7

정규식 지원

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **정규식 사용 시 이점 나열**
- **정규식을 사용하여 문자열 검색, 일치 및 대체**

ORACLE

목표

이 단원에서는 정규식 지원 기능의 사용법에 대해 알아봅니다. 정규식 지원은 SQL 및 PL/SQL에서 모두 사용할 수 있습니다.

단원 내용

- **정규식 소개**
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP_LIKE
  - REGEXP_REPLACE
  - REGEXP_INSTR
  - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용
- 정규식 및 Check 제약 조건

ORACLE

정규식이란?

- 정규식에서 표준 구문 규칙을 사용하여 문자열 데이터의 간단한 패턴 및 복잡한 패턴을 검색하고 조작할 수 있습니다.
- SQL 함수 및 조건 집합을 사용하여 SQL 및 PL/SQL에서 문자열을 검색하고 조작할 수 있습니다.
- 다음을 사용하여 정규식을 지정합니다.
  - 메타 문자: 검색 알고리즘을 지정하는 연산자
  - 리터럴: 검색 중인 문자

ORACLE

정규식이란?

오라클 데이터베이스는 정규식을 지원합니다. 정규식의 구현은 ASCII 데이터와 일치하는 의미 및 구문을 위해 IEEE(Institute of Electrical and Electronics Engineers)가 제정한 POSIX(Portable Operating System for UNIX) 표준을 따릅니다. 오라클의 다중 언어 기능은 POSIX 표준을 넘어 연산자의 일치 기능을 확장합니다. 정규식은 검색 및 조작을 위해 간단한 패턴 및 복잡한 패턴을 설명하는 방식입니다.

문자열 조작 및 검색은 웹 기반 응용 프로그램 논리의 대부분을 차지합니다. 사용 범위는 지정된 텍스트에서 "San Francisco"란 단어를 찾는 간단한 검색부터, 텍스트에서 모든 URL을 추출하는 복잡한 검색과 두번째 문자가 모음인 모든 단어를 찾는 매우 복잡한 검색에 이르기까지 다양합니다.

고유의 SQL과 정규식을 함께 사용하면 오라클 데이터베이스에 저장된 임의의 데이터에서 매우 강력한 검색 및 조작 연산을 수행할 수 있습니다. 이 기능을 사용하면 복잡한 프로그래밍을 간단하게 해결할 수도 있습니다.

정규식 사용 시 이점

정규식을 사용하여 데이터베이스에서 복잡한 일치 논리를 구현하면 다음과 같이 이점이 있습니다.

- **오라클 데이터베이스에서 일치 논리를 중앙화함으로써 middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다.**
- **서버측 정규식을 사용하여 제약 조건을 적용함으로써 클라이언트에서 데이터 검증 논리 코드를 작성할 필요가 없습니다.**
- **내장 SQL 및 PL/SQL 정규식 함수와 조건을 사용하여 Oracle Database 11*g*의 이전 버전보다 더욱 쉽고 강력하게 문자열을 조작할 수 있습니다.**

ORACLE

정규식 사용 시 이점

정규식은 PERL 및 Java와 같은 프로그래밍 언어의 강력한 텍스트 처리 구성 요소입니다. 예를 들어, PERL 스크립트는 디렉토리의 각 HTML 파일을 처리하고 해당 내용을 단일 문자열로 스칼라 변수로 읽어들인 다음 정규식을 사용하여 문자열에서 URL을 검색합니다. 많은 개발자가 PERL을 사용하는 한 가지 이유는 강력한 패턴 일치 기능이 있기 때문입니다. 개발자는 Oracle의 정규식 지원을 통해 데이터베이스에서 복잡한 일치 논리를 구현할 수 있습니다. 이 기법은 다음과 같은 이유로 유용합니다.

- 오라클 데이터베이스에서 일치 논리를 중앙화함으로써 middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다. SQL 정규식 함수는 처리 논리를 데이터에 더욱 근접하게 이동함으로써 더욱 효율적인 솔루션을 제공합니다.
- Oracle Database 10*g* 이전에는 개발자가 일반적으로 클라이언트에서 데이터 검증 논리 코드를 작성했으므로 여러 클라이언트에 대해 중복되는 동일한 검증 논리가 필요했습니다. 서버측 정규식을 사용하여 제약 조건을 적용하면 이러한 문제가 해결됩니다.
- 내장 SQL 및 PL/SQL 정규식 함수와 조건을 사용하여 Oracle Database 10*g*의 이전 버전보다 더욱 강력하면서도 간단하게 문자열을 조작할 수 있습니다.

SQL 및 PL/SQL에서 정규식 함수 및 조건 사용

함수 또는 조건 이름	설명
REGEXP_LIKE	LIKE 연산자와 유사하지만 간단한 패턴 일치(조건) 대신 정규식 일치를 수행합니다.
REGEXP_REPLACE	정규식 패턴을 검색하여 대체 문자열로 바꿉니다.
REGEXP_INSTR	정규식 패턴에 대해 문자열을 검색하고 일치가 발견된 위치를 반환합니다.
REGEXP_SUBSTR	지정된 문자열 내에서 정규식 패턴을 검색하고 일치하는 부분 문자열을 추출합니다.
REGEXP_COUNT	입력 문자열에서 패턴 일치가 발견되는 횟수를 반환합니다.

SQL 및 PL/SQL에서 정규식 함수 및 조건 사용

오라클 데이터베이스는 정규식을 사용하여 문자열을 검색 및 조작할 수 있는 일련의 SQL 함수를 제공합니다. 텍스트 리터럴, 바인드 변수 또는 CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2, VARCHAR2 (LONG 제외) 와 같은 문자 데이터를 포함하는 열에서 이러한 함수를 사용합니다. 정규식은 작은 따옴표로 묶어야 합니다. 이렇게 하면 SQL 함수가 전체 표현식을 해석하고 코드의 가독성을 높일 수 있습니다.

- REGEXP_LIKE: 이 조건은 패턴에 대한 문자 열을 검색합니다. Query의 WHERE 절에서 이 조건을 사용하여 지정한 정규식과 일치하는 행을 반환합니다.
- REGEXP_REPLACE: 이 함수는 문자 열에서 패턴을 검색하고 해당 패턴의 각 발생 값을 지정한 패턴으로 대체합니다.
- REGEXP_INSTR: 이 함수는 문자열에서 지정된 정규식 패턴의 발생 값을 검색합니다. 찾고자 하는 발생 값과 검색 시작 위치를 지정합니다. 이 함수는 일치가 발견된 문자열의 위치를 나타내는 정수를 반환합니다.
- REGEXP_SUBSTR: 이 함수는 지정한 정규식 패턴과 일치하는 실제 부분 문자열을 반환합니다.
- REGEXP_COUNT: 이 함수는 입력 문자열에서 패턴 일치가 발견되는 횟수를 반환합니다.

단원 내용

- 정규식 소개
- **정규식에서 메타 문자 사용**
- 정규식 함수 사용:
  - REGEXP_LIKE
  - REGEXP_REPLACE
  - REGEXP_INSTR
  - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용

ORACLE

메타 문자란?

- 메타 문자는 대체 문자, 반복 문자, 일치하지 않는 문자 또는 일련의 문자와 같이 특별한 의미를 지닌 특수 문자입니다.
- 패턴 일치에서 여러 개의 미리 정의된 메타 문자 기호를 사용할 수 있습니다.
- 예를 들어, `^(f|ht)tps?:$` 정규식은 문자열 시작 부분에서 다음을 검색합니다.
  - 리터럴 f 또는 ht
  - t 리터럴
  - p 리터럴(선택적으로 다음에 s 리터럴이 나옴)
  - 문자열 끝부분의 ":" 리터럴

ORACLE

메타 문자란?

슬라이드의 정규식은 http:, https:, ftp: 및 ftps: 문자열을 일치시킵니다.

참고: 정규식의 메타 문자 전체 리스트는 *Oracle Database Advanced Application Developer's Guide 11g Release 2*를 참조하십시오.

정규식에서 메타 문자 사용

구문	설명
.	지원되는 character set에서 NULL을 제외한 임의의 문자와 일치
+	한 번 이상 발생 수 일치
?	0 또는 1번 발생 수 일치
*	선행 하위식의 0번 이상 발생 수 일치
{m}	선행 표현식의 정확히 *m*번 발생 수 일치
{m, }	선행 하위식과 최소 *m*번 이상 발생 수 일치
{m,n}	선행 하위식의 최소 *m*번 이상, 최대 *n*번 이하 발생 수 일치
[…]	괄호 안의 리스트에 있는 임의의 단일 문자와 일치
|	여러 대안 중 하나와 일치
(...)	괄호로 묶인 표현식을 한 단위로 취급합니다. 하위식은 리터럴의 문자열이나 연산자를 포함한 복잡한 표현식이 될 수 있습니다.

정규식 함수에서 메타 문자 사용

임의의 문자, ".": **a.b**는 문자열 **abb**, **acb**, **adb**와 일치하지만 **acc**와는 일치하지 않습니다.

하나 이상, "+": **a+**는 문자열 **a**, **aa**, **aaa**와 일치하지만 **bbb**와는 일치하지 않습니다.

0 또는 1, "?": **ab?c**는 문자열 **abc**, **ac**와 일치하지만 **abbc**와는 일치하지 않습니다.

0 이상, "*": **ab*c**는 문자열 **ac**, **abc**, **abbc**와 일치하지만 **abb**와는 일치하지 않습니다.

정확한 숫자, "{m}": **a{3}**는 문자열 **aaa**와 일치하지만 **aa**와는 일치하지 않습니다.

최소 숫자, "{m,}": **a{3,}**은 문자열 **aaa**, **aaaa**와 일치하지만 **aa**와는 일치하지 않습니다.

사이 숫자, "{m,n}": **a{3,5}**는 문자열 **aaa**, **aaaa**, **aaaaa**와 일치하지만 **aa**와는 일치하지 않습니다.

문자 리스트 일치, "[…]": **[abc]**는 문자열 **all**, **bill**, **cold**의 첫번째 문자와 일치하지만 **doll**의 어떠한 문자와도 일치하지 않습니다.

또는 "|": **a|b**는 문자 **a** 또는 문자 **b**와 일치합니다.

하위식, "(…)":**(abc)?def**는 선택적 문자열 **abc** 다음에 **def**가 나옵니다. 이 표현식은 **abcdefghi** 및 **def**와 일치하지만 **ghi**와는 일치하지 않습니다. 하위식은 리터럴의 문자열이나 연산자를 포함한 복잡한 표현식이 될 수 있습니다.

정규식에서 메타 문자 사용

구문	설명
^	문자열 시작 부분과 일치
$	문자열 끝부분과 일치
\	표현식에서 후속 메타 문자를 리터럴로 처리합니다.
\n	괄호 안에 그룹화된 *n*번째 (1–9) 선행 하위식과 일치합니다. 괄호는 표현식이 기억되도록 만들고 backreference에서 표현식을 참조합니다.
\d	숫자 문자
[:class:]	지정된 POSIX 문자 클래스에 속한 임의의 문자와 일치
[^:class:]	괄호 안의 리스트에 *없는* 임의의 단일 문자와 일치

정규식 함수에서 메타 문자 사용(계속)

행 앵커의 시작/끝, "^" 및 "$":** **^def**는 문자열 **defghi**의 **def**와 일치하지만 **abcdef**의 **def**와는 일치하지 않습니다. **def$는 문자열 **abcdef**의 **def**와 일치하지만 문자열 **defghi**의 **def**와는 일치하지 않습니다.

이스케이프 문자 "\": **\+**는 **+**를 검색합니다. 문자열 **abc+def**의 더하기(+) 문자와 일치하지만 **Abcdef**와는 일치하지 않습니다.

Backreference, "\n": **(abc|def)xy\1**은 문자열 **abcxyabc** 및 **defxydef**와 일치하지만 **abcxydef** 또는 **abcxy**와는 일치하지 않습니다. backreference를 사용하면 실제 문자열을 미리 알고 있지 않아도 반복되는 문자열을 검색할 수 있습니다. 예를 들어, **^(.*)\1$** 표현식은 동일한 문자열의 2개의 인접한 instance로 구성된 행과 일치합니다.

숫자 문자, "\d": ^\[\d{3}\] \d{3}-\d{4}$ 표현식은 [650] 555-1212와 일치하지만 650-555-1212와는 일치하지 않습니다.

문자 클래스, "[:class:]": **[[:upper:]]+**는 하나 이상의 연속되는 대문자를 검색합니다. 이 경우 문자열 **abcDEFghi**의 **DEF**와 일치하지만 문자열 **abcdefghi**와는 일치하지 않습니다.

비일치 문자 리스트 (또는 클래스), "[^...]": **[^abc]**는 문자열 **abcdef**의 문자 d와 일치하지만 문자 **a**, **b** 또는 **c**와는 일치하지 않습니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- **정규식 함수 사용:**
  - **REGEXP_LIKE**
  - **REGEXP_REPLACE**
  - **REGEXP_INSTR**
  - **REGEXP_SUBSTR**
- 하위식 액세스
- REGEXP_COUNT 함수 사용

ORACLE

정규식 함수 및 조건: 구문

```
REGEXP_LIKE   (source_char, pattern [,match_option]

REGEXP_INSTR  (source_char, pattern [, position
               [, occurrence [, return_option
               [, match_option [, subexpr]]]]])

REGEXP_SUBSTR (source_char, pattern [, position
               [, occurrence [, match_option
               [, subexpr]]]])

REGEXP_REPLACE(source_char, pattern [,replacestr
                [, position [, occurrence
                [, match_option]]]])

REGEXP_COUNT (source_char, pattern [, position
               [, occurrence [, match_option]]])
```


정규식 함수 및 조건: 구문

정규식 함수 및 조건의 구문은 다음과 같습니다.

- `source_char`: 검색 값 역할을 하는 문자 표현식
- `pattern`: 정규식, 텍스트 리터럴
- `occurrence`: Oracle 서버가 검색해야 하는 `source_char`에서 패턴의 발생 값을 나타내는 양의 정수. 기본값은 1입니다.
- `position`: Oracle 서버가 검색을 시작해야 하는 `source_char`의 문자를 나타내는 양의 정수. 기본값은 1입니다.
- `return_option`:
  - 0: 발생 값의 첫번째 문자의 위치를 반환합니다(기본값).
  - 1: 발생 값 다음의 문자 위치를 반환합니다.
- `Replacestr`: 패턴 대체 문자열
- `match_parameter`:
  - "`c`": 대소문자를 구분하는 일치 사용(기본값)
  - "`i`": 대소문자를 구분하지 않는 일치 사용
  - "`n`": 임의의 문자 일치 연산자 허용
  - "`m`": 소스 문자열을 다중 행으로 처리
- `subexpr`: 괄호로 묶인 패턴 부분. 하위식은 이 단원 뒷부분에서 다룹니다.

REGEXP_LIKE 조건을 사용하여 기본 검색 수행

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

ORACLE

REGEXP_LIKE 조건을 사용하여 기본 검색 수행

REGEXP_LIKE는 LIKE에서 수행하는 단순한 패턴 일치 대신 정규식 일치를 수행한다는 점을 제외하고 LIKE 조건과 일치합니다. 이 조건은 입력 character set에 의해 정의된 문자를 사용하여 문자열을 평가합니다.

REGEXP_LIKE의 예제

EMPLOYEES 테이블에 대한 이 query에서는 first name에 Steven 또는 Stephen이 포함된 모든 사원이 표시됩니다. 다음은 query에 사용된 표현식 '^Ste(v|ph)en$'에 대한 설명입니다.

- ^는 표현식의 시작을 나타냅니다.
- $는 표현식의 끝을 나타냅니다.
- |는 둘 중 하나(또는)를 나타냅니다.

REGEXP_REPLACE 함수를 사용하여 패턴 대체

```
REGEXP_REPLACE(source_char, pattern [,replacestr
[, position [, occurrence [, match_option]]]])
```

```
SELECT REGEXP_REPLACE(phone_number, '\.','-') AS phone
FROM employees;
```

원본

	LAST_NAME	PHONE
1	OConnell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

일부 결과

	LAST_NAME	PHONE
1	OConnell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555

ORACLE

REGEXP_REPLACE 함수를 사용하여 패턴 대체

REGEXP_REPLACE 함수를 사용하여 마침표(.) 구분자를 대시(-) 구분자로 대체하도록 전화 번호 형식을 다시 지정할 수 있습니다. 다음은 정규식 예제에서 사용되는 각 요소에 대한 설명입니다.

- phone_number는 소스 열입니다.
- '\.'는 검색 패턴입니다.
  - 작은 따옴표('')를 사용하여 리터럴 문자 마침표(.)를 검색합니다.
  - 백슬래시(\)를 사용하여 일반적으로 메타 문자로 처리되는 문자를 검색합니다.
- '-'는 대체 문자열입니다.

REGEXP_INSTR 함수를 사용하여 패턴 찾기

```
REGEXP_INSTR  (source_char, pattern [, position [,
   occurrence [, return_option [, match_option]]]])
```

```
SELECT street_address,
REGEXP_INSTR(street_address,'[[:alpha:]]') AS
   First_Alpha_Position
FROM locations;
```

	STREET_ADDRESS	FIRST_ALPHA_POSITION
1	1297 Via Cola di Rie	6
2	93091 Calle della Testa	7
3	2017 Shinjuku-ku	6
4	9450 Kamiya-cho	6

ORACLE

REGEXP_INSTR 함수를 사용하여 패턴 찾기

이 예제에서는 REGEXP_INSTR 함수로 주소를 검색하여 대소문자 여부와 상관없이 첫번째 알파벳 문자의 위치를 찾습니다. [:*class*:]는 문자 클래스를 나타내며 해당 클래스 내의 임의의 문자와 일치합니다. [:alpha:]는 임의의 알파벳 문자와 일치합니다. 부분적인 결과가 표시됩니다.

다음은 query에 사용된 표현식 '[[:alpha:]]'에 대한 설명입니다.

- [는 표현식을 시작합니다.
- [:alpha:]는 알파벳 문자 클래스를 나타냅니다.
-]는 표현식을 종료합니다.

참고: POSIX 문자 클래스 연산자를 사용하면 특정 POSIX 문자 클래스의 멤버인 문자 리스트 내에서 표현식을 검색할 수 있습니다. 이 연산자를 사용하여 대문자와 같은 특정 형식을 검색하거나 자릿수 또는 구두점 문자 등의 특수 문자를 검색할 수 있습니다. 전체 POSIX 문자 클래스를 지원합니다. [:class:] 구문을 사용합니다. 여기서 class는 검색할 POSIX 문자 클래스의 이름을 나타냅니다. 정규식 [[:upper:]]+는 하나 이상의 연속하는 대문자를 검색합니다.

REGEXP_SUBSTR 함수를 사용하여 부분 문자열 추출

```
REGEXP_SUBSTR (source_char, pattern [, position
               [, occurrence [, match_option]]])
```

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ') AS Road
FROM locations;
```

	ROAD
1	Via
2	Calle
3	(null)
4	(null)
5	Jabberwocky

REGEXP_SUBSTR 함수를 사용하여 부분 문자열 추출

이 예제에서는 LOCATIONS 테이블에서 road name을 추출합니다. 이를 위해 REGEXP_SUBSTR 함수를 사용하여 STREET_ADDRESS 열에서 첫번째 공백 뒤에 있는 내용을 반환합니다. 다음은 query에 사용된 표현식 '[^]+ '에 대한 설명입니다.

- [는 표현식을 시작합니다.
- ^는 NOT을 나타냅니다.
- 공백을 나타냅니다.
-]는 표현식을 종료합니다.
- +는 1 이상을 나타냅니다.
- 공백을 나타냅니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP_LIKE
  - REGEXP_REPLACE
  - REGEXP_INSTR
  - REGEXP_SUBSTR
- **하위식 액세스**
- REGEXP_COUNT 함수 사용

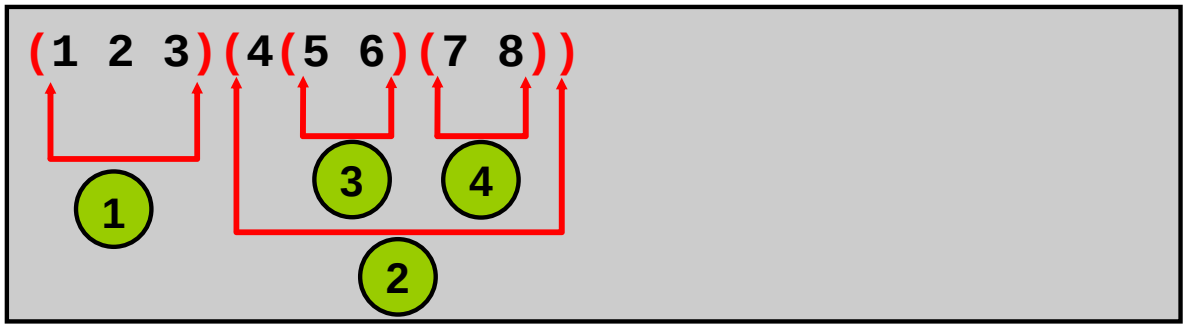
ORACLE

하위식

다음 표현식을 살펴보십시오.

```
(1 2 3)(4(5 6)(7 8))
```

하위식은 다음과 같습니다.



ORACLE

하위식

Oracle Database 11*g*는 하위식에 액세스할 수 있도록 정규식 지원 파라미터를 제공합니다. 슬라이드 예제에 숫자 문자열이 표시되어 있습니다. 괄호는 숫자 문자열 내의 하위식을 식별합니다. 왼쪽에서 오른쪽, 외부 괄호에서 내부 괄호 순서로 읽을 경우 숫자 문자열의 하위식은 다음과 같습니다.

1. 123
2. 45678
3. 56
4. 78

`REGEXP_INSTR` 및 `REGEXP_SUBSTR` 함수를 사용하여 이러한 하위식을 검색할 수 있습니다.

정규식을 지원하는 하위식 사용

```
SELECT
  REGEXP_INSTR
① ('0123456789',       -- source char or search value
② '(123)(4(56)(78))', -- regular expression patterns
③ 1,                   -- position to start searching
④ 1,                   -- occurrence
⑤ 0,                   -- return option
⑥ 'i',                 -- match option (case insensitive)
⑦ 1)              -- sub-expression on which to search
   "Position"
FROM dual;
```

Position
2

정규식을 지원하는 하위식 사용

REGEXP_INSTR 및 REGEXP_SUBSTR에는 계산되는 정규식의 특정 부분 문자열을 대상으로 할 수 있는 선택적 SUBEXPR 파라미터가 있습니다.

슬라이드의 예제에서 하위식 리스트의 첫번째 하위식 패턴을 검색하려고 합니다. 제공된 예제는 REGEXP_INSTR 함수에 대한 여러 파라미터를 식별합니다.

1. 검색 중인 문자열이 식별됩니다.
2. 하위식이 식별됩니다. 첫번째 하위식은 123이고 두번째는 45678, 세번째는 56, 네번째는 78입니다.
3. 세번째 파라미터는 검색을 시작할 위치를 식별합니다.
4. 네번째 파라미터는 찾으려는 패턴의 발생 값을 식별합니다. 1은 첫번째 발생 값을 찾는다는 의미입니다.
5. 다섯번째 파라미터는 반환 옵션입니다. 발생 값의 첫번째 문자의 위치입니다. 1을 지정하면 발생 값 다음의 문자 위치가 반환됩니다.
6. 여섯번째 파라미터는 검색에서 대소문자를 구분해야 하는지 여부를 식별합니다.
7. 마지막 파라미터는 Oracle Database 11*g*에서 추가된 파라미터입니다. 이 파라미터는 찾으려는 하위식을 지정합니다. 제공된 예제에서는 첫번째 하위식을 검색하며, 결과는 123입니다.

*n*번째 하위식에 액세스하는 이유

- 더욱 실제적인 사용: DNA 시퀀싱
- 쥐의 DNA에서 면역에 필요한 단백질을 식별하는 특정 하위 패턴을 찾으려고 합니다.

```
SELECT
    REGEXP_INSTR('ccacctttccctccactcctcacgttctcacctgtaaagcgtccctc
    cctcatccccatgccccttaccctgcagggtagagtaggctagaaaccagagagctccaagc
    tccatctgtggagaggtgccatccttgggctgcagagagaggagaatttgccccaaagctgcc
    tgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttca
    ccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagag
    gctcttgggtctgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggc
    atgtaggggcgtggggatgcgctctgctctgctctcctctcctgaacccctgaaccctctggc
    taccccagagcacttagagccag',
                    '(gtc(tcac)(aaag))',
                     1, 1, 0, 'i',
                     1) "Position"
FROM dual;
```

	Position
1	195

ORACLE

*n*번째 하위식에 액세스하는 이유

생명 과학 분야에서 추가 처리를 위해 DNA 시퀀스로부터 하위식 일치의 오프셋을 추출해야 하는 경우가 있습니다. 예를 들어, `gtc`가 앞에 나오고 이어서 `tcac`, `aaag`가 나오는 DNA 시퀀스의 시작 오프셋과 같은 특정 단백질 시퀀스를 찾으려고 합니다. 이러한 결과를 얻으려면 일치가 발견된 위치를 반환하는 `REGEXP_INSTR` 함수를 사용하면 됩니다.

슬라이드 예제에서는 첫번째 하위식 `(gtc)`의 위치가 반환됩니다. `gtc`는 DNA 문자열의 195 위치에서 시작되는 것으로 나타납니다.

슬라이드 예제를 수정하여 두번째 하위식 `(tcac)`를 검색하면 query 결과는 다음과 같이 출력됩니다. `tcac`는 DNA 문자열의 198 위치에서 시작되는 것으로 나타납니다.

	Position
1	198

슬라이드 예제를 수정하여 세번째 하위식 `(aaag)`를 검색하면 query 결과는 다음과 같이 출력됩니다. `aaag`는 DNA 문자열의 202 위치에서 시작하는 것으로 나타납니다.

	Position
1	202

REGEXP_SUBSTR: 예제

```
SELECT
  REGEXP_SUBSTR
①('acgctgcactgca', -- source char or search value
② 'acg(.*)gca',    -- regular expression pattern
③ 1,               -- position to start searching
④ 1,               -- occurrence
⑤ 'i',             -- match option (case insensitive)
⑥ 1)               -- sub-expression
 "Value"
FROM dual;
```

Value
1 ctgcact

REGEXP_SUBSTR: 예제

슬라이드의 예제는 다음과 같이 설명됩니다.

1. `acgctgcactgca`는 검색할 소스입니다.
2. `acg(.*)gca`는 검색할 패턴입니다. `acg` 다음에 `gca`가 오는 문자열을 찾습니다. `acg`와 `gca` 사이에 다른 문자가 포함될 수 있습니다.
3. 소스의 첫번째 문자에서 검색을 시작합니다.
4. 패턴의 첫번째 발생 값을 검색합니다.
5. 소스에서 대소문자를 구분하지 않는 일치를 사용합니다.
6. 대상으로 지정할 *n*번째 하위식을 식별하는 음이 아닌 정수 값을 사용합니다. 이는 하위식 파라미터입니다. 이 예제에서 `1`은 첫번째 하위식을 나타냅니다. 0–9 사이의 값을 사용할 수 있습니다. 0은 하위식이 대상으로 지정되지 않음을 의미합니다. 이 파라미터에 대한 기본값은 0입니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP_LIKE
  - REGEXP_REPLACE
  - REGEXP_INSTR
  - REGEXP_SUBSTR
- 하위식 액세스
- **REGEXP_COUNT 함수 사용**

ORACLE

REGEXP_COUNT 함수 사용

```
REGEXP_COUNT (source_char, pattern [, position
                [, occurrence [, match_option]]])
```

```
SELECT REGEXP_COUNT(
    'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttaccctgcag
    ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagaggag
    aatttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagtt
    ttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtc
    tgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctg
    ctctgctctcctctcctgaacccctgaacccctctggctaccccagagcacttagagccag',
    'gtc') AS Count
FROM dual;
```

	COUNT
1	4

ORACLE

REGEXP_COUNT 함수 사용

REGEXP_COUNT 함수는 입력 character set에서 정의된 문자를 사용하여 문자열을 평가합니다. 이 함수는 패턴의 발생 수를 나타내는 정수를 반환합니다. 일치가 발견되지 않으면 이 함수는 0을 반환합니다.

슬라이드 예제에서는 REGEXP_COUNT 함수를 사용하여 DNA 부분 문자열의 발생 수를 판별합니다.

다음 예제는 문자열 123123123123에서 패턴 123이 발생하는 횟수가 세 번임을 보여줍니다. 검색은 문자열의 두번째 위치에서 시작됩니다.

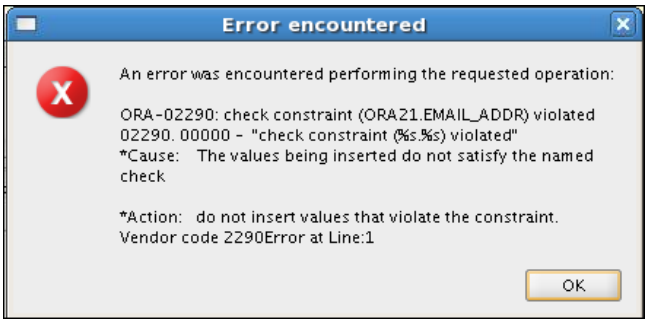
```
SELECT REGEXP_COUNT
   ('123123123123', -- source char or search value
   '123',           -- regular expression pattern
    2,              -- position where the search should start
   'i')             -- match option (case insensitive)
   As Count
FROM dual;
```

	COUNT
1	3

정규식 및 Check 제약 조건: 예제

```
ALTER TABLE emp8
  ADD CONSTRAINT email_addr
  CHECK(REGEXP_LIKE(email,'@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES
  (500,'Christian','Patel','ChrisP2creme.com',
   1234567890,'12-Jan-2004','HR_REP',2000,null,102,40);
```



ORACLE

정규식 및 Check 제약 조건: 예제

CHECK 제약 조건에서도 정규식을 사용할 수 있습니다. 이 예제에서는 CHECK 제약 조건이 EMPLOYEES 테이블의 EMAIL 열에 추가되었습니다. 이렇게 하면 "@" 기호를 포함한 문자열만을 받아들입니다. 제약 조건을 테스트합니다. 전자 메일 주소에 필요한 필수 기호가 포함되어 있지 않으므로 CHECK 제약 조건을 위반했습니다. NOVALIDATE 절은 기존 데이터를 검사했는지 여부를 확인합니다.

슬라이드 예제에서는 다음 코드를 사용하여 emp8 테이블을 생성합니다.

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

참고: 슬라이드의 예제는 SQL Developer의 "Execute Statement" 옵션을 사용하여 실행합니다. "Run Script" 옵션을 사용하는 경우 출력 형식이 달라집니다.

퀴즈

SQL 및 PL/SQL에서 정규식을 사용할 경우 옳은 설명을 고르십시오.

1. middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다.
2. 클라이언트에서 데이터 검증 논리를 작성할 필요가 없습니다.
3. 서버의 제약 조건을 적용할 수 있습니다.

ORACLE

정답: 1, 2, 3

요약

이 단원에서는 정규식을 사용하여 문자열을 검색, 일치 및 대체하는 방법을 배웠습니다.

ORACLE

요약

이 단원에서는 정규식 지원 기능의 사용법을 배웠습니다. 정규식 지원은 SQL 및 PL/SQL에서 모두 사용할 수 있습니다.

연습 7: 개요

이 연습에서는 정규식 함수를 사용하여 다음 작업을 수행하는 방법을 다룹니다.

- **데이터 검색, 대체 및 조작**
- **새 CONTACTS 테이블을 생성하고 해당 전화 번호가 특정 표준 형식으로 데이터베이스에 입력되도록 p_number 열에 CHECK 제약 조건을 추가**
- **다양한 형식을 사용하여 p_number 열에 일부 전화 번호 추가 테스트**

ORACLE

연습 7: 개요

이 연습에서는 정규식 함수를 사용하여 데이터를 검색, 대체 및 조작합니다. 또한 새 CONTACTS 테이블을 생성하고 해당 전화 번호가 특정 표준 형식으로 데이터베이스에 입력되도록 p_number 열에 CHECK 제약 조건을 추가합니다.