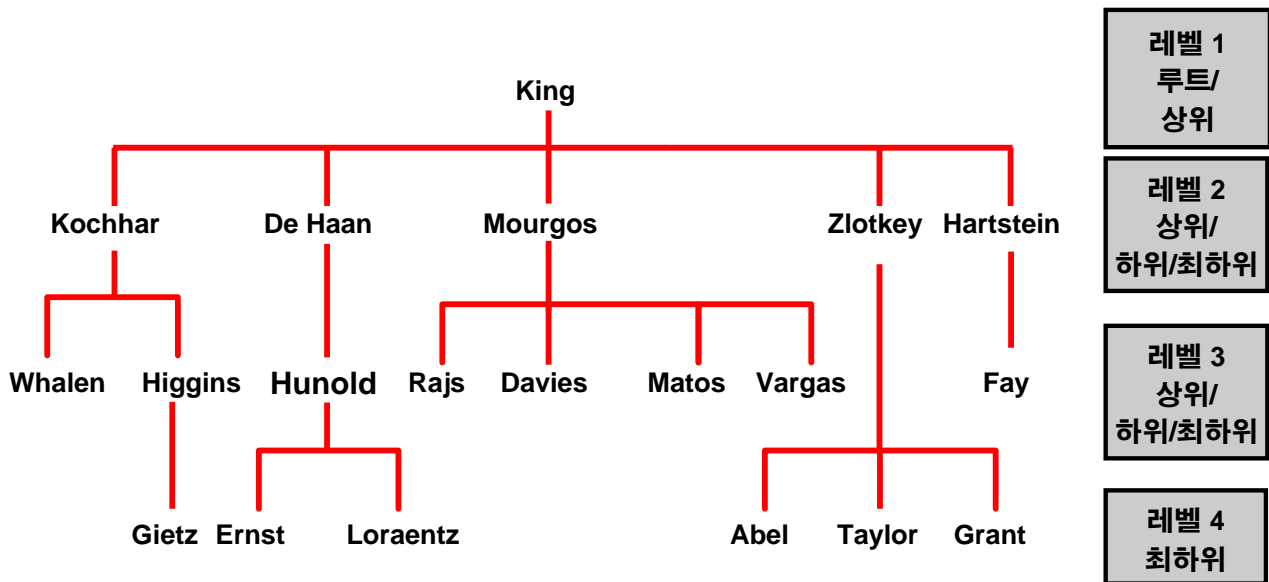
#### 예제

아래 예에서 EMPLOYEE\_ID 값은 상위 행과 MANAGER\_ID에 대해 평가되고 SALARY 값은 하위 행에 대해 평가됩니다. PRIOR 연산자는 EMPLOYEE\_ID 값에만 적용됩니다.

```
... CONNECT BY PRIOR employee_id = manager_id  
AND salary > 15000;
```

하위 행으로 지정하려면 상위 행의 EMPLOYEE\_ID 값과 동일한 MANAGER\_ID 값이 있어야 하고 \$15,000보다 큰 SALARY 값이 있어야 합니다.

## LEVEL 의사 열로 행 순위 지정



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### LEVEL 의사 열로 행 순위 지정

LEVEL pseudocolumn을 사용하여 계층 구조의 행 순위나 레벨을 명시적으로 표시할 수 있습니다. 이렇게 하면 보고서를 보다 쉽게 읽을 수 있습니다. 큰 분기에서 하나 이상의 분기가 나뉘는 분기를 노드라고 하며 분기 끝을 최하위 행 또는 최하위 노드라고 합니다. 슬라이드의 그래픽은 역 트리의 노드를 LEVEL 값과 함께 보여줍니다. 예를 들어, 사원 Higgins는 상위 노드인 동시에 하위 노드이고 사원 Davies는 하위 노드인 동시에 최하위 노드입니다.

#### LEVEL Pseudocolumn

값	하향식 레벨	상향식 레벨
1	루트 노드	루트 노드
2	루트 노드의 하위 노드	루트 노드의 상위 노드
3	하위 노드의 하위 노드 등	상위 노드의 상위 노드 등

슬라이드에서 King은 루트 또는 상위 노드입니다(LEVEL = 1). Kochhar, De Haan, Mourgos, Zlotkey, Hartstein, Higgins 및 Hunold는 하위 노드인 동시에 상위 노드입니다(LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant 및 Fay는 하위 노드인 동시에 최하위 노드입니다(LEVEL = 3 및 LEVEL = 4).

**참고:** 루트 노드는 역 트리 내의 최상위 노드입니다. 하위 노드는 루트가 아닌 노드입니다. 상위 노드는 하위 노드가 있는 노드입니다. 최하위 노드는 하위 노드가 없는 노드입니다. Hierarchical query에 의해 반환된 레벨 수는 사용 가능한 유저 메모리의 제한을 받습니다.



## LEVEL 및 LPAD를 사용하여 계층 보고서 형식 지정

최상위 레벨에서 시작하여 다음 레벨을 각각 들여쓰기한 형태의 회사 관리 레벨이 표시된 보고서를 생성합니다.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
       AS org_chart
FROM   employees
START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### LEVEL 및 LPAD를 사용하여 계층 보고서 형식 지정

트리의 노드에는 루트 기준의 레벨 수가 지정됩니다. LEVEL pseudocolumn과 함께 LPAD 함수를 사용하여 계층 보고서를 들여쓰기한 트리 구조로 표시합니다.

슬라이드의 예제는 다음과 같습니다.

- `LPAD(char1, n [, char2])`는 길이 `n`의 왼쪽을 `char2`의 문자 시퀀스로 채운 `char1`을 반환합니다. 인수 `n`은 터미널 화면에 표시되는 반환 값의 총 길이입니다.
- `LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')`는 표시 형식을 정의합니다.
- `char1`은 `LAST_NAME`이고, `n`은 반환 값의 총 길이로 `LAST_NAME + (LEVEL*2) - 2`이고, `char2`는 `'_'`입니다.

즉, 이 명령문은 SQL에게 `LAST_NAME`을 가져와서 결과 문자열의 길이가 `LENGTH(last_name) + (LEVEL*2) - 2`에 의해 결정된 값과 같아질 때까지 왼쪽을 `'_'` 문자로 채우도록 지시합니다.

King의 경우 `LEVEL = 1`이므로,  $(2 * 1) - 2 = 2 - 2 = 0$ 입니다. 따라서 King은 `'_'` 문자로 채워지지 않고 1열에 표시됩니다.

Kochhar의 경우 `LEVEL = 2`이므로  $(2 * 2) - 2 = 4 - 2 = 2$ 입니다. 따라서 Kochhar는 두 개의 `'_'` 문자로 채워지고 들여쓰기한 상태로 표시됩니다.

EMPLOYEES 테이블에서 레코드의 나머지 부분은 비슷하게 표시됩니다.

**LEVEL 및 LPAD를 사용하여 계층 보고서 서식 지정(계속)**

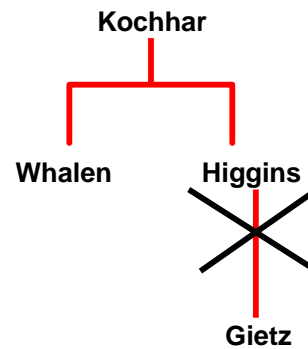
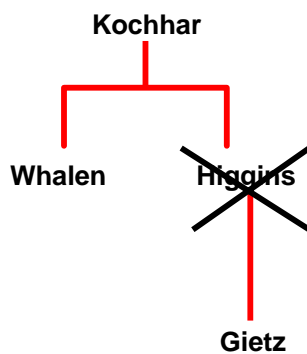
R2 ORG_CHART	
1	King
2	__Kochhar
3	___Greenberg
4	____Faviet
5	_____Chen
6	_____Sciarra
7	_____Urman
8	_____Popp
9	_____Whalen
10	_____Mavris
11	_____Baer
12	_____Higgins
13	_____Gietz
14	__De Haan
15	___Hunold
16	____Ernst
17	_____Austin

# 분기 제거

WHERE 절을 사용하여  
노드를 제거합니다.

CONNECT BY 절을 사용하여  
분기를 제거합니다.

```
WHERE last_name != 'Higgins' CONNECT BY PRIOR  
employee_id = manager_id  
AND last_name != 'Higgins'
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 분기 제거

WHERE 절과 CONNECT BY 절을 사용하여 트리를 제거합니다. 즉, 표시할 노드 또는 행을 제어합니다. 사용하는 술어는 부울 조건으로 동작합니다.

### 예제

루트에서 시작하여 하향식으로 탐색하고 결과에서 Higgins 사원은 제거하고 하위 행은 처리합니다.

```
SELECT department_id, employee_id, last_name, job_id, salary  
FROM employees  
WHERE last_name != 'Higgins'  
START WITH manager_id IS NULL  
CONNECT BY PRIOR employee_id = manager_id;
```

루트에서 시작하여 하향식으로 탐색하고 Higgins 사원과 하위 행을 모두 제거합니다.

```
SELECT department_id, employee_id, last_name, job_id, salary  
FROM employees  
START WITH manager_id IS NULL  
CONNECT BY PRIOR employee_id = manager_id  
AND last_name != 'Higgins';
```

## 요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- Hierarchical query를 사용하여 테이블의 행 사이의 계층 관계 확인
- Query 방향 및 시작점 지정
- 노드 또는 분기 제거

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

Hierarchical query를 사용하여 테이블에 있는 행 사이의 계층 관계에 준하여 데이터를 검색할 수 있습니다. LEVEL 의사 열은 탐색한 계층 트리의 레벨을 계산합니다. CONNECT BY PRIOR 절을 사용하여 query 방향을 지정할 수 있습니다. START WITH 절을 사용하여 시작점을 지정할 수 있습니다. WHERE 절과 CONNECT BY 절을 사용하여 트리 분기를 제거할 수 있습니다.

# III

## 고급 스크립트 작성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 부록을 마치면 다음을 수행할 수 있습니다.**

- **SQL을 사용하여 SQL을 생성함으로써 해결되는 문제 유형 설명**
- **DROP TABLE 문의 스크립트를 생성하는 스크립트 작성**
- **INSERT INTO 문의 스크립트를 생성하는 스크립트 작성**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 부록에서는 SQL 스크립트를 생성하는 SQL 스크립트 작성 방법을 설명합니다.

## SQL을 사용하여 SQL 생성

- SQL은 SQL 언어로 된 스크립트를 생성하는 데 사용됩니다.
- 데이터 디렉터리의 특성은 다음과 같습니다.
  - 데이터베이스 정보를 포함하는 테이블 및 뷰의 모음입니다.
  - Oracle 서버에 의해 생성 및 유지 관리됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL을 사용하여 SQL 생성

SQL은 다른 SQL 문을 생성하기 위한 강력한 도구가 될 수 있습니다. 대부분의 경우 SQL은 스크립트 파일 작성과 관련이 있습니다. SQL에서 SQL을 사용하여 다음을 수행할 수 있습니다.

- 반복 코딩을 피함
- 데이터 디렉터리의 정보 액세스
- 데이터베이스 객체 삭제 또는 재생성
- 런타임 파라미터를 포함하는 동적 SQL 생성

이 부록에 사용된 예제에는 데이터 디렉터리의 정보를 선택하는 과정이 포함되어 있습니다.

데이터 디렉터리란 데이터베이스에 대한 정보를 포함하는 테이블 및 뷰의 모음입니다. 이 모음은 Oracle 서버에 의해 생성 및 유지 관리됩니다. 모든 데이터 디렉터리 테이블은 SYS 사용자가 소유합니다. 데이터 디렉터리에 저장되는 정보에는 Oracle 서버 유저의 이름, 유저에게 부여된 권한, 데이터베이스 객체 이름, 테이블 제약 조건 및 감사(audit) 정보가 포함됩니다. 데이터 디렉터리 뷰에는 네 가지 범주가 있습니다. 범주마다 의도된 용도를 명확하게 반영하는 접두어가 있습니다.

## SQL을 사용하여 SQL 생성(계속)

접두어	설명
USER_	유저가 소유한 객체의 세부 정보를 포함합니다.
ALL_	유저가 소유한 객체 외에도 유저에게 액세스 권한이 부여된 객체의 세부 정보를 포함합니다.
DBA_	데이터베이스에 있는 객체에 액세스할 수 있는 DBA 권한을 가진 유저의 세부 정보를 포함합니다.
V\$_	데이터베이스 서버 성능 및 잠금에 대한 정보를 저장하며 DBA만 사용할 수 있습니다.



## 기본 스크립트 작성

```
SELECT 'CREATE TABLE ' || table_name ||  
      '_test ' || 'AS SELECT * FROM '  
      || table_name || ' WHERE 1=2;'  
      AS "Create Table Script"  
FROM   user_tables;
```

1	Create Table Script
1	CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;
2	CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
3	CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
4	CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
5	CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
6	CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 기본 스크립트

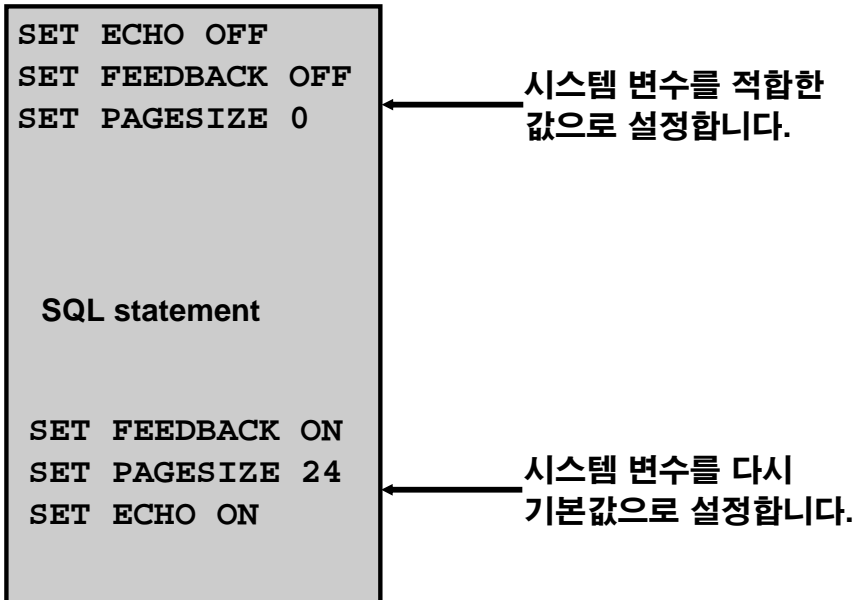
슬라이드의 예제는 사용자가 소유한 모든 테이블에서 CREATE TABLE 문을 사용하는 보고서를 생성합니다. 보고서에 생성된 각 CREATE TABLE 문의 구문은 접미어가 \_test인 테이블 이름을 사용하고 해당되는 기존 테이블에서 구조만 취하여 테이블을 생성합니다. 이전 테이블 이름은 데이터 디렉터리 뷰 USER\_TABLES의 TABLE\_NAME 열에서 가져옵니다.

다음 단계는 프로세스를 자동화할 수 있도록 보고서를 향상시키는 것입니다.

**참고:** 데이터 디렉터리 테이블을 query하여 자신이 소유한 다양한 데이터베이스 객체를 조회할 수 있습니다. 자주 사용되는 데이터 디렉터리 뷰는 다음과 같습니다.

- USER\_TABLES: 사용자가 소유한 테이블에 대한 설명을 표시합니다.
- USER\_OBJECTS: 사용자가 소유한 모든 객체를 표시합니다.
- USER\_TAB\_PRIVS\_MADE: 사용자가 소유한 객체에 대해 부여된 모든 권한을 표시합니다.
- USER\_COL\_PRIVS\_MADE: 사용자가 소유한 객체의 열에 대해 부여된 모든 권한을 표시합니다.

## 환경 제어



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 환경 제어

생성된 SQL 문을 실행하려면 SQL 문을 파일로 캡처한 다음 해당 파일을 실행해야 합니다. 또한 생성되는 출력을 정리할 계획을 세워서 머리글, 피드백 메시지, 상단 제목 등의 요소가 출력되지 않도록 해야 합니다. SQL Developer에서는 이러한 명령문을 스크립트로 저장할 수 있습니다.

Enter SQL Statement 상자의 내용을 저장하려면 Save 아이콘을 누르거나 **File > Save** 메뉴 항목을 사용하십시오. 또는 Enter SQL Statement 상자를 마우스 오른쪽 버튼으로 누른 다음 드롭다운 메뉴에서 Save File 옵션을 선택하십시오.

**참고:** 일부 SQL\*Plus 문은 SQL Worksheet에서 지원되지 않습니다. 지원되는 SQL\*Plus 문과 SQL Worksheet에서 지원하지 않는 SQL\*Plus 문의 전체 리스트는 SQL Developer 온라인 도움말에서 *SQL\*Plus Statements Supported and Not Supported in SQL Worksheet* 항목을 참조하십시오.

## 전체 그림

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM   user_objects
WHERE  object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 전체 그림

위 슬라이드의 명령 출력은 SQL Developer에서 dropem.sql 파일에 저장됩니다. SQL Developer에서 출력을 파일로 저장하려면 Script Output 창 아래에 있는 Save File 옵션을 사용하십시오. dropem.sql 파일은 다음 데이터를 포함합니다. 이 파일은 이제 스크립트 파일을 찾아 로드하고 실행하여 SQL Developer에서 시작될 수 있습니다.

	'DROPTABLE'  OBJECT_NAME  ';'
1	DROP TABLE REGIONS;
2	DROP TABLE COUNTRIES;
3	DROP TABLE LOCATIONS;
4	DROP TABLE DEPARTMENTS;
5	DROP TABLE JOBS;
6	DROP TABLE EMPLOYEES;
7	DROP TABLE JOB_HISTORY;
8	DROP TABLE JOB_GRADES;

## 테이블 내용을 파일로 덤프

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0

SELECT
  'INSERT INTO departments_test VALUES
  (' || department_id || ', ' || department_name ||
  ', ' || location_id || ');'
  AS "Insert Statements Script"
FROM departments
/

SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 테이블 내용을 파일로 덤프

테이블 행에 대한 값을 텍스트 파일에 INSERT INTO VALUES 문 형식으로 저장해 놓으면 유용하게 사용할 수 있습니다. 테이블이 실수로 삭제되었을 경우 이 스크립트를 실행하여 테이블을 채울 수 있습니다.

위 슬라이드의 예제는 SQL Developer에서 Save File 옵션을 사용하여 data.sql 파일에 캡처된, DEPARTMENTS\_TEST 테이블에 대한 INSERT 문을 생성합니다.

data.sql 스크립트 파일의 내용은 다음과 같습니다.

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

## 테이블 내용을 파일로 덤프

소스	결과
<code>'X'</code>	<code>'X'</code>
<code>''</code>	<code>'</code>
<code>''    department_name    ''</code>	<code>'Administration'</code>
<code>','</code>	<code>','</code>
<code>');'</code>	<code>');'</code>

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 테이블 내용을 파일로 덤프(계속)

이전 슬라이드에서 작은 따옴표를 많이 볼 수 있었습니다. 네 개의 작은 따옴표로 구성된 한 집합은 최종 명령문에서 하나의 작은 따옴표로 나타납니다. 또한 문자 및 날짜 값은 따옴표로 묶어야 합니다.

한 문자열 내에서 하나의 작은 따옴표를 표시하려면, 그 앞에 또 다른 작은 따옴표를 붙여야 합니다. 예를 들어 슬라이드의 다섯번째 예제에서, 바깥을 둘러싼 따옴표는 전체 문자열에 대한 따옴표입니다. 두번째 따옴표는 세번째 따옴표를 표시하기 위한 접두어 역할을 합니다. 따라서 결과적으로 하나의 작은 따옴표 다음에 괄호가 오고 그 다음에 세미콜론이 옵니다.

## 동적 슬어 생성

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&deptno', null,
DECODE ('&hiredate', null, ' ',
'WHERE hire_date=TO_DATE('' || '&hiredate'', 'DD-MON-YYYY'')),
DECODE ('&hiredate', null,
'WHERE department_id = ' || '&deptno',
'WHERE department_id = ' || '&deptno' ||
' AND hire_date = TO_DATE('' || '&hiredate'', 'DD-MON-YYYY''))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 동적 슬어 생성

위 슬라이드의 예제는 부서에서 특정일에 채용된 모든 사원에 대한 데이터를 검색하는 SELECT 문을 생성합니다. 스크립트는 WHERE 절을 동적으로 생성합니다.

**참고:** 일단 유저 변수가 나타나면 UNDEFINE 명령을 사용하여 변수를 삭제해야 합니다.

첫번째 SELECT 문은 유저에게 부서 번호 입력을 요청합니다. 부서 번호를 입력하지 않을 경우 DECODE 함수에 의해 부서 번호는 널로 취급되며, 유저는 다음으로 채용 날짜 입력을 요청 받습니다. 채용 날짜를 입력하지 않을 경우 채용 날짜는 DECODE 함수에 의해 널로 처리되어 생성되는 동적 WHERE 절도 널이 되며, 그 결과 두번째 SELECT 문은 EMPLOYEES 테이블에서 모든 행을 검색하게 됩니다.

**참고:** NEW\_V[ALUE] 변수는 열 값을 보유하기 위한 변수를 지정합니다. TTITLE 명령의 변수를 참조할 수 있습니다. NEW\_VALUE를 사용하여 최상위 제목에 열 값 또는 날짜를 표시합니다. SKIP PAGE 작업을 사용하여 BREAK 명령에 열을 포함시켜야 합니다. 변수 이름에는 파운드 기호(#)가 포함될 수 없습니다. NEW\_VALUE는 각 페이지에 대한 새 마스터 레코드가 있는 마스터/상세 보고서에 유용합니다.

## 동적 술어 생성(계속)

**참고:** 여기에서 채용 날짜는 DD-MON-YYYY 형식으로 입력해야 합니다.

슬라이드의 SELECT 문을 다음과 같이 해석할 수 있습니다.

```

IF    (<<deptno>> is not entered) THEN
    IF (<<hiredate>> is not entered) THEN
        return empty string
    ELSE
        return the string 'WHERE hire_date =
TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
    ELSE
        IF (<<hiredate>> is not entered) THEN
            return the string 'WHERE department_id =
<<deptno>> entered'
        ELSE
            return the string 'WHERE department_id =
<<deptno>> entered
                                AND hire_date =
TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
        END IF

```

반환된 문자열은 DYN\_WHERE\_CLAUSE 변수의 값이 되고, 이 값은 두번째 SELECT 문에 사용됩니다.

**참고:** 이 예제에는 SQL\*Plus를 사용하십시오.

슬라이드의 첫번째 예제가 실행되면 유저는 DEPTNO 및 HIREDATE에 대한 값을 입력해야 합니다.

Enter value for deptno: 10

Enter value for hiredate: 17-SEP-1987

다음과 같은 MY\_COL 값이 생성됩니다.

```

MY_COL
-----
WHERE department_id = 10 AND hire_date = TO_DATE('27-SEP-1987','DD-MON-YYYY')

```

슬라이드의 두번째 예제가 실행되면 다음과 같은 출력이 생성됩니다.

```

LAST_NAME
-----
Whalen

```

## 요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- SQL 스크립트를 작성하여 다른 SQL 스크립트 생성
- 스크립트 파일을 데이터 디렉터리로 사용
- 출력을 파일로 캡처

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

SQL은 SQL 스크립트를 생성하는 데 사용될 수 있습니다. 이러한 스크립트는 반복 코딩을 피하고, 객체를 삭제하거나 재생성하며, 데이터 디렉터리에서 도움말을 얻거나 런타임 파라미터를 포함하는 동적 SQL을 생성하는 데 사용될 수 있습니다.



# 오라클 데이터베이스 구조 구성 요소



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 부록을 마치면 다음을 수행할 수 있습니다.**

- **주요 데이터베이스 구조 구성 요소 나열**
- **백그라운드 프로세스 설명**
- **메모리 구조 설명**
- **논리적/물리적 저장 영역 구조 상호 연관**



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 오라클 데이터베이스 구조에 대해 간략히 설명합니다. 또한 오라클 데이터베이스의 물리적/논리적 구조를 살펴보고 다양한 구성 요소와 해당 기능에 대해 배웁니다.

# 오라클 데이터베이스 구조: 개요

Oracle RDBMS(관계형 데이터베이스 관리 시스템)는 정보를 관리하는 데 있어 개방적이고 종합적이며 통합적인 접근 방식을 제공하는 데이터베이스 관리 시스템입니다.



ORACLE

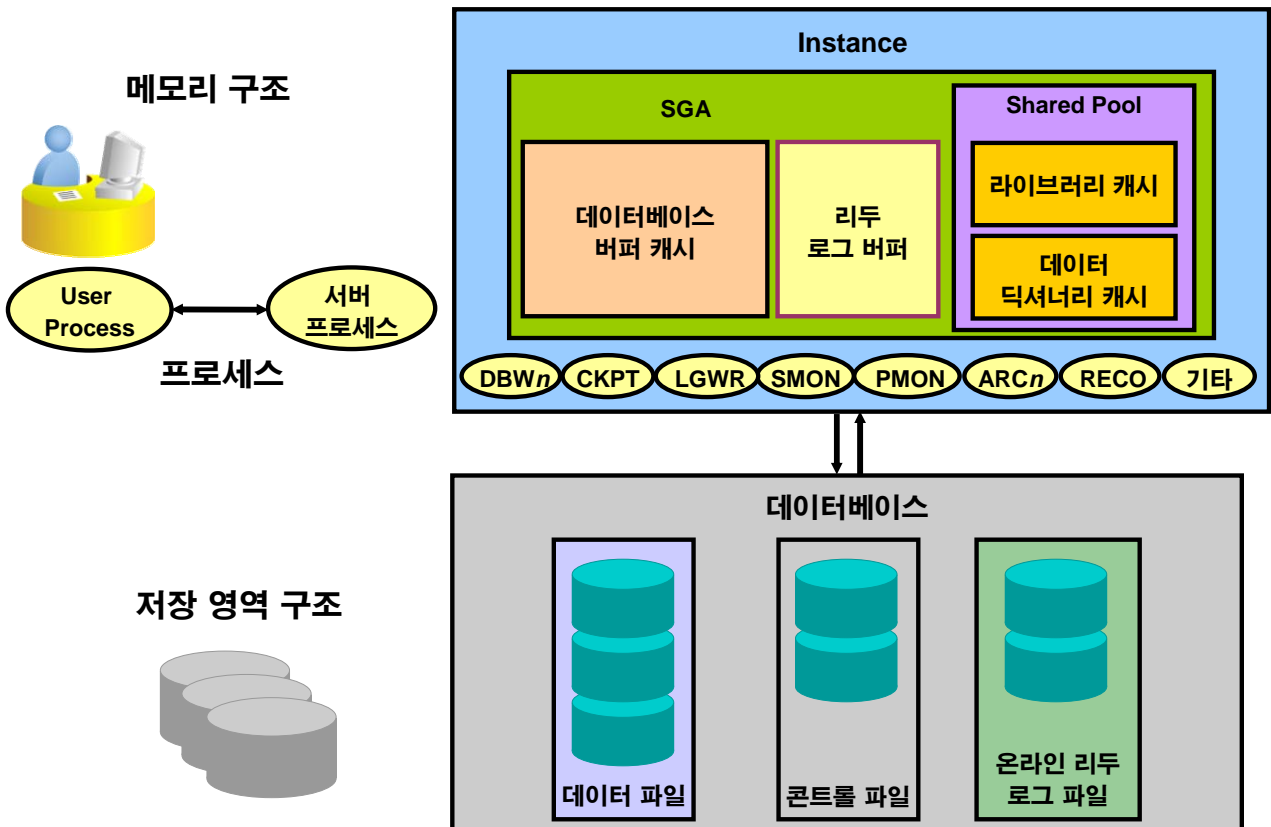
Copyright © 2009, Oracle. All rights reserved.

## 오라클 데이터베이스 구조: 개요

데이터베이스는 하나의 단위로 취급되는 데이터 모음입니다. 데이터베이스의 목적은 관련 정보를 저장하고 검색하는 것입니다.

오라클 데이터베이스는 다중 유저 환경에서 대량의 데이터를 안정적으로 관리하여 다수의 유저가 동일한 데이터에 동시에 액세스할 수 있도록 합니다. 단, 이 기능은 높은 성능을 제공해야 가능합니다. 동시에 오라클 데이터베이스는 무단 액세스를 차단하고 failure recovery에 대한 효과적인 해결책을 제공합니다.

# 오라클 데이터베이스 서버 구조



Copyright © 2009, Oracle. All rights reserved.

## 오라클 데이터베이스 서버 구조

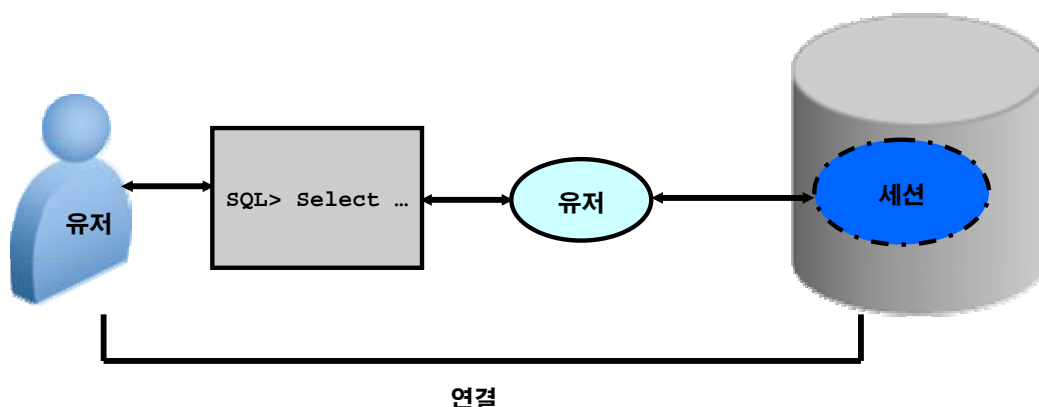
오라클 데이터베이스는 두 가지 주요 구성 요소인 instance와 데이터베이스로 구성됩니다.

- Instance는 메모리 구조인 시스템 글로벌 영역(SGA)과 데이터베이스 내에서 작업을 수행하는 백그라운드 프로세스로 구성됩니다. Instance가 시작될 때마다 SGA가 할당되고 백그라운드 프로세스가 시작됩니다.
- 데이터베이스는 논리적 구조와 물리적 구조로 구성됩니다. 논리적 구조와 물리적 구조는 개별적이므로 논리적 저장 영역 구조에 대한 액세스에 영향을 주지 않고 데이터의 물리적 저장 영역을 관리할 수 있습니다. 물리적 저장 영역 구조는 다음과 같습니다.
  - 데이터베이스 구성이 저장되는 컨트롤 파일
  - 데이터베이스 recovery에 필요한 정보를 포함하는 리두 로그 파일
  - 모든 데이터가 저장되는 데이터 파일

Oracle instance는 메모리 구조와 프로세스를 사용하여 데이터베이스 저장 영역 구조를 관리하고 액세스합니다. 모든 메모리 구조는 데이터베이스 서버를 구성하는 컴퓨터의 기본 메모리에 존재합니다. 프로세스는 이러한 컴퓨터의 메모리에서 작동하는 작업입니다. 프로세스는 일련의 단계를 실행할 수 있는 운영 체제의 "제어 스레드" 또는 메커니즘으로 정의할 수 있습니다.

## 데이터베이스에 연결

- **연결:** User process와 데이터베이스 instance 사이의 통신 경로
- **세션:** User process를 통해 이루어지는 데이터베이스 instance에 대한 특정 유저 연결



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 데이터베이스에 연결

데이터베이스의 정보에 액세스하기 위해서는 유저가 도구(예: SQL\*Plus)를 사용하여 데이터베이스에 연결해야 합니다. 유저가 연결을 설정하면 해당 유저에 대한 세션이 생성됩니다. 연결과 세션은 user process와 밀접하게 관련되어 있지만 의미는 매우 다릅니다.

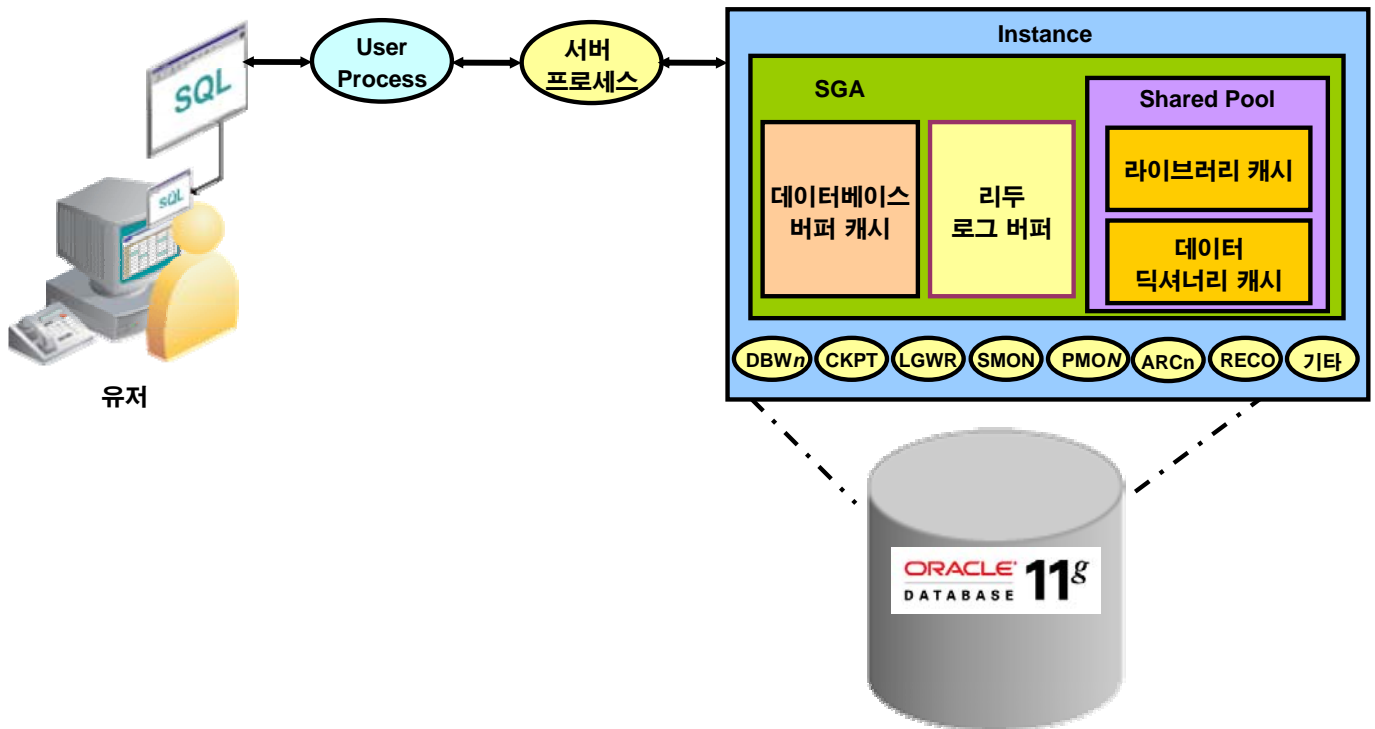
연결은 user process와 오라클 데이터베이스 instance 사이의 통신 경로입니다. 통신 경로는 사용 가능한 프로세스 간 통신 메커니즘 또는 네트워크 소프트웨어(서로 다른 컴퓨터에서 응용 프로그램과 오라클 데이터를 실행하며 네트워크를 통해 통신하는 경우)를 사용하여 설정됩니다.

세션은 데이터베이스 instance에 대한 현재 유저 로그인 상태를 나타냅니다. 예를 들어, 유저가 SQL\*Plus를 시작하는 경우 유효한 username과 암호를 제공해야 합니다. 그러면 해당 유저에 대한 세션이 설정됩니다. 세션은 유저가 연결된 때부터 연결을 끊거나 데이터베이스 응용 프로그램을 종료할 때까지 지속됩니다.

전용 연결의 경우 영구 전용 프로세스에서 세션을 처리하고, 공유 연결의 경우 풀에서 선택된 사용 가능한 서버 프로세스, 즉 middle-tier나 Oracle Shared Server 구조에서 세션을 처리합니다.

동일한 username을 사용하여 한 명의 오라클 데이터베이스 유저에 대해 여러 세션을 생성할 수 있으며 이러한 여러 세션이 동시에 존재할 수 있습니다. 단, 다른 응용 프로그램이나 동일한 응용 프로그램의 다중 호출을 통해서만 가능합니다.

## 오라클 데이터베이스와 상호 작용



Copyright © 2009, Oracle. All rights reserved.

### 오라클 데이터베이스와 상호 작용

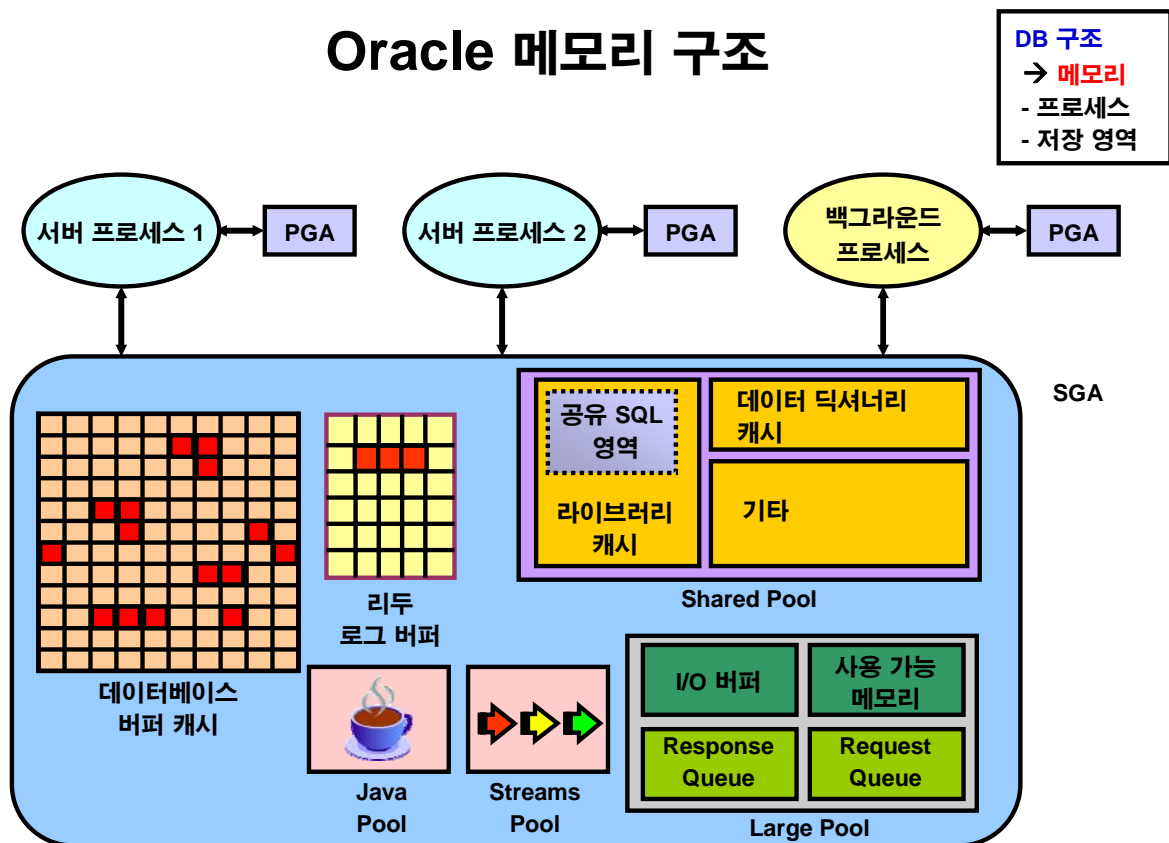
다음 예는 가장 기본적인 레벨의 오라클 데이터베이스 작업에 대해 설명하며, User process와 연관된 서버 프로세스가 네트워크로 연결된 별도의 컴퓨터에 있는 오라클 데이터베이스 구성을 보여줍니다.

1. 오라클 데이터베이스가 설치된 노드(호스트 또는 데이터베이스 서버라고도 함)에서 instance가 시작되었습니다.
2. 유저가 user process를 생성하는 응용 프로그램을 시작합니다. 응용 프로그램에서 서버에 대한 연결을 설정하려고 시도합니다. 로컬 또는 클라이언트 서버 연결이거나 middle-tier로부터의 3계층 연결일 수 있습니다.
3. 서버에서 적절한 Oracle Net Services 처리기가 있는 리스너를 실행합니다. 서버에서 응용 프로그램의 연결 요청을 감지하고 user process 대신 dedicated server 프로세스를 생성합니다.
4. 유저가 DML 유형 SQL 문을 실행하고 트랜잭션을 커밋합니다. 예를 들어, 유저가 테이블에서 고객의 주소를 변경하고 변경 내용을 커밋합니다.
5. 서버 프로세스에서 해당 명령문을 받아 shared pool(SGA 구성 요소)에 유사한 SQL 문이 포함된 공유 SQL 영역이 있는지 검사합니다. 공유 SQL 영역이 있으면 서버 프로세스는 요청된 데이터에 대한 유저의 액세스 권한을 확인하고 기존 공유 SQL 영역이 명령문 처리에 사용됩니다. 공유 SQL 영역이 없으면 명령문이 구문 분석되고 처리될 수 있도록 해당 명령문에 대해 새 공유 SQL 영역이 할당됩니다.

## 오라클 데이터베이스와 상호 작용(계속)

6. 서버 프로세스가 실제 데이터 파일(테이블이 저장됨)이나 SGA에 캐시된 데이터 파일에서 필요한 데이터 값을 검색합니다.
7. 서버 프로세스가 SGA의 데이터를 수정합니다. 트랜잭션이 커밋되었으므로 LGWR(로그 기록자 프로세스)이 즉시 트랜잭션을 리두 로그 파일에 기록합니다. DBWn(데이터베이스 기록자 프로세스)이 수정된 블록을 디스크에 영구적으로 기록합니다(이렇게 하는 것이 효율적인 경우).
8. 트랜잭션이 성공하면 서버 프로세스가 네트워크를 통해 메시지를 응용 프로그램으로 보냅니다. 그리고 트랜잭션이 실패하면 오류 메시지가 전송됩니다.
9. 이러한 전체 과정에서 개입이 필요한 상황이 되면 다른 백그라운드 프로세스가 실행됩니다. 또한 데이터베이스 서버는 다른 유저의 트랜잭션을 관리하며 동일한 데이터를 요청하는 트랜잭션 간의 경합을 방지합니다.

# Oracle 메모리 구조



Copyright © 2009, Oracle. All rights reserved.

## Oracle 메모리 구조

오라클 데이터베이스는 여러 가지 목적을 위해 메모리 구조를 생성하여 사용합니다. 예를 들어, 메모리에는 실행 중인 프로그램 코드, 유저 공유 데이터 및 연결된 개별 유저에 대한 전용 데이터 영역이 저장됩니다. Instance와 연관된 두 가지 기본 메모리 구조는 다음과 같습니다.

- 시스템 글로벌 영역(SGA) - SGA 구성 요소라고 하는 공유 메모리 구조의 그룹으로 한 개의 오라클 데이터베이스 instance에 대한 데이터 및 제어 정보를 포함합니다. SGA는 모든 서버 프로세스와 백그라운드 프로세스에서 공유합니다. SGA에 저장된 데이터의 예로는 캐시에 저장된 데이터 블록 및 공유 SQL 영역이 있습니다.
- 프로그램 글로벌 영역(PGA) - 서버 프로세스 또는 백그라운드 프로세스에 대한 데이터 및 제어 정보를 포함하는 메모리 영역입니다. PGA는 서버 프로세스 또는 백그라운드 프로세스가 시작될 때 오라클 데이터베이스에서 생성하는 비공유 메모리입니다. 서버 프로세스만 PGA에 액세스할 수 있습니다. 서버 프로세스와 백그라운드 프로세스는 각각 자체의 PGA를 갖습니다.



## Oracle 메모리 구조(계속)

SGA는 instance에 대한 데이터 및 제어 정보를 포함하는 메모리 영역입니다. SGA는 다음 데이터 구조를 포함합니다.

- **데이터베이스 버퍼 캐시:** 데이터베이스에서 검색된 데이터 블록을 캐시합니다.
- **리두 로그 버퍼:** instance recovery에 사용되는 리두 정보가 디스크에 저장된 물리적 리두 로그 파일에 기록될 때까지 해당 정보를 캐시합니다.
- **Shared Pool:** 유저 간에 공유할 수 있는 다양한 생성자를 캐시합니다.
- **Large Pool:** Oracle 백업 및 recovery 작업이나 입/출력(I/O) 서버 프로세스와 같은 특정 대규모 프로세스에 대한 대용량 메모리 할당을 제공하는 선택적 영역입니다.
- **Java Pool:** JVM(Java Virtual Machine) 내의 모든 세션별 Java 코드 및 데이터에 사용됩니다.
- **Streams Pool:** Oracle Streams에서 캡처 및 적용에 필요한 정보를 저장하는 데 사용됩니다.

Enterprise Manager 또는 SQL\*Plus를 사용하여 instance를 시작하면 SGA에 대해 할당된 메모리 양이 표시됩니다.

동적 SGA infrastructure를 사용하면 instance를 종료하지 않고 데이터베이스 버퍼 캐시, shared pool, large pool, Java pool 및 streams pool의 크기를 변경할 수 있습니다.

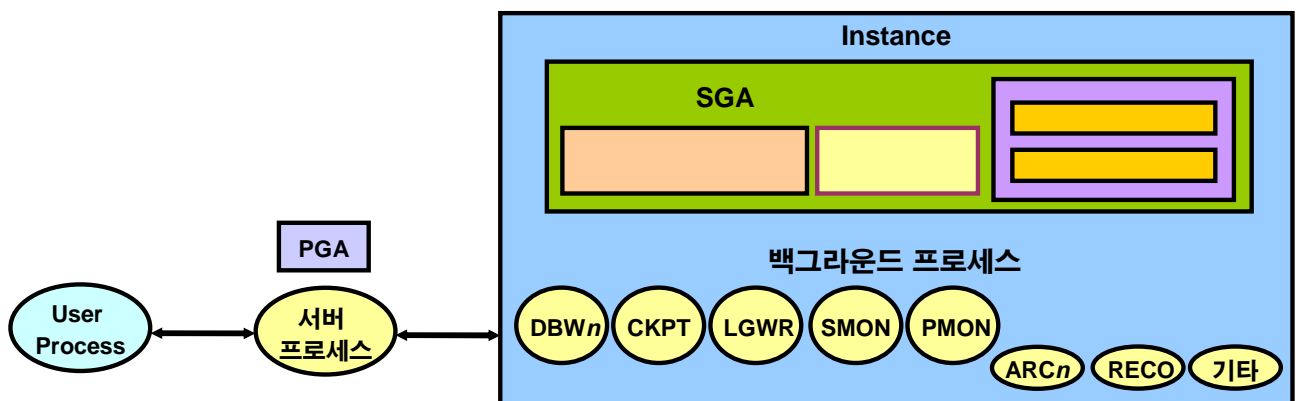
오라클 데이터베이스는 초기화 파라미터를 사용하여 메모리 구조를 생성하고 구성합니다. 예를 들어, SGA\_TARGET 파라미터는 SGA 구성 요소의 총 크기를 지정합니다. SGA\_TARGET을 0으로 설정하면 자동 공유 메모리 관리(Automatic Shared Memory Management)가 비활성화됩니다.

## 프로세스 구조

### DB 구조

- 메모리
- 프로세스
- 저장 영역

- User process:
  - 데이터베이스 유저 또는 일괄 처리 프로세스가 오라클 데이터베이스에 연결될 때 시작됩니다.
- 데이터베이스 프로세스:
  - 서버 프로세스: Oracle instance에 연결하며, 유저가 세션을 설정할 때 시작됩니다.
  - 백그라운드 프로세스: Oracle instance가 시작될 때 시작됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 프로세스 구조

오라클 데이터베이스 서버의 프로세스는 다음 두 개의 주요 그룹으로 분류할 수 있습니다.

- 응용 프로그램 또는 Oracle 도구 코드를 실행하는 user process
- 오라클 데이터베이스 서버 코드를 실행하는 오라클 데이터베이스 프로세스. 여기에는 서버 프로세스와 백그라운드 프로세스가 해당됩니다.

유저가 응용 프로그램이나 Oracle 도구(예: SQL\*Plus)를 실행하면 오라클 데이터베이스는 유저의 응용 프로그램을 실행하는 *user process*를 생성합니다. 오라클 데이터베이스는 *user process*에서 실행한 명령을 처리하는 *서버 프로세스*도 생성합니다. 또한 Oracle 서버는 *instance*에 대해 서로 상호 작용하거나 운영 체제와 상호 작용하는 일련의 *백그라운드 프로세스*를 포함하고 있습니다. 이러한 백그라운드 프로세스는 메모리 구조를 관리하고, 디스크에 데이터를 기록하기 위해 I/O를 비동기적으로 수행하고, 기타 필요한 작업을 수행하는 데 사용됩니다.

운영 체제와 선택한 오라클 데이터베이스 옵션에 따라 오라클 데이터베이스 구성마다 프로세스 구조가 다릅니다. 연결된 유저에 대한 코드는 *dedicated server* 또는 *shared server*로 구성할 수 있습니다.

- *Dedicated server*를 사용할 경우 각 유저에 대해 *user process*에서 데이터베이스 응용 프로그램을 실행하며 *user process*는 오라클 데이터베이스 서버 코드를 실행하는 *dedicated server 프로세스*에서 처리됩니다.
- *Shared server*를 사용하면 각 연결에 대해 *dedicated server 프로세스*를 실행할 필요가 없습니다. 디스패처가 여러 개의 수신 네트워크 세션 요청을 *shared server 프로세스* 풀로 전달합니다. *Shared server 프로세스*는 모든 클라이언트 요청을 처리합니다.

## 프로세스 구조(계속)

### 서버 프로세스

오라클 데이터베이스는 서버 프로세스를 생성하여 instance에 연결된 user process의 요청을 처리합니다. 응용 프로그램과 오라클 데이터베이스가 동일한 컴퓨터에서 작동하는 경우 user process와 해당 서버 프로세스를 단일 프로세스로 결합하여 시스템 오버헤드를 줄일 수 있습니다. 그러나 응용 프로그램과 오라클 데이터베이스가 다른 컴퓨터에서 작동할 경우 user process는 항상 별도의 서버 프로세스를 통해 오라클 데이터베이스와 통신합니다.

각 유저의 응용 프로그램 대신 생성된 서버 프로세스는 다음과 같은 작업을 수행할 수 있습니다.

- 응용 프로그램을 통해 실행된 SQL 문을 구문 분석하고 실행합니다.
- SGA에 데이터 블록이 없는 경우 디스크의 데이터 파일에서 SGA의 공유 데이터베이스 버퍼로 필요한 데이터 블록을 읽어 들입니다.
- 응용 프로그램이 정보를 처리할 수 있는 방식으로 결과를 반환합니다.

### 백그라운드 프로세스

성능을 최대화하고 여러 명의 유저를 수용하기 위해 다중 프로세스 오라클 데이터베이스 시스템에서는 백그라운드 프로세스라고 하는 몇 개의 추가 오라클 데이터베이스 프로세스를 사용합니다. 오라클 데이터베이스 instance는 여러 개의 백그라운드 프로세스를 사용할 수 있습니다.

데이터베이스 instance를 성공적으로 시작하는 데 필요한 백그라운드 프로세스는 다음과 같습니다.

- DBWn(데이터베이스 기록자)
- LGWR(로그 기록자)
- 체크포인트(CKPT)
- SMON(시스템 모니터)
- PMON(프로세스 모니터)

다음 백그라운드 프로세스는 필요할 때 시작할 수 있는 선택적 백그라운드 프로세스에 대한 몇 가지 예입니다.

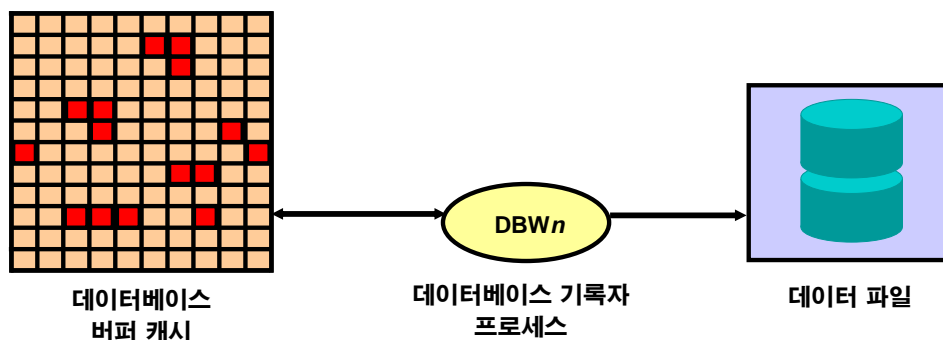
- 복구자(RECO)
- 작업 큐
- 아카이버(ARCn)
- 큐 모니터(QMNn)

다른 백그라운드 프로세스는 RAC(Real Application Cluster)와 같은 고급 구성에서 찾을 수 있습니다. 백그라운드 프로세스에 대한 자세한 내용은 V\$BGPROCESS 뷰를 참조하십시오. 대부분의 운영 체제에서 백그라운드 프로세스는 instance가 시작되면 자동으로 생성됩니다.

## 데이터베이스 기록자 프로세스

데이터베이스 버퍼 캐시의 수정된(더티) 버퍼를 디스크에 기록합니다.

- 다른 프로세싱을 수행하는 동안 비동기적으로
- 체크포인트를 전진시키기 위해 주기적으로



ORACLE

Copyright © 2009, Oracle. All rights reserved.

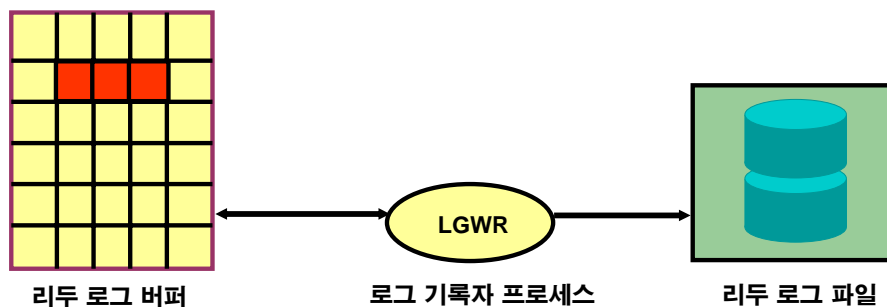
### 데이터베이스 기록자 프로세스

DBW<sub>n</sub>(데이터베이스 기록자) 프로세스는 버퍼의 내용을 데이터 파일에 기록하며, 데이터베이스 버퍼 캐시의 수정된(더티) 버퍼를 디스크에 기록합니다. 대부분의 시스템에서는 하나의 데이터베이스 기록자 프로세스(DBW0)만 있으면 충분하지만 시스템에서 많은 데이터를 수정하는 경우 추가 프로세스(DBW1 - DBW9)를 구성하여 쓰기 성능을 향상시킬 수 있습니다. 단일 프로세서 시스템에서는 이러한 추가 DBW<sub>n</sub> 프로세스가 유용하지 않습니다.

데이터베이스 버퍼 캐시의 버퍼가 수정된 경우 해당 버퍼는 "더티" 버퍼로 표시되고 시스템 변경 번호(SCN) 순서로 보관된 더티 버퍼의 LRUW 리스트에 추가됩니다. 따라서 이와 같이 변경된 버퍼에 해당하는, 리두 로그에 기록된 리두의 순서와 일치하게 됩니다. 버퍼 캐시에서 사용 가능한 버퍼의 수가 내부 임계값보다 적어서 서버 프로세스가 사용 가능한 버퍼를 확보하기 힘든 경우 DBW<sub>n</sub>은 LRUW 리스트의 순서에 따라 수정된 순서대로 더티 버퍼를 데이터 파일에 기록합니다.

## 로그 기록자 프로세스

- 리두 로그 버퍼를 디스크의 리두 로그 파일에 기록합니다.
- LGWR은 다음 경우 기록합니다.
  - 프로세스가 트랜잭션을 커밋할 때
  - 리두 로그 버퍼의 1/3이 찼을 때
  - DBWn 프로세스가 수정된 버퍼를 디스크에 기록하기 전에



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 로그 기록자 프로세스

LGWR(로그 기록자) 프로세스는 리두 로그 버퍼 항목을 디스크의 리두 로그 파일에 기록하여 리두 로그 버퍼를 관리하며, 마지막으로 기록된 이후 버퍼로 복사된 모든 리두 항목을 기록합니다.

리두 로그 버퍼는 일종의 순환 버퍼이므로 LGWR이 리두 로그 버퍼의 리두 항목을 리두 로그 파일에 기록하면 서버 프로세스가 새 항목을 복사하여 디스크에 기록된 리두 로그 버퍼의 항목을 덮어 쓸 수 있습니다. LGWR은 일반적으로 빠르게 기록하여 리두 로그에 대한 액세스가 많을 때도 항상 새 항목을 위한 버퍼 공간을 확보해 둡니다.

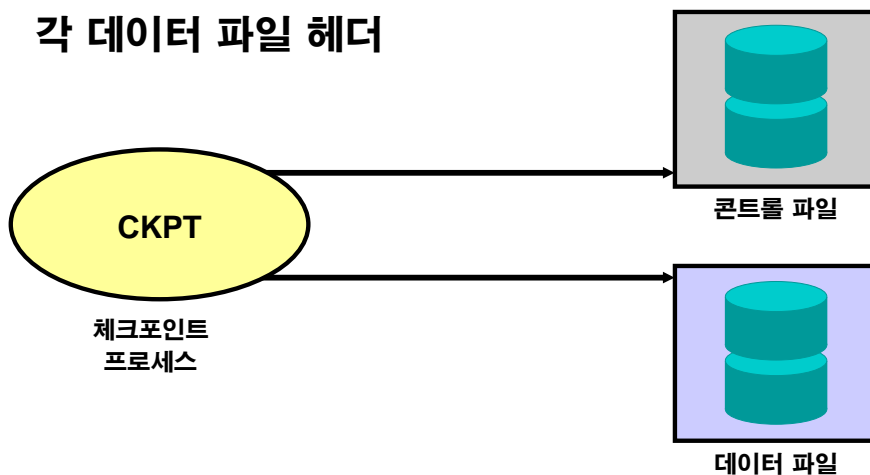
LGWR은 인접한 한 개의 버퍼 부분을 디스크에 기록합니다. LGWR은 다음 경우 기록합니다.

- User process가 트랜잭션을 커밋할 때
- 리두 로그 버퍼의 1/3이 찼을 때
- DBWn 프로세스가 수정된 버퍼를 디스크에 기록하기 전에(필요한 경우)

# 체크포인트 프로세스

체크포인트 정보를 읽는 위치:

- 컨트롤 파일
- 각 데이터 파일 헤더



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 체크포인트 프로세스

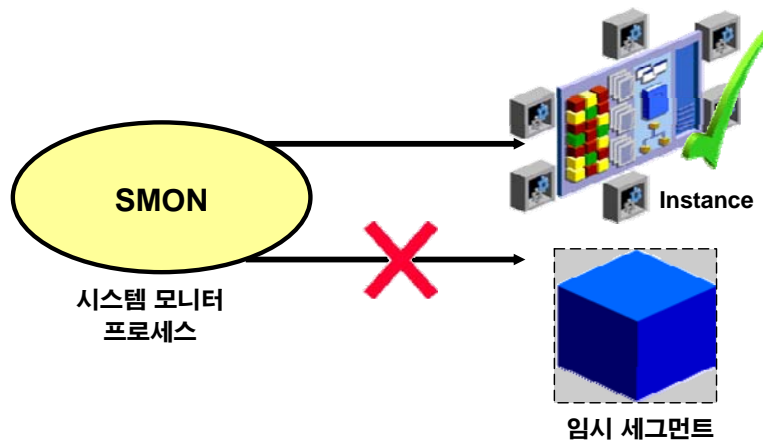
체크포인트는 데이터베이스의 리두 스레드에 SCN을 정의하는 데이터 구조로, 컨트롤 파일과 각 데이터 파일 헤더에 기록되며 매우 중요한 recovery 요소입니다.

체크포인트가 발생할 경우 오라클 데이터베이스는 모든 데이터 파일의 헤더를 갱신하여 체크포인트의 세부 사항을 기록해야 하는데, 이 작업은 CKPT 프로세스에 의해 수행됩니다. CKPT 프로세스는 블록을 디스크에 기록하지 않지만 DBWn은 항상 블록을 디스크에 기록합니다. 파일에 기록된 SCN은 해당 SCN이 디스크에 기록되기 전에 데이터베이스 블록에 대한 모든 변경이 이루어졌음을 나타냅니다.

Oracle Enterprise Manager에서 SYSTEM\_STATISTICS 모니터에 의해 표시되는 정적 DBWR 체크포인트는 완료된 체크포인트 요청 수를 나타냅니다.

## 시스템 모니터 프로세스

- Instance 시작 시 recovery 수행
- 사용하지 않는 임시 세그먼트 정리



ORACLE

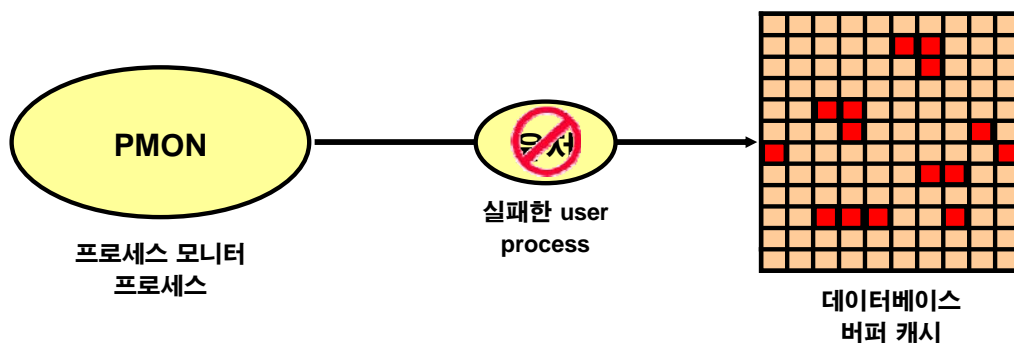
Copyright © 2009, Oracle. All rights reserved.

### 시스템 모니터 프로세스

SMON(시스템 모니터) 프로세스는 instance가 시작될 때 필요에 따라 recovery를 수행하며, 더 이상 사용하지 않는 임시 세그먼트도 정리합니다. 파일 읽기 또는 오프라인 오류로 인해 instance recovery 중 종료된 트랜잭션을 건너뛰는 경우 테이블스페이스나 파일이 다시 온라인으로 전환되면 SMON이 이를 recovery합니다. SMON은 정기적으로 필요 여부를 확인합니다. SMON이 필요할 경우 다른 프로세스에서 SMON을 호출할 수 있습니다.

## 프로세스 모니터 프로세스

- User process가 실패할 경우 프로세스 recovery 수행
  - 데이터베이스 버퍼 캐시 정리
  - User process에서 사용하는 리소스 해제
- Idle 세션 타임아웃에 대한 세션 모니터
- 리스너에 동적으로 데이터베이스 서비스 등록



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 프로세스 모니터 프로세스

PMON(프로세스 모니터)은 user process가 실패할 경우 프로세스 recovery를 수행하고, 데이터베이스 버퍼 캐시를 정리하며, user process에서 사용하고 있는 리소스를 해제합니다. 예를 들어, PMON은 활성 트랜잭션 테이블의 상태를 재설정하고, 잠금을 해제하며, 활성 프로세스 리스트에서 프로세스 ID를 제거합니다.

PMON은 정기적으로 디스패처와 서버 프로세스의 상태를 확인하여 실행이 정지된 경우(오라클 데이터베이스에서 의도적으로 종료한 경우 제외) 해당 디스패처나 서버 프로세스를 재시작합니다. 또한 instance 및 디스패처 프로세스에 대한 정보도 네트워크 리스너에 등록합니다.

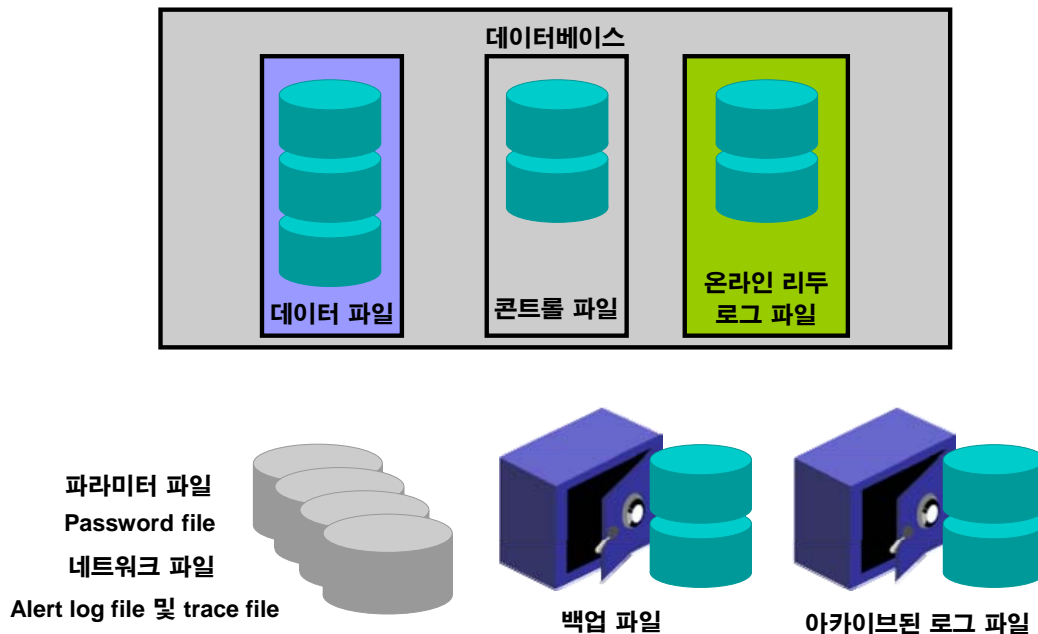
SMON과 마찬가지로 PMON은 정기적으로 필요 여부를 확인하여 다른 프로세스에서 PMON을 필요로 할 경우 호출할 수 있습니다.



# 오라클 데이터베이스 저장 영역 구조

## DB 구조

- 메모리
- 프로세스
- 저장 영역



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 오라클 데이터베이스 저장 영역 구조

오라클 데이터베이스는 다음과 같은 파일로 구성되어 있습니다.

- **컨트롤 파일:** 데이터베이스 자체에 대한 데이터(즉, 물리적 데이터베이스 구조 정보)를 포함합니다. 이 파일은 데이터베이스에 중요합니다. 이 파일이 없으면 데이터베이스 내의 데이터에 액세스할 때 데이터 파일을 열 수 없습니다.
- **데이터 파일:** 데이터베이스의 유저 또는 응용 프로그램 데이터, 메타 데이터 및 데이터 디렉토리를 포함합니다.
- **온라인 리두 로그 파일:** 데이터베이스의 instance recovery를 허용합니다. 데이터베이스 서버가 손상되었지만 해당 데이터 파일은 손실되지 않은 경우 instance는 이러한 파일 안에 있는 정보를 사용하여 데이터베이스를 recovery할 수 있습니다.

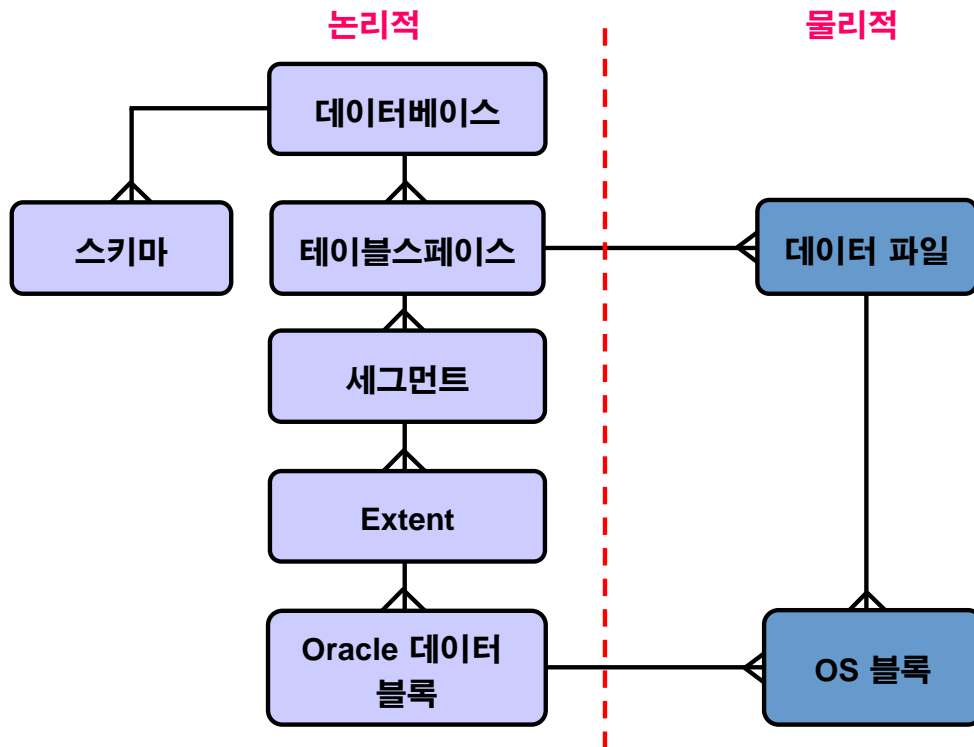
다음은 성공적인 데이터베이스 실행에 필요한 추가 파일입니다.

- **백업 파일:** 데이터베이스 recovery에 사용됩니다. 백업 파일은 일반적으로 Media Failure 또는 User Error로 원본 파일이 손상되었거나 삭제되었을 경우에 복원됩니다.
- **아카이브된 로그 파일:** instance에 의해 생성되는 데이터 변경(리두)에 대한 기록을 지속적으로 포함합니다. 이 파일과 데이터베이스 백업을 사용하면 손실된 데이터 파일을 recovery할 수 있습니다. 즉, 아카이브 로그는 복원된 데이터 파일의 recovery를 활성화합니다.
- **파라미터 파일:** instance 시작 시 어떻게 instance를 구성할지 정의하는 데 사용됩니다.
- **Password file:** sysdba/sysoper/sysasm이 데이터베이스에 원격으로 연결하여 관리 작업을 수행할 수 있도록 합니다.

## 오라클 데이터베이스 저장 영역 구조(계속)

- **네트워크 파일:** 데이터베이스 리스너를 시작하는 데 사용되며 유저 연결에 필요한 정보를 저장합니다.
- **Trace file:** 각 서버와 백그라운드 프로세스는 연관된 trace file에 정보를 기록할 수 있습니다. 내부 오류가 프로세스에서 감지되면 프로세스는 오류에 대한 정보를 해당 trace file에 덤프합니다. Trace file에 기록된 정보 중 일부는 데이터베이스 관리자가 사용하고 일부는 오라클 고객 지원 센터에서 사용합니다.
- **Alert log file:** 특수 trace 항목으로, 데이터베이스의 alert log에는 메시지와 오류가 시간순으로 기록되어 있습니다. Instance마다 한 개의 alert log file이 있습니다. 오라클은 이 alert log를 정기적으로 검토할 것을 권장합니다.

# 논리적 및 물리적 데이터베이스 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 논리적 및 물리적 데이터베이스 구조

오라클 데이터베이스는 논리적 저장 영역 구조와 물리적 저장 영역 구조를 갖습니다.

### 테이블스페이스

데이터베이스는 테이블스페이스라는 논리적 저장 영역 단위로 나뉘어 관련된 논리적 구조를 함께 그룹화합니다. 예를 들어, 테이블스페이스는 일반적으로 응용 프로그램의 모든 객체를 간단히 몇 개의 관리 작업으로 그룹화합니다. 응용 프로그램 데이터에 대한 테이블스페이스와 응용 프로그램 인덱스에 대한 추가 테이블스페이스를 가질 수 있습니다.

### 데이터베이스, 테이블스페이스 및 데이터 파일

위의 슬라이드에는 데이터베이스, 테이블스페이스 및 데이터 파일 간의 관계가 나와 있습니다. 각 데이터베이스는 논리적으로 하나 이상의 테이블스페이스로 나뉩니다. 각 테이블스페이스에는 테이블스페이스에 있는 모든 논리적 구조 데이터를 물리적으로 저장할 수 있는 하나 이상의 데이터 파일이 명시적으로 생성됩니다. TEMPORARY 테이블스페이스의 경우 데이터 파일 대신 테이블스페이스에 임시 파일이 있습니다.

## 논리적 및 물리적 데이터베이스 구조(계속)

### 스키마

스키마는 데이터베이스 사용자가 소유하는 데이터베이스 객체의 모음입니다. 스키마 객체는 데이터베이스의 데이터를 직접 참조하는 논리적 구조입니다. 스키마 객체에는 테이블, 뷰, 시퀀스, 내장 프로시저, 동의어, 인덱스, 클러스터 및 데이터베이스 링크가 있습니다. 일반적으로 스키마 객체에는 응용 프로그램이 데이터베이스에 생성하는 모든 것이 포함됩니다.

### 데이터 블록

가장 작은 세분성 레벨에서 오라클 데이터베이스의 데이터는 데이터 블록에 저장됩니다. 하나의 데이터 블록은 디스크에서 특정 바이트 수의 물리적 데이터베이스 공간에 해당합니다. 데이터 블록 크기는 생성 시 각 테이블스페이스에 대해 지정됩니다. 데이터베이스는 사용 가능한 데이터베이스 공간을 Oracle 데이터 블록으로 사용하고 할당합니다.

### Extent

그 다음 레벨의 논리적 데이터베이스 공간은 extent입니다. Extent는 단일 할당으로 얻은 일정 수의 연속적인 데이터 블록으로, 특정 유형의 정보를 저장하는 데 사용됩니다.

### 세그먼트

Extent 다음 레벨의 논리적 데이터베이스 저장 영역은 세그먼트입니다. 세그먼트는 특정 논리적 구조에 할당된 일련의 extent입니다. 예를 들어, 다음과 같은 다양한 유형의 세그먼트가 있습니다.

- **데이터 세그먼트:** 클러스터화되지 않은 각각의 비인덱스 구성(non-index-organized) 테이블에는 데이터 세그먼트가 있습니다. 단, external table, 전역 임시 테이블(global temporary table) 및 partition 테이블은 예외입니다. 이 경우 테이블마다 한 개 이상의 세그먼트가 있습니다. 테이블 데이터는 모두 해당 데이터 세그먼트의 extent에 저장됩니다. Partition 테이블의 경우 각 partition은 데이터 세그먼트를 가집니다. 각 클러스터는 데이터 세그먼트를 가집니다. 클러스터에 있는 모든 테이블의 데이터는 클러스터의 데이터 세그먼트에 저장됩니다.
- **인덱스 세그먼트:** 각 인덱스는 해당 데이터를 모두 저장하는 인덱스 세그먼트를 가집니다. Partition 인덱스의 경우 각 partition은 인덱스 세그먼트를 가집니다.
- **언두 세그먼트:** 일시적으로 언두 정보를 저장하기 위해 수많은 언두 세그먼트를 포함하는 UNDO 테이블스페이스가 데이터베이스 instance당 한 개씩 생성됩니다. 언두 세그먼트의 정보는 데이터베이스 recovery 중 유저에게 커밋되지 않은 트랜잭션을 롤백하기 위해 읽기 일관성 데이터베이스 정보를 생성하는 데 사용됩니다.
- **임시 세그먼트:** 임시 세그먼트는 SQL 문에서 실행을 완료할 임시 작업 영역이 필요할 때 오라클 데이터베이스에 의해 생성됩니다. 명령문의 실행이 완료되면 나중에 사용할 수 있도록 임시 세그먼트의 extent가 instance로 반환됩니다. 모든 유저에 대해 기본 임시 테이블스페이스를 지정하거나 데이터베이스 차원에서 사용할 기본 임시 테이블스페이스를 지정하십시오.

오라클 데이터베이스는 동적으로 공간을 할당합니다. 세그먼트의 기존 extent가 가득 차면 다른 extent가 추가됩니다. Extent는 필요에 따라 할당되므로 세그먼트의 extent는 디스크에서 인접해 있을 수도 있고 그렇지 않을 수도 있습니다.

# SQL 문 처리

- 다음을 사용하여 instance에 연결합니다.
  - User process
  - 서버 프로세스
- 사용되는 Oracle 서버 구성 요소는 SQL 문 유형에 따라 다릅니다.
  - Query는 행을 반환합니다.
  - DML(데이터 조작어) 문은 변경 사항을 기록합니다.
  - 커밋은 트랜잭션 recovery를 보장합니다.
- 일부 Oracle 서버 구성 요소는 SQL 문 처리에 관여하지 않습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## SQL 문 처리

Oracle instance의 모든 구성 요소가 SQL 문 처리에 사용되는 것은 아닙니다. User processes와 서버 프로세스는 유저를 Oracle instance에 연결하는 데 사용됩니다. 이러한 프로세스는 Oracle instance에 속하지는 않지만 SQL 문을 처리하는 데 반드시 필요합니다.

일부 백그라운드 프로세스, SGA 구조 및 데이터베이스 파일도 SQL 문을 처리하는 데 사용됩니다. SQL 문의 유형에 따라 다른 구성 요소가 사용됩니다.

- Query에는 유저에게 행을 반환하기 위한 추가 처리가 필요합니다.
- DML 문에는 데이터의 변경 사항을 기록하기 위한 추가 처리가 합니다.
- 커밋 프로세스는 트랜잭션의 수정된 데이터가 recovery될 수 있도록 보장합니다.

일부 필수 백그라운드 프로세스는 SQL 문 처리에 직접 관여하지는 않지만 성능 개선 및 데이터베이스 recovery에 사용됩니다. 예를 들어, 선택적 아카이버 백그라운드 프로세스인 ARCn을 사용하면 운용 중인 데이터베이스를 recovery할 수 있습니다.

# Query 처리

- **구문 분석:**
  - 동일한 명령문을 검색합니다.
  - 구문, 객체 이름 및 권한을 검사합니다.
  - 구문 분석 중 사용된 객체를 잠급니다.
  - 실행 계획을 생성하고 저장합니다.
- **실행:** 선택된 행을 식별합니다.
- **패치(fetch):** User process로 행을 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Query 처리

Query는 성공할 경우 데이터를 결과로 반환한다는 점에서 다른 SQL 문 유형과 다릅니다. 다른 명령문이 단순히 성공 또는 failure를 반환하는 반면 query는 한 행이나 수천 개의 행을 반환할 수 있습니다.

Query 처리의 주요 세 단계는 다음과 같습니다.

- 구문 분석
- 실행
- 패치(fetch)

구문 분석 단계에서는 SQL 문이 user process에서 서버 프로세스로 전달되고 구문 분석된 SQL 문이 공유 SQL 영역으로 로드됩니다.

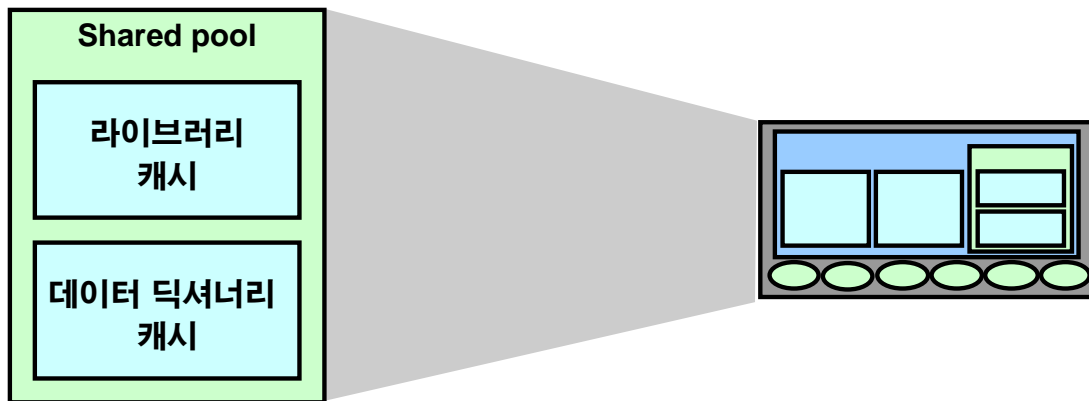
구문 분석 중 서버 프로세스는 다음 기능을 수행합니다.

- Shared pool에서 기존의 SQL 문 복사본을 검색합니다.
- 구문을 확인하여 SQL 문을 검증합니다.
- 데이터 디렉터리 조회를 실행하여 테이블 및 열 정의를 검증합니다.

실행 단계는 최적의 옵티마이저 접근 방식을 사용하여 명령문을 실행하고 패치(fetch)는 유저에게 다시 행을 가져옵니다.

# Shared Pool

- 라이브러리 캐시는 SQL 문 텍스트, 구문 분석된 코드 및 실행 계획을 포함합니다.
- 데이터 디렉터리 캐시는 테이블, 열 및 기타 객체 정의 및 권한을 포함합니다.
- Shared pool은 SHARED\_POOL\_SIZE에 의해 크기가 조정됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Shared Pool

구문 분석 단계에서 서버 프로세스는 shared pool이라는 SGA의 영역을 사용하여 SQL 문을 컴파일합니다. Shared pool은 다음 두 가지 기본 구성 요소를 가집니다.

- 라이브러리 캐시
- 데이터 디렉터리 캐시

### 라이브러리 캐시

라이브러리 캐시는 공유 SQL 영역이라는 메모리 구조에 가장 최근에 사용된 SQL 문에 대한 정보를 저장합니다. 공유 SQL 영역은 다음을 포함합니다.

- SQL 문 텍스트
- 구문 분석 트리: 명령문의 컴파일된 버전
- 실행 계획: 명령문을 실행할 때 수행할 단계

옵티마이저는 최적의 실행 계획을 결정하는 Oracle 서버의 기능입니다.

SQL 문이 재실행되고 공유 SQL 영역이 이미 SQL 문에 대한 실행 계획을 포함하고 있을 경우 서버 프로세스는 SQL 문의 구문을 분석할 필요가 없습니다. 라이브러리 캐시는 구문 분석 시간과 메모리 요구 사항을 줄임으로써 SQL 문을 재사용하는 응용 프로그램의 성능을 향상시킵니다. SQL 문이 재사용되지 않을 경우 SQL 문은 결국 라이브러리 캐시에서 삭제됩니다.

## Shared Pool(계속)

### 데이터 디렉터리 캐시

데이터 디렉터리 캐시(디렉터리 캐시 또는 행 캐시라고도 함)는 데이터베이스에서 가장 최근에 사용된 정의의 모음입니다. 데이터베이스 파일, 테이블, 인덱스, 열, 유저, 권한 및 기타 데이터베이스 객체에 대한 정보를 포함합니다.

구문 분석 단계에서 서버 프로세스는 디렉터리 캐시에서 정보를 찾아 SQL 문에 지정된 객체 이름을 분석하고 액세스 권한을 검증합니다. 필요한 경우 서버 프로세스는 데이터 파일에서 이 정보의 로드를 시작합니다.

### Shared Pool 크기 조정

Shared pool의 크기는 초기화 파라미터 SHARED\_POOL\_SIZE에 의해 지정됩니다.



# 데이터베이스 버퍼 캐시

- 데이터베이스 버퍼 캐시는 가장 최근에 사용된 블록을 저장합니다.
- 버퍼 크기는 DB\_BLOCK\_SIZE를 기반으로 합니다.
- 버퍼 수는 DB\_BLOCK\_BUFFERS에 의해 정의됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 데이터베이스 버퍼 캐시

Query를 처리할 때 서버 프로세스는 데이터베이스 버퍼 캐시에서 필요한 블록을 찾습니다. 데이터베이스 버퍼 캐시에서 블록이 발견되지 않을 경우 서버 프로세스는 데이터 파일에서 블록을 읽고 버퍼 캐시에 복사본을 배치합니다. 동일한 블록에 대해 발생하는 이후의 요청은 이 블록을 메모리에서 찾을 수 있기 때문에 물리적 읽기를 수행할 필요가 없습니다. Oracle 서버는 LRU(Least Recently Used) 알고리즘을 사용하여 최근에 액세스하지 않은 버퍼를 삭제함으로써 버퍼 캐시에 새 블록을 위한 공간을 만듭니다.

### 데이터베이스 버퍼 캐시 크기 조정

버퍼 캐시 내 각 버퍼의 크기는 Oracle 블록의 크기와 같고 DB\_BLOCK\_SIZE 파라미터에 의해 지정됩니다. 버퍼 수는 DB\_BLOCK\_BUFFERS 파라미터의 값과 같습니다.

## 프로그램 글로벌 영역(PGA)

- 공유되지 않습니다.
- 서버 프로세스에 의해서만 쓰기가 가능합니다.
- 다음을 포함합니다.
  - 정렬 영역
  - 세션 정보
  - Cursor State
  - Stack Space



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 프로그램 글로벌 영역(PGA)

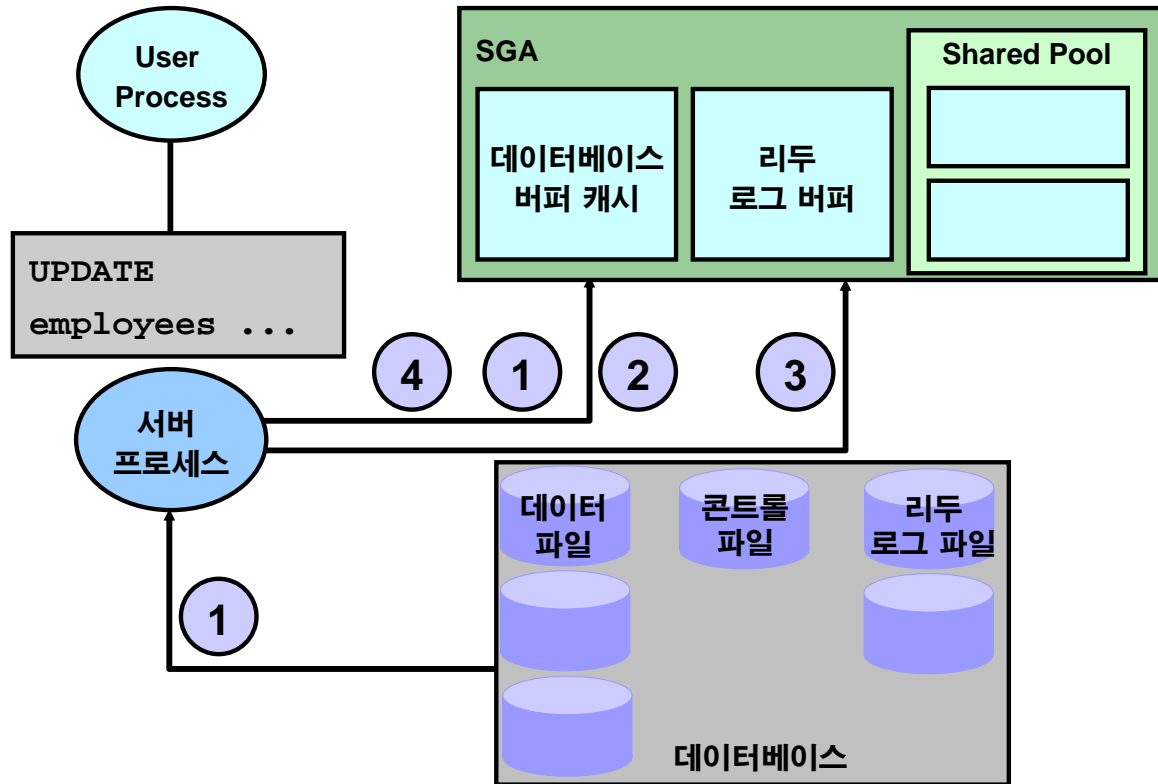
프로그램 글로벌 영역(PGA)은 서버 프로세스에 대한 데이터 및 제어 정보를 포함하는 메모리 영역입니다. 이것은 서버 프로세스가 시작될 때 오라클에서 생성하는 비공유 메모리입니다. PGA에는 해당 서버 프로세스만 액세스할 수 있으며, PGA를 대신하여 작동하는 Oracle 서버 코드로만 읽고 쓸 수 있습니다. Oracle instance에 연결된 각 서버 프로세스에 의해 할당된 PGA 메모리는 instance에 의해 할당된 집계 PGA 메모리라고도 합니다.

Dedicated server 구성에서 서버의 PGA는 다음 구성 요소를 포함합니다.

- **정렬 영역:** SQL 문을 처리하는 데 필요한 정렬에 사용됩니다.
- **세션 정보:** 세션에 대한 유저 권한 및 성능 통계를 포함합니다.
- **Cursor state:** 현재 세션에 의해 사용되는 SQL 문을 처리하는 단계를 나타냅니다.
- **Stack space:** 다른 세션 변수를 포함합니다.

PGA는 프로세스가 생성될 때 할당되고 프로세스가 종료될 때 할당이 해제됩니다.

## DML 문 처리



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DML 문 처리

DML(데이터 조작어) 문을 처리하는 데는 구문 분석과 실행이라는 두 단계만 필요합니다.

- 구문 분석은 query를 처리하는 데 사용되는 구문 분석 단계와 동일합니다.
- 실행에는 데이터 변경을 위한 추가 처리가 필요합니다.

### DML 실행 단계

DML 문을 실행하려면 다음을 수행합니다.

- 데이터 및 롤백 블록이 버퍼 캐시에 없는 경우 서버 프로세스가 이러한 블록을 데이터 파일에서 읽어 버퍼 캐시로 복사합니다.
- 서버 프로세스는 수정할 행을 잠급니다.
- 리두 로그 버퍼에서 서버 프로세스가 롤백 및 데이터 블록에 대한 변경 사항을 기록합니다.
- 롤백 블록 변경은 데이터가 수정되기 전에 데이터 값을 기록합니다. 롤백 블록은 필요한 경우 DML 문이 롤백될 수 있도록 데이터의 "이전 이미지"를 저장하는 데 사용됩니다.
- 데이터 블록 변이 새로운 데이터 값을 기록합니다.

## SQL 문 처리(계속)

서버 프로세스는 "이전 이미지"를 롤백 블록에 기록하고 데이터 블록을 갱신합니다. 이 두 가지 변경은 모두 데이터베이스 버퍼 캐시에서 이루어집니다. 버퍼 캐시에서 변경된 블록은 더티 버퍼, 즉 디스크에 있는 해당 블록과 동일하지 않은 버퍼로 표시됩니다.

DELETE 또는 INSERT 명령의 처리는 유사한 단계를 사용합니다. DELETE에 대한 "이전 이미지"는 삭제된 행에 있는 열 값을 포함하며, INSERT의 이전 이미지는 행 위치 정보를 포함합니다.

블록에서 변경된 사항은 메모리 구조에만 기록되고 즉시 디스크에 쓰여지지 않기 때문에, SGA의 손실을 가져오는 컴퓨터 failure가 발생할 경우에는 이러한 변경 사항도 손실될 수 있습니다.

## 리두 로그 버퍼

- LOG\_BUFFER에 의해 정의된 크기를 가집니다.
- Instance를 통해 변경된 사항을 기록합니다.
- 연속적으로 사용됩니다.
- 순환 버퍼입니다.



ORACLE

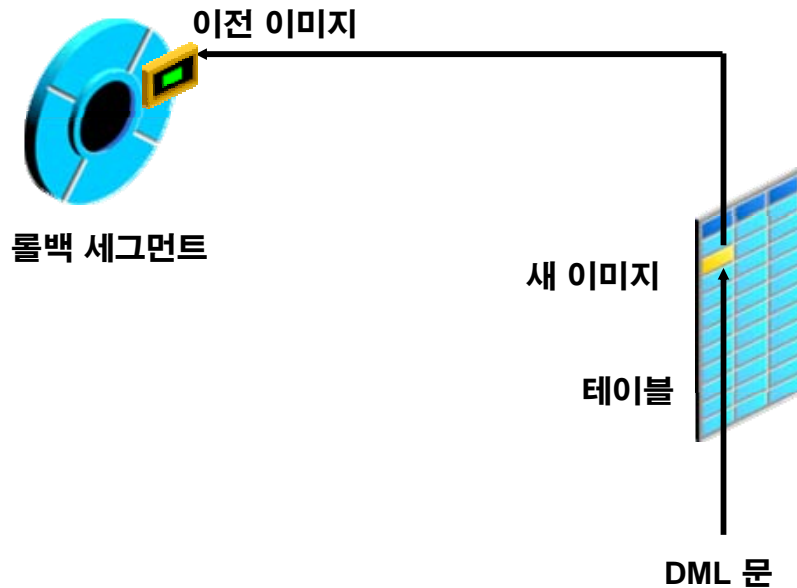
Copyright © 2009, Oracle. All rights reserved.

### 리두 로그 버퍼

서버 프로세스는 SGA의 일부인 리두 로그 버퍼에 데이터 파일 블록에서 변경된 사항을 대부분 기록합니다. 리두 로그 버퍼는 다음과 같은 특성을 가집니다.

- 바이트 단위의 크기는 LOG\_BUFFER 파라미터에 의해 정의됩니다.
- 리두 항목에 변경된 블록, 변경 위치 및 새로운 값을 기록합니다. 리두 항목은 변경되는 블록에서 유형에 차이가 없으며, 단지 블록에서 변경된 바이트를 기록합니다.
- 리두 로그 버퍼는 연속적으로 사용되어 한 트랜잭션의 변경 사항을 다른 트랜잭션의 변경 사항에 끼울 수 있습니다.
- 버퍼가 다 차면 재사용되는 순환 버퍼입니다. 단, 모든 이전 리두 항목이 리두 로그 파일에 기록된 후에야 재사용할 수 있습니다.

## 롤백 세그먼트



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 롤백 세그먼트

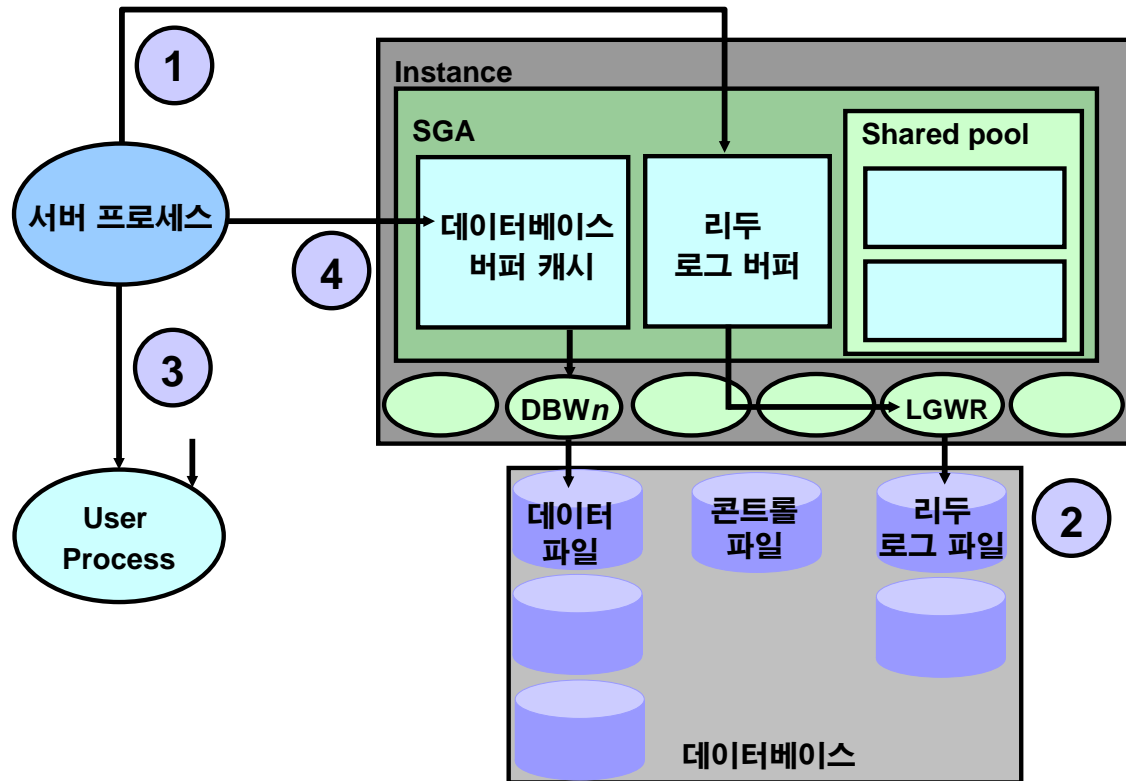
내용을 변경하기 전 서버 프로세스는 이전 데이터 값을 롤백 세그먼트에 저장합니다. 이 "이전 이미지"는 다음 작업에 사용됩니다.

- 트랜잭션이 롤백될 경우 변경 언두
- DML 문으로 수행한 커밋되지 않은 변경 사항을 다른 트랜잭션이 보지 못하도록 하여 읽기 일관성 제공
- Failure 발생 시 데이터베이스를 일관된 상태로 recovery

테이블 및 인덱스와 같은 롤백 세그먼트는 데이터 파일에 존재하며 롤백 블록은 필요에 따라 데이터베이스 버퍼 캐시로 유입됩니다. 롤백 세그먼트는 DBA에 의해 생성됩니다.

롤백 세그먼트에 대한 변경 사항은 리두 로그 버퍼에 기록됩니다.

## COMMIT 처리



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### COMMIT 처리

Oracle 서버는 instance failure 발생 시 커밋된 변경 사항이 recovery될 수 있도록 하는 빠른 COMMIT 방식을 사용합니다.

#### 시스템 변경 번호

트랜잭션이 커밋할 때마다 Oracle 서버는 트랜잭션에 커밋 SCN을 할당합니다. SCN은 일정하게 증가하며 데이터베이스 내에서 고유합니다. Oracle 서버는 SCN을 내부 시간 기록으로 사용하여 데이터를 동기화하고 데이터 파일에서 데이터를 검색할 때 읽기 일관성을 제공합니다. SCN을 사용하면 Oracle 서버는 운영 체제의 날짜와 시간에 관계없이 일관성 검사를 수행할 수 있습니다.

#### COMMIT 처리 단계

COMMIT이 실행될 때 다음 단계가 수행됩니다.

1. 서버 프로세스는 SCN과 함께 커밋 기록을 리두 로그 버퍼에 배치합니다.
2. LGWR은 커밋 레코드를 포함하여 모든 리두 로그 버퍼 항목을 리두 로그 파일에 연속적으로 기록합니다. 이 시점부터 Oracle 서버는 instance failure가 발생할 경우에도 변경 사항이 손실되지 않도록 보장할 수 있습니다.

## 커밋 처리(계속)

3. 유저는 커밋이 완료되었다는 통지를 받습니다.
4. 서버 프로세스는 트랜잭션이 완료되었고 자원 잠금이 해제될 수 있음을 나타내는 정보를 기록합니다.

더티 버퍼를 데이터 파일에 비우는 작업은 DBW0에 의해 독립적으로 수행되며 커밋 이전 또는 이후에 이루어질 수 있습니다.

### 빠른 커밋의 이점

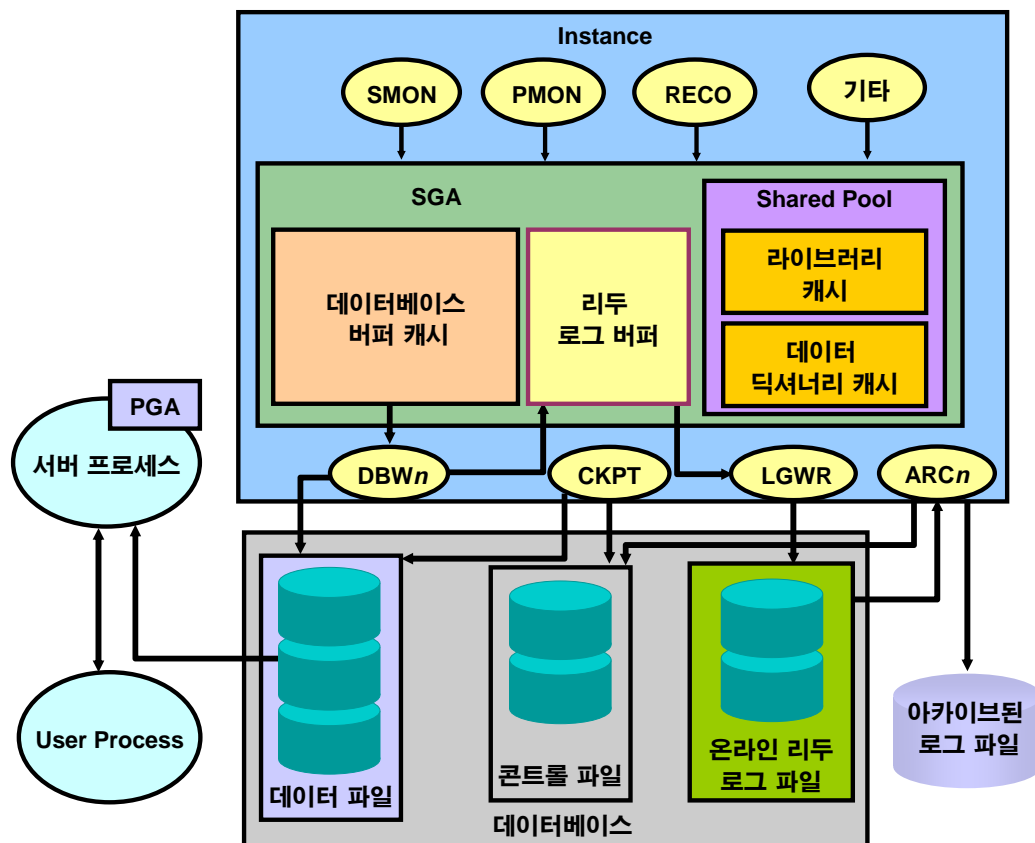
빠른 커밋 방식은 데이터 파일 대신 리두 로그 버퍼에 변경 사항을 기록함으로써 데이터 recovery를 보장합니다. 이 방식의 이점은 다음과 같습니다.

- 로그 파일에 순차적으로 쓰는 것이 데이터 파일의 다른 블록에 쓰는 것보다 속도가 빠릅니다.
- 로그 파일에는 변경 사항을 기록하는 데 필요한 최소 정보만 기록되는 반면, 데이터 파일에는 전체 데이터 블록이 기록되어야 합니다.
- 여러 트랜잭션이 동시에 커밋을 요청할 경우 instance는 리두 로그 레코드를 한 번의 쓰기로 피기백합니다.
- 리두 로그 버퍼가 특별히 가득 차지 않는 한, 트랜잭션당 한 번만 동기적으로 쓰면 됩니다. 피기백이 이루어질 경우 트랜잭션당 동기 쓰기 횟수는 한 번 미만일 수 있습니다.
- 리두 로그 버퍼는 커밋 이전에 비워질 수 있기 때문에 트랜잭션의 크기는 실제 커밋 연산에 필요한 시간에 영향을 미치지 않습니다.

**참고:** 트랜잭션을 롤백해도 LGWR은 디스크 쓰기를 시작하지 않습니다. Oracle 서버는 항상 failure recovery 시 커밋되지 않은 변경 사항을 롤백합니다. 롤백 후 롤백 항목이 디스크에 기록되기 전에 failure가 발생할 경우 커밋 레코드가 없으면 트랜잭션에 의해 변경된 사항은 롤백됩니다.



## 오라클 데이터베이스 구조 요약



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 오라클 데이터베이스 구조 요약

오라클 데이터베이스는 instance와 관련 데이터베이스로 구성됩니다.

- Instance는 SGA와 백그라운드 프로세스로 구성됩니다.
  - **SGA:** 데이터베이스 버퍼 캐시, 리두 로그 버퍼, shared pool 등
  - **백그라운드 프로세스:** SMON, PMON, DBWn, CKPT, LGWR 등
- 데이터베이스는 저장 영역 구조로 구성됩니다.
  - **논리적:** 테이블스페이스, 스키마, 세그먼트, extent 및 Oracle 블록
  - **물리적:** 데이터 파일, 컨트롤 파일, 리두 로그 파일

유저가 응용 프로그램을 통해 오라클 데이터베이스에 액세스하면 user process 대신 서버 프로세스가 instance와 통신합니다.