Using Regular Expressions

Purpose

In this module, you will learn how to use regular expression support. This is a new feature introduced in Oracle Database 10 *g* .

Topics

This module will discuss the following:

- Overview
- Prerequisites
- Using Regular Expressions in Oracle Database 10 *g*
- Using Basics Searches
- Using the Multilingual Capabilities
- Regular Expressions and Check Constraints

Overview

Back to List of Topics

In Oracle Database 10 *g* , you can use both SQL and PL/SQL to implement regular expression support. Regular expressions are a method of describing both simple and complex patterns for searching and manipulating. String manipulation and searching contribute to a large percentage of the logic within a web based application. Usage ranges from the simple: find the word SAN FRANCISCO in a specified text; to the complex extract of all URLs from the text; to the more complex: find all words whose every second character is a vowel.

Oracle Database 10 *g* introduces support for Regular expressions. The implementation complies with the Portable Operating System for UNIX (POSIX) standard, controlled by the Institute of Electrical and Electronics Engineers (IEEE), for ASCII data matching semantics and syntax. Oracle's multi-lingual capabilities extend the matching capabilities of the operators beyond the POSIX standard.

When coupled with native SQL, the use of regular expressions allows for very powerful search and manipulation operations on any data stored in an Oracle database. You can use this feature to easily solve problems that would otherwise be very complex to program.

Prerequisites

Back to List

Before starting this module, you should have performed the following:

☐ **1.** Completed the Configuring Linux for the Installation of Oracle Database 10g lesson

☐ **2.** Completed the Installing the Oracle Database 10g on Linux lesson

☐ **3.** Download the regexp.zip into your working directory.

Using Regular Expressions in Oracle Database 10 *g*

Back to List of Topics

Matching Mechanism

If you have a string 'aabcd' and you specify to search for 'a(b|c)d', the search will look for 'a' followed by 'b' or by 'c' followed by 'd'.

Regular Expression:
'a(b|c)d'

String to Match:
'aabcd'

a	a	b	c	d	Description	Result
*					Look for 'a' and succeed	Match
	*				Look for 'b' and fail	No match
	*				Look for 'c' and fail, reset, and advance	No match
	*				Look for 'a' and succeed	Match
		*			Look for 'b' and succeed, remember 'c' as alternative	Match
			*		Look for 'd' and fail	No match
		*			Look for 'c' as last remembered alternative and fail, reset, and advance	No match
		*			Look for 'a' and fail, reset, and advance	No match
			*		Look for 'a' and fail, reset, and advance	No match
				*	Look for 'a' and fail, reset, and advance	No match

Given the string 'aabcd', 'a(b|c)d' does not match it.

To implement regular expression support in either SQL or PL/SQL, you use a new set of functions. These functions are:

Function Name	Description
`REGEXP_LIKE`	Similar to the `LIKE` operator, but performs regular expression matching instead of simple pattern matching
`REGEXP_INSTR`	Searches for a given string for a regular expression pattern and returns the position were the match is found
`REGEXP_REPLACE`	Searches for a regular expression pattern and replaces it with a replacement string
`REGEXP_SUBSTR`	Searches for a regular expression pattern within a given string and returns the matched substring

Meta Characters

Meta characters are special characters that have a special meaning, such as a wild card character, or a repeating character, or a non matching character, or a range of characters.

You can use several predefined meta character symbols in the pattern matching with the functions.

Symbol	Description
*	Matches zero or more occurrences
|	Alternation operator for specifying alternative matches
^/$	Matches the start-of-line and end-of-line
[]	Bracket expression for a matching list matching any one of the expressions represented in the list
[^exp]	If carrot is inside the bracket, it negates the expression
{m}	Matches exactly m times
{m,n}	Matches at least m times but no more than n times

[: :]	Specifies a character class and matches any character in that class
\	Can have 4 different meanings: 1. Stand for itself. 2. Quote the next character. 3. Introduce an operator. 4. Do nothing.
+	Matches one or more occurrence
?	Matches zero or one occurrence
.	Matches any character in the supported character set, except NULL
()	Grouping expression, treated as a single subexpression
\n	Backreference expression
[==]	Specifies equivalence classes
[..]	Specifies one collation element, such as a multicharacter element

Using Basic Searches

Back to List of Topics

The following examples demonstrate the use of the regular expression functions. Perform the following steps:

☐ **1.** Start SQL*Plus.

Connect to Oracle with the userid and password **`oe/oe`**

☐ **2.** Examine the syntax of the `REGEXP_LIKE` function:

REGEXP_LIKE(srcstr, pattern [,match_option])

Where

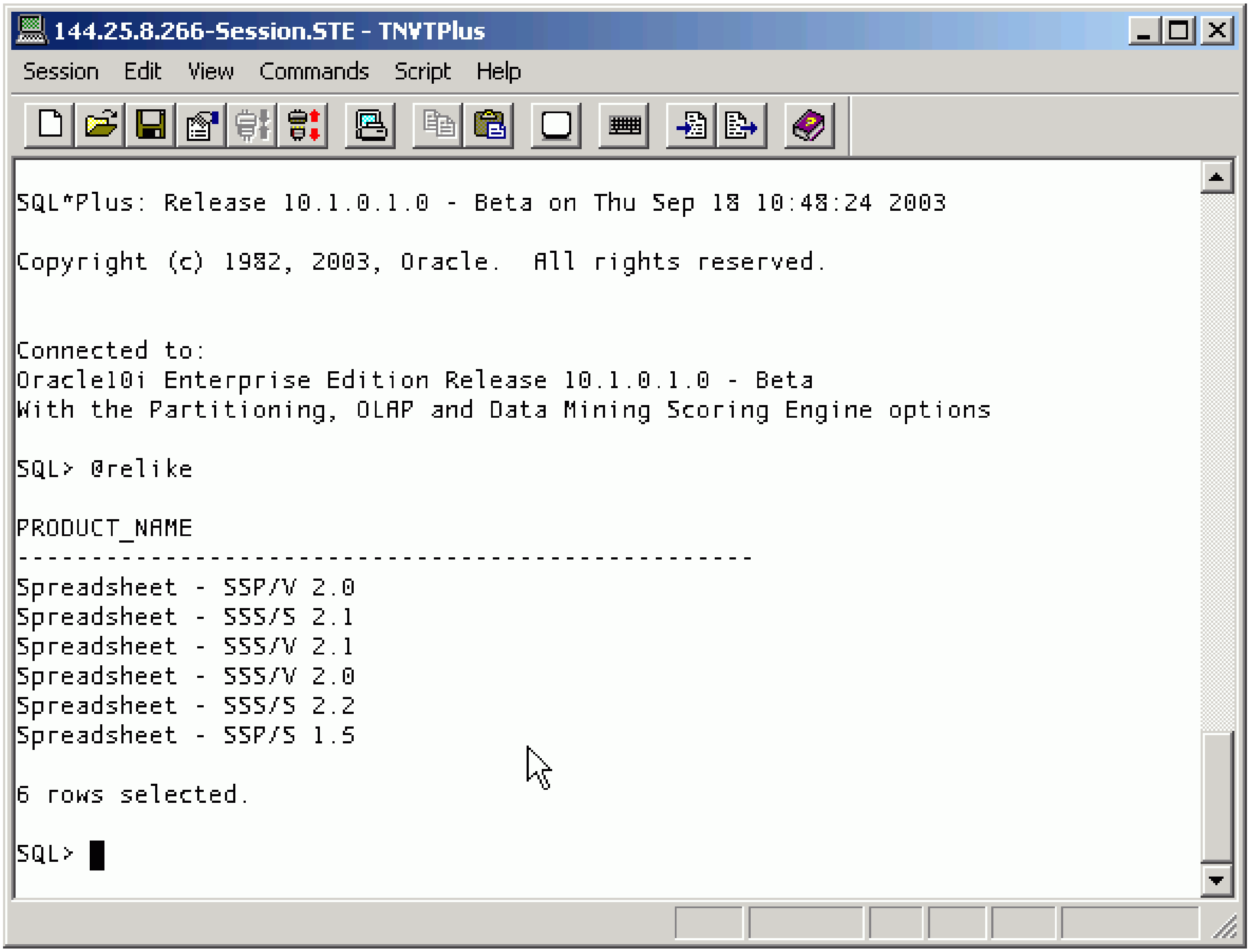
`srcstr:` search value
`pattern:` regular expression
`match_option:` option to change default matching, it can include one or more of the following values:
`'c'` - use case sensitive matching (default)
`'i'` - use case insensitive matching
`'n'` - allows match-any-character operator
`'m'` - treats source string as multiple line

To locate all products with names containing 'SSP/S', 'SSP/V', 'SSS/V', or 'SSS/S' in the `PRODUCT_NAME` column from the PRODUCT_INFORMATION table, execute the following script:

```
@relike.sql
```

The **`relike.sql`** contains the following SQL:

```
SELECT product_name
  FROM oe.product_information
  WHERE regexp_like (product_name, 'SS[PS]/[VS]');
```



```
SQL*Plus: Release 10.1.0.1.0 - Beta on Thu Sep 18 10:48:24 2003

Copyright (c) 1982, 2003, Oracle.  All rights reserved.


Connected to:
Oracle10i Enterprise Edition Release 10.1.0.1.0 - Beta
With the Partitioning, OLAP and Data Mining Scoring Engine options

SQL> @relike

PRODUCT_NAME
--------------------------------------------------
Spreadsheet - SSP/V 2.0
Spreadsheet - SSS/S 2.1
Spreadsheet - SSS/V 2.1
Spreadsheet - SSS/V 2.0
Spreadsheet - SSS/S 2.2
Spreadsheet - SSP/S 1.5

6 rows selected.

SQL>
```

☐ **3.** The `REGEXP_INSTR` function returns the position of a given pattern within a string. Examine the syntax:

REGEXP_INSTR(srcstr, pattern [, position [, occurrence
[, return_option [, match_option]]]])

Where

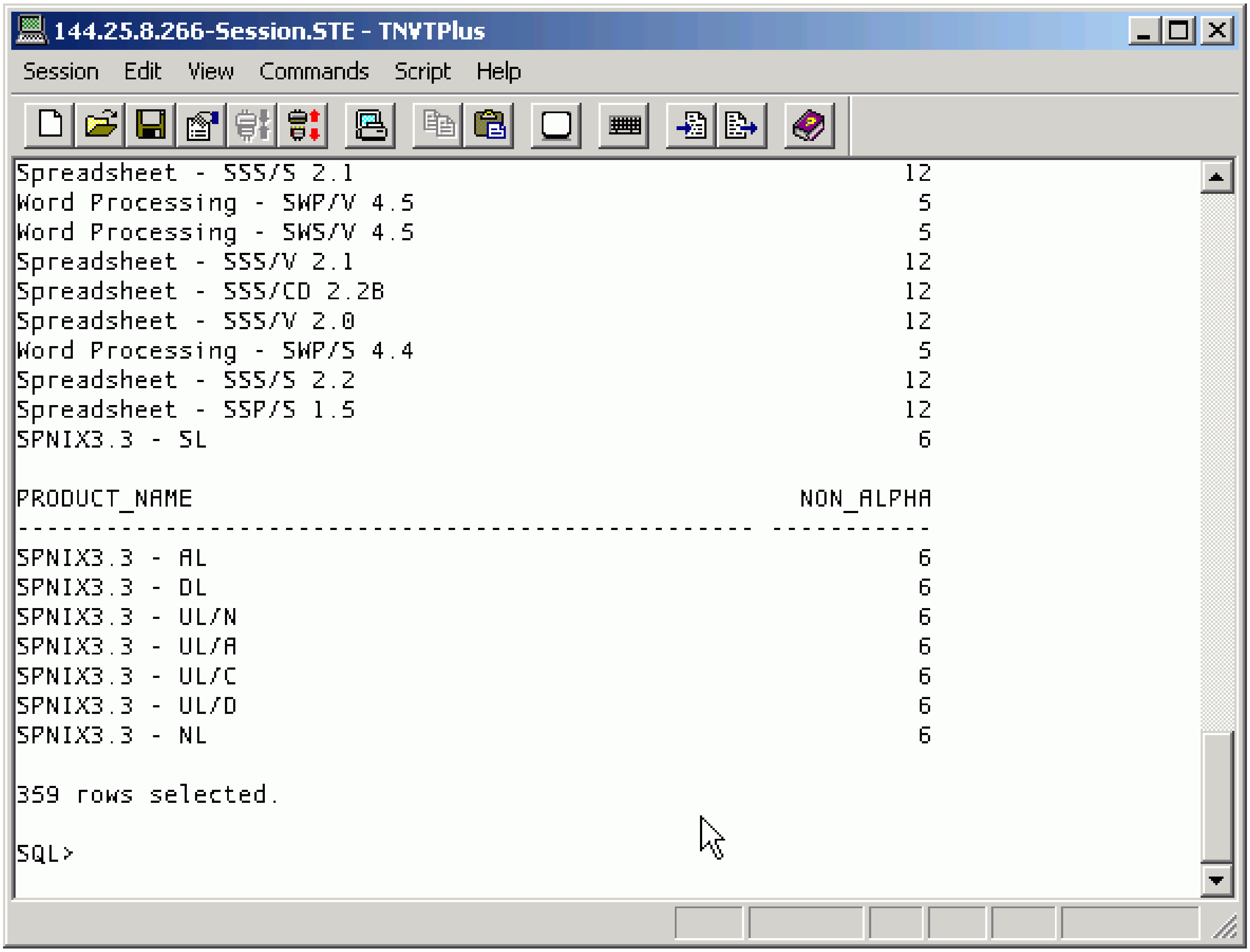
`position` : search starting position
`occurrence` : occurrence to search for
`return_option` : indicate the start or end position of occurrence
`match_option` : option to change default matching it can include one or more of the following values:
' `c` ' - use case sensitive matching (default)
' `i` ' - use case insensitive matching
' `n` ' - allows match-any-character operator
' `m` ' - treats source string as multiple line

To search the product names to find the location of the first non-alphabetic character, regardless of whether it is in upper or lower case, execute the following script:

```
@reinstr.sql
```

The **`reinstr.sql`** contains the following SQL:

```
COLUMN non_alpha FORMAT 9999999999

SELECT product_name, REGEXP_INSTR(product_name, '[^[:alpha:]]') non_alpha

   FROM   oe.product_information ;
```



```
Spreadsheet - SSS/S 2.1                                     12
Word Processing - SWP/V 4.5                                  5
Word Processing - SWS/V 4.5                                  5
Spreadsheet - SSS/V 2.1                                     12
Spreadsheet - SSS/CD 2.2B                                   12
Spreadsheet - SSS/V 2.0                                     12
Word Processing - SWP/S 4.4                                  5
Spreadsheet - SSS/S 2.2                                     12
Spreadsheet - SSP/S 1.5                                     12
SPNIX3.3 - SL                                                6

PRODUCT_NAME                                         NON_ALPHA
-------------------------------------------------- -----------
SPNIX3.3 - AL                                                6
SPNIX3.3 - DL                                                6
SPNIX3.3 - UL/N                                              6
SPNIX3.3 - UL/A                                              6
SPNIX3.3 - UL/C                                              6
SPNIX3.3 - UL/D                                              6
SPNIX3.3 - NL                                                6

359 rows selected.

SQL>
```

Note that `[^[:<class>:]]` implies a character class and matches any character from within that class; `[:alpha:]` matches with any alphabetic character. In this case, you are negating this expression by using the ^.

☐ **4.** The `REGEXP_SUBSTR` function returns a given string based on a pattern of occurrence. Examine the syntax:

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

Where

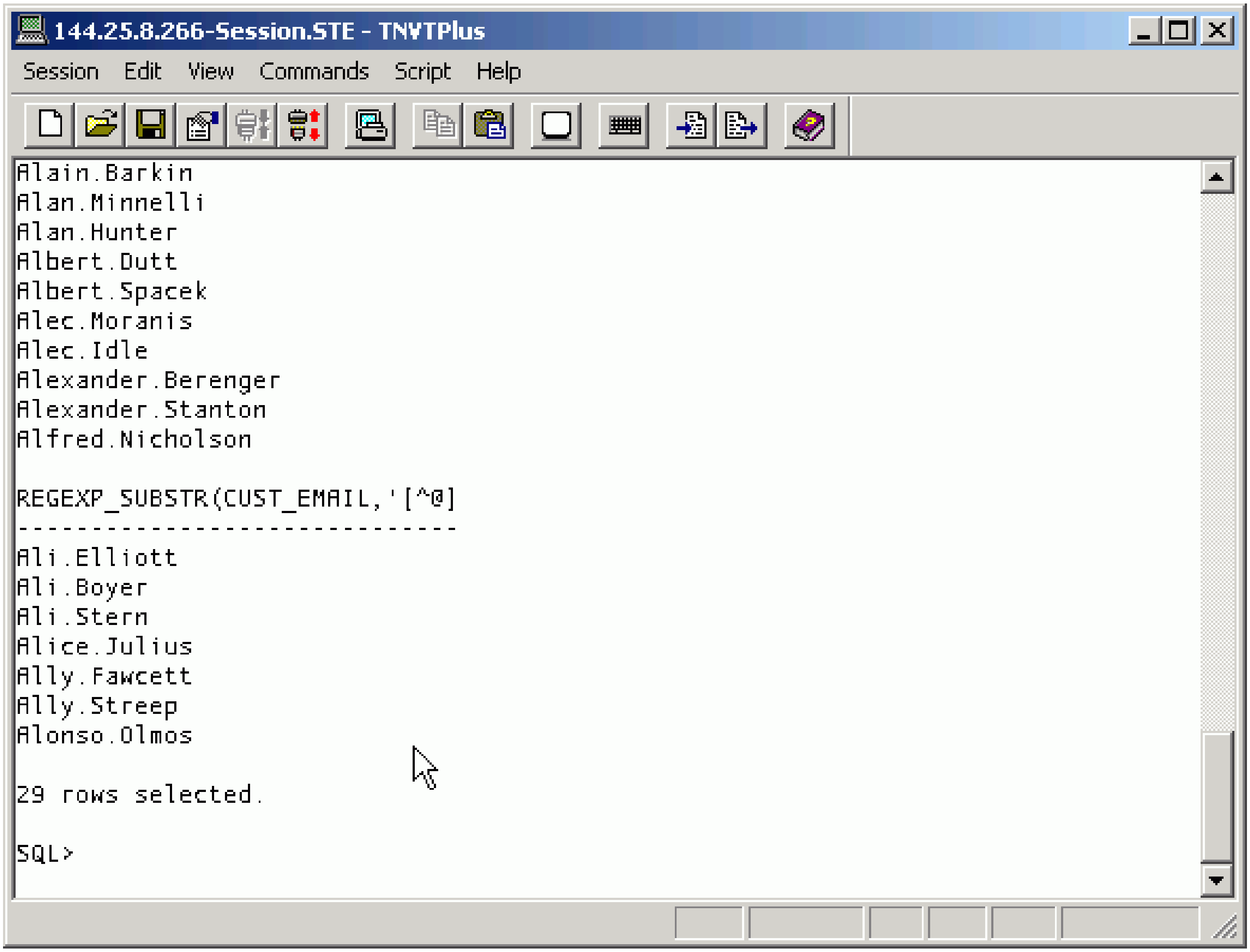
`position` : search starting position
`occurrence` : occurrence to search for
`match_option` : option to change default matching, it can include one or more of the following values:
‘ `c` ’ - use case sensitive matching (default)
‘ `i` ’ - use case insensitive matching
‘ `n` ’ - allows match-any-character operator
‘ `m` ’ - treats source string as multiple line

To extract the email names from the `CUSTOMERS` table. Extract only those email names for customers located in Switzerland. In order to do this, return the contents in the `CUST_EMAIL` column that precedes the '@'symbol for customers with an `NLS_TERRITORY` of Switzerland. execute the following script:

```
@resubstr.sql
```

The **`resubstr.sql`** contains the following SQL:

```
SELECT REGEXP_SUBSTR(cust_email, '[^@]+')

  FROM   oe.customers

  WHERE  nls_territory = 'SWITZERLAND' ;
```


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
Alain.Barkin
Alan.Minnelli
Alan.Hunter
Albert.Dutt
Albert.Spacek
Alec.Moranis
Alec.Idle
Alexander.Berenger
Alexander.Stanton
Alfred.Nicholson

REGEXP_SUBSTR(CUST_EMAIL,'[^@]
------------------------------
Ali.Elliott
Ali.Boyer
Ali.Stern
Alice.Julius
Ally.Fawcett
Ally.Streep
Alonso.Olmos

29 rows selected.

SQL>
```

Note that in this example, the result will return the first substring that does not have an @ symbol.

☐ **5.** The `REGEXP_REPLACE` function returns a "replaced" substring from a given string. Examine the syntax:

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

Where

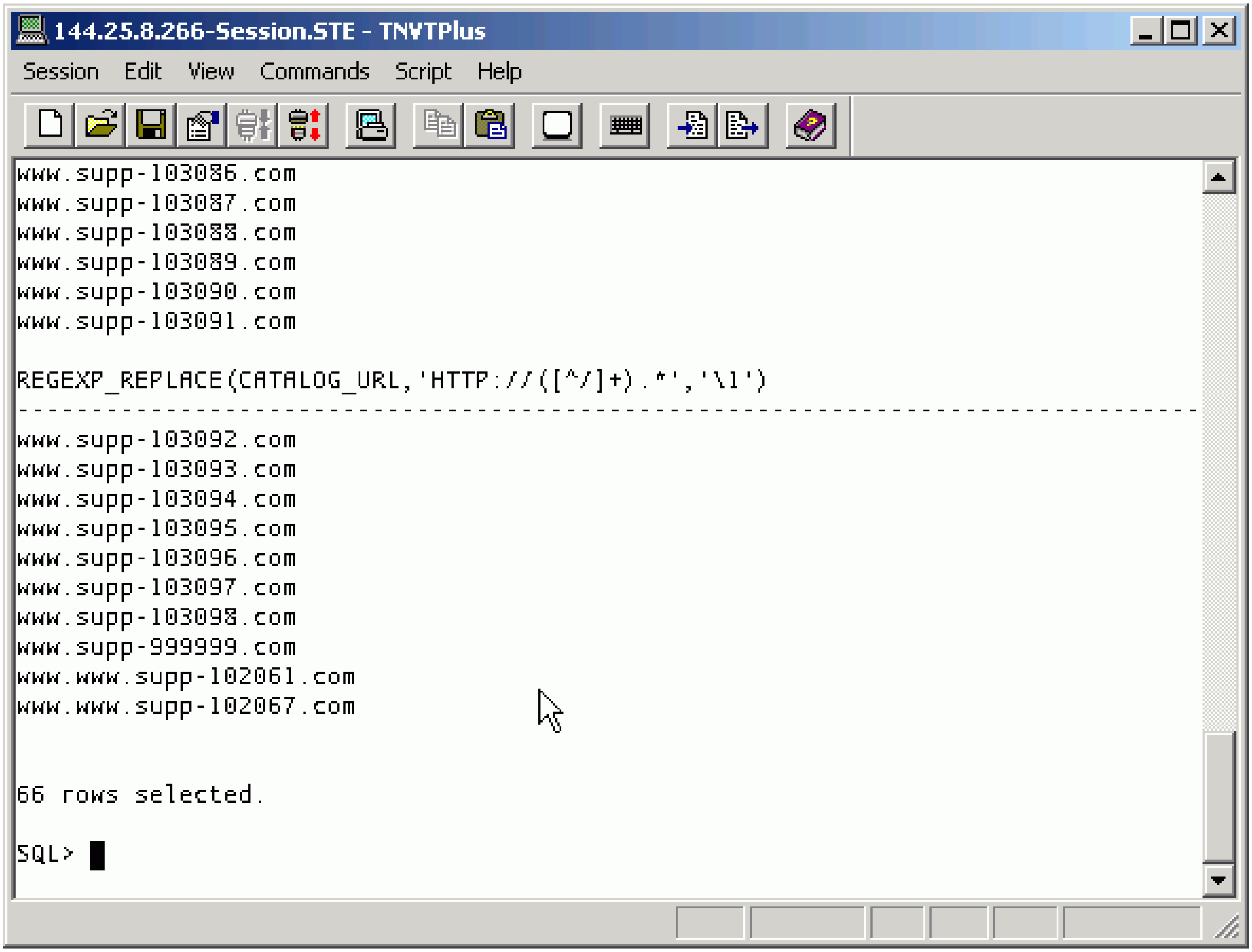
`position` : search starting position
`occurrence` : occurrence to search for
`replacestr` : character string replacing pattern
`match_option` : option to change default matching, it can include one or more of the following values:
' `c` ' - use case sensitive matching (default)
' `i` ' - use case insensitive matching
' `n` ' - allows match-any-character operator
' `m` ' - treats source string as multiple line

To return information from the `CATALOG_URL` column in the `PRODUCT_INFORMATION` table. Performing a full scan on the column would result in hundreds of rows being returned as it lists a specific html page location within a number of catalog domains. However, in this example find only the individual domain names themselves and not the lower level pages they contain. To find the domain names without all the extraneous information, use the `REGEXP_REPLACE` function, execute the following script:

```
@rereplace.sql
```

The **`rereplace.sql`** contains the following SQL:

```
SELECT UNIQUE REGEXP_REPLACE (catalog_url, 'http://([^/]+).*', '\1')
  FROM oe.product_information ;
```



```
www.supp-103086.com
www.supp-103087.com
www.supp-103088.com
www.supp-103089.com
www.supp-103090.com
www.supp-103091.com

REGEXP_REPLACE(CATALOG_URL,'HTTP://([^/]+).*','\1')
--------------------------------------------------------------------------------
www.supp-103092.com
www.supp-103093.com
www.supp-103094.com
www.supp-103095.com
www.supp-103096.com
www.supp-103097.com
www.supp-103098.com
www.supp-999999.com
www.www.supp-102061.com
www.www.supp-102067.com


66 rows selected.

SQL> 
```

And explanation of how the string was processed is as follows:

http://	The expression starts by looking for this string literal; there are no special metacharacters here.
([^/]+)	Then the expression looks for a series of characters as long as they are *not* forward slash (/)
.*	The expression finishes up by consuming the rest of the string with this part of the expression.
\1	The matching expression is replaced with backreference 1 which is whatever was matched between the first set of parentheses.

Using the Multilingual Capabilities

Back to List of Topics

Regular expression functions support multilingual capabilities. Regular expressions can be used in locale sensitive applications. In this example you will combine the use of regular expressions with Oracle's NLS Language feature. Perform the following:

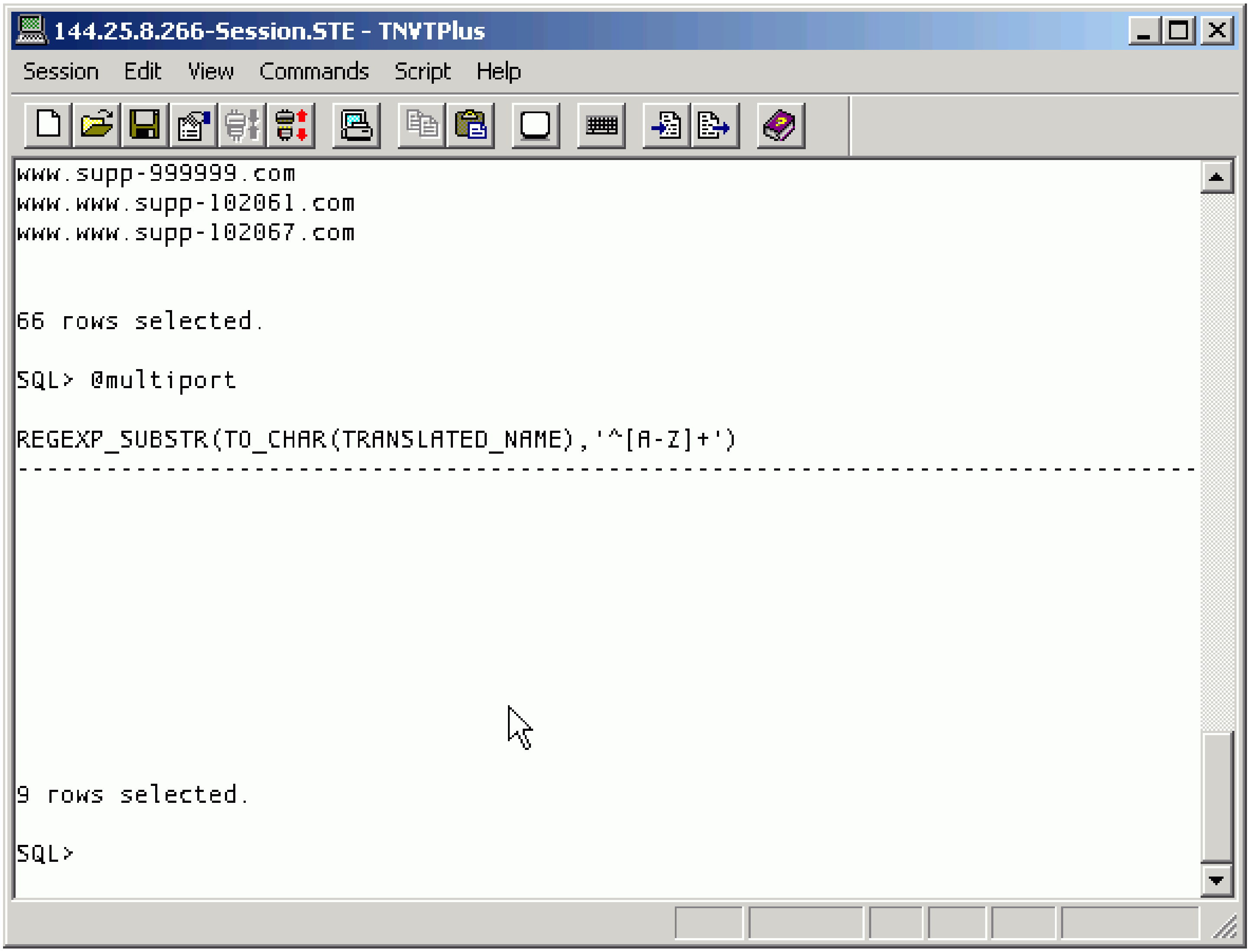
☐ **1.** You will seek the Product Description in the Portuguese language by executing the following script:

```
@multiport.sql
```

The `multiport.sql` contains the following SQL:

```
SELECT regexp_substr(to_char(translated_name), '^[a-z]+')
FROM   oe.product_descriptions
WHERE  language_id = 'PT'
AND    translated_name like 'G%' ;
```

Note that the data is not displayed.



144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
www.supp-999999.com
www.www.supp-102061.com
www.www.supp-102067.com


66 rows selected.

SQL> @multiport

REGEXP_SUBSTR(TO_CHAR(TRANSLATED_NAME),'^[A-Z]+')
--------------------------------------------------------------------------------








9 rows selected.

SQL>
```

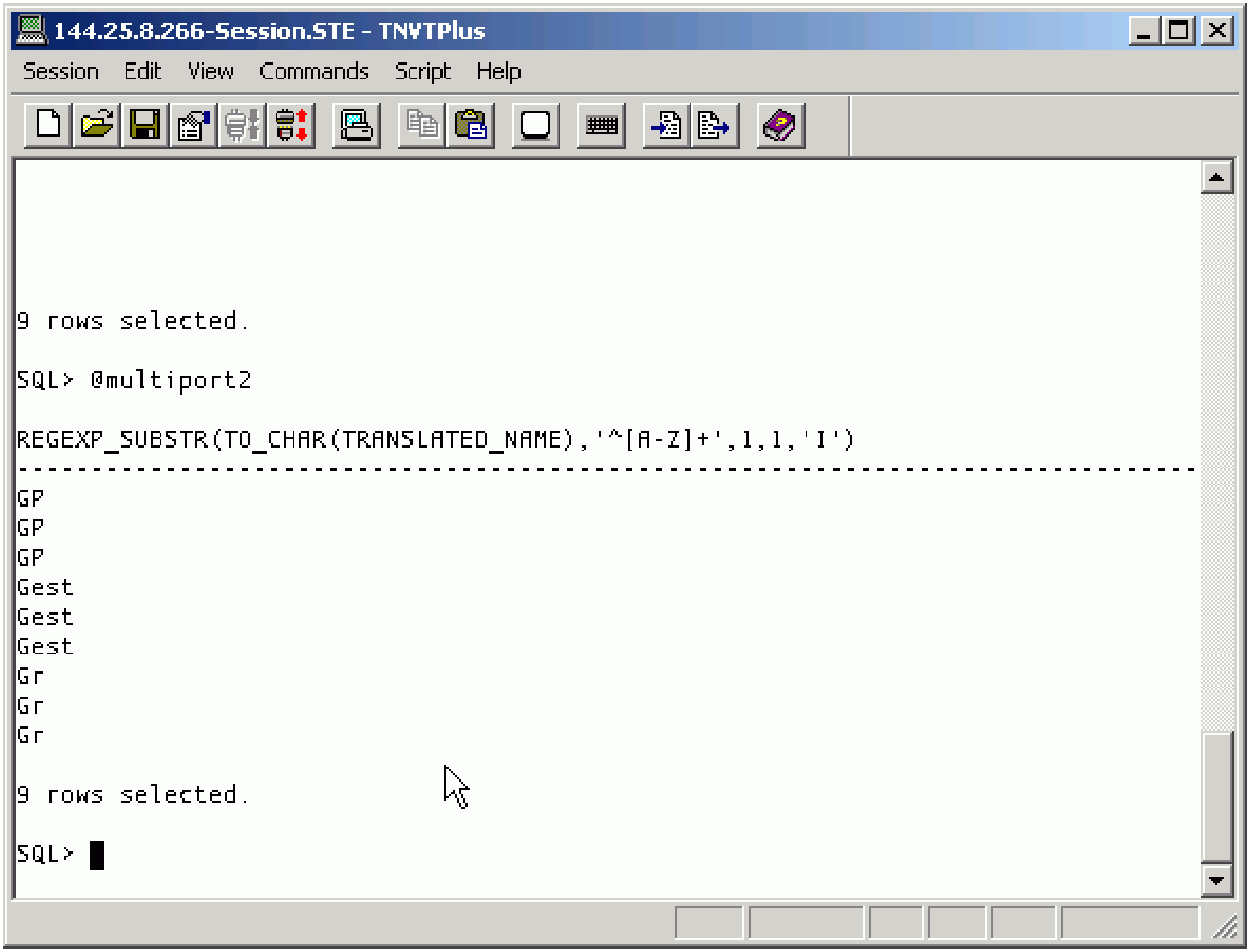
Note the ^ is outside the bracket, there we are searching for any strings or substrings that start with any character from a to z.

☐ **2.** Perform the same query, but this time use the case insensitive "i" switched on. Execute the following script:

```
@multiport2.sql
```

The **`multiport2.sql`** contains the following SQL:

```
SELECT regexp_substr(to_char(translated_name), '^[a-z]+', 1, 1, 'i')
FROM   oe.product_descriptions
WHERE  language_id = 'PT'
AND    translated_name like 'G%' ;
```



144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
9 rows selected.

SQL> @multiport2

REGEXP_SUBSTR(TO_CHAR(TRANSLATED_NAME),'^[A-Z]+',1,1,'I')
--------------------------------------------------------------------------------
GP
GP
GP
Gest
Gest
Gest
Gr
Gr
Gr

9 rows selected.

SQL>
```

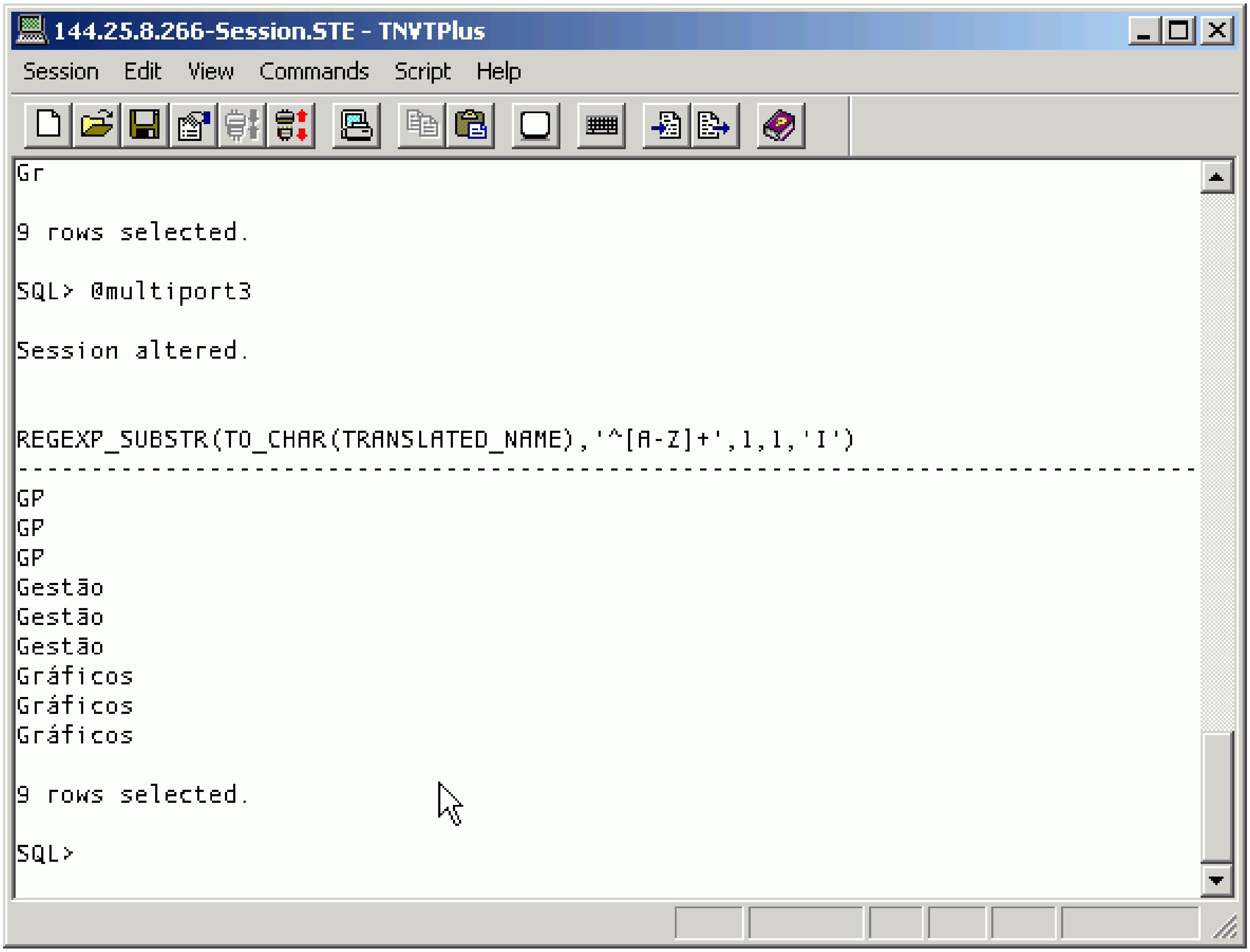
☐ **3.** The results are still incomplete as the returned strings are being trimmed as soon as a non-English character is encountered. This is because the range `[a-z]` is sensitive to `NLS_LANGUAGE` . Hence, you need to set the `NLS_LANGUAGE` parameter appropriately to return the complete results. Execute the following query:

```
@multiport3.sql
```

The **`multiport3.sql`** contains the following SQL:

```
ALTER SESSION SET NLS_LANGUAGE=PORTUGUESE
;
SELECT regexp_substr(to_char(translated_name), '^[a-z]+', 1, 1, 'i')
```

```
FROM    oe.product_descriptions
WHERE   language_id = 'PT'
AND     translated_name like 'G%' ;
```



```
Gr

9 rows selected.

SQL> @multiport3

Session altered.


REGEXP_SUBSTR(TO_CHAR(TRANSLATED_NAME),'^[A-Z]+',1,1,'I')
--------------------------------------------------------------------------------
GP
GP
GP
Gestão
Gestão
Gestão
Gráficos
Gráficos
Gráficos

9 rows selected.

SQL>
```

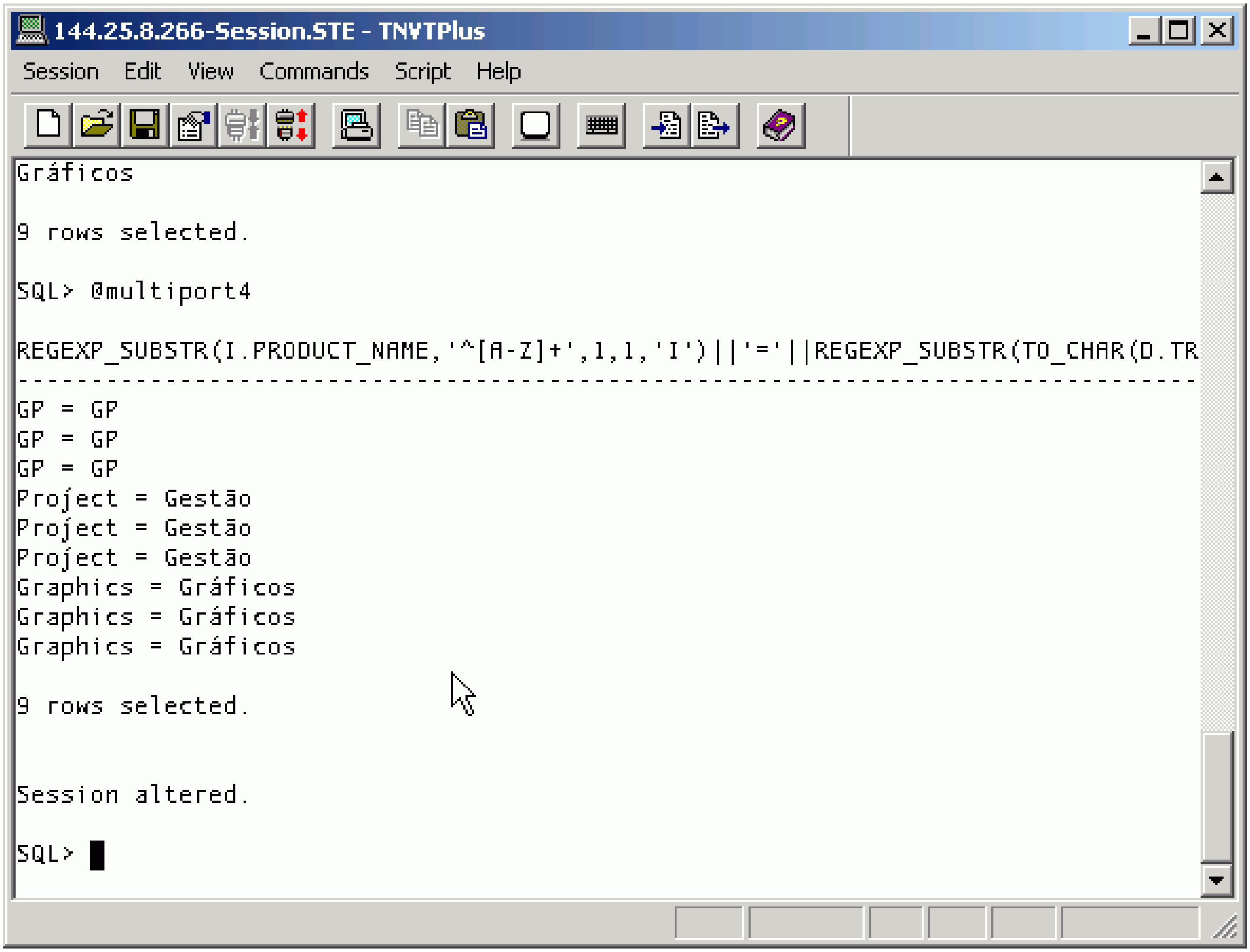
☐ **4.** The final step is to see the results in both English as well as Portuguese to ensure that the translation has taken place. Execute the following script:

```
@multiport4.sql
```

The **`multiport4.sql`** contains the following SQL:

SELECT REGEXP_SUBSTR(i.product_name, '^[a-z]+', 1, 1, 'i') || ' = '

```
        || regexp_substr(to_char(d.translated_name), '^[a-z]+', 1, 1, 'i')
FROM   oe.product_descriptions d, oe.product_information i
WHERE  d.language_id = 'PT'
AND    d.translated_name like 'G%'
AND    i.product_id = d.product_id ;

ALTER SESSION SET NLS_LANGUAGE=AMERICAN;
```



```
Gráficos

9 rows selected.

SQL> @multiport4

REGEXP_SUBSTR(I.PRODUCT_NAME,'^[A-Z]+',1,1,'I')||'='||REGEXP_SUBSTR(TO_CHAR(D.TR
--------------------------------------------------------------------------------
GP = GP
GP = GP
GP = GP
Project = Gestão
Project = Gestão
Project = Gestão
Graphics = Gráficos
Graphics = Gráficos
Graphics = Gráficos

9 rows selected.


Session altered.

SQL>
```

Regular Expressions and Check Constraints

Back to List of Topics

Regular expressions can also be used in check constraints. Perform the following:

☐ **1.** You will add a check constraint on the `CUST_EMAIL` column of the `CUSTOMERS` table. This will ensure that only strings containing an "@" symbol will be accepted. Execute the following script:

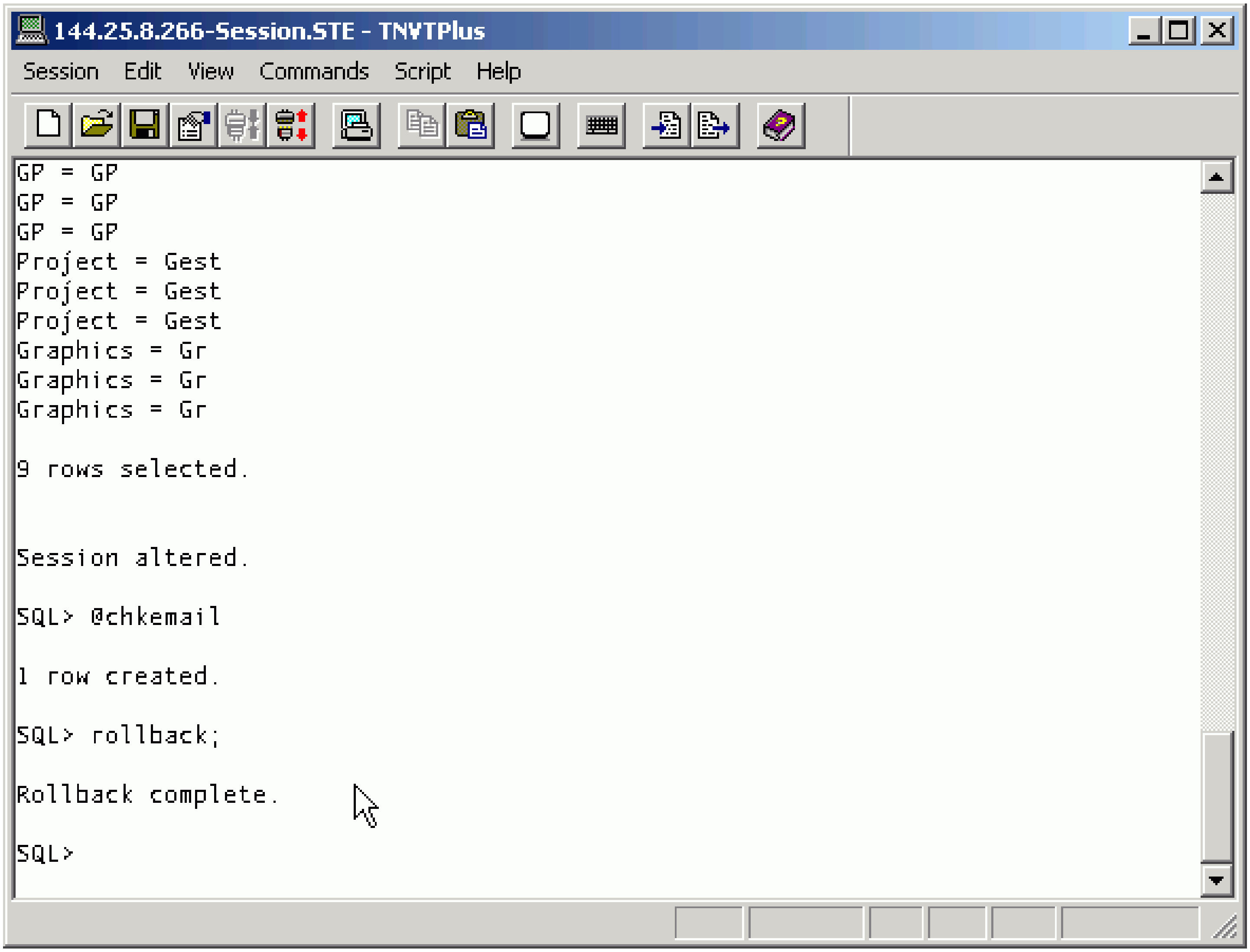
```
@chkemail.sql
```

The **`chkemail.sql`** contains the following SQL:

```
INSERT INTO customers VALUES
           (9999,'Christian','Patel',
            cust_address_typ ('1003 Canyon Road','87501',



'Santa Fe','NM','US'),
            phone_list_typ ('+1 505 243 4144'),'us','AMERICA','100',
            'ChrisP+creme.com', 149, null, null, null, null, null) ;
```

Because there was no validation being performed, an email address not containing an `' @'` was accepted. Perform a rollback before commencing the next step.

```
ROLLBACK ;
```


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

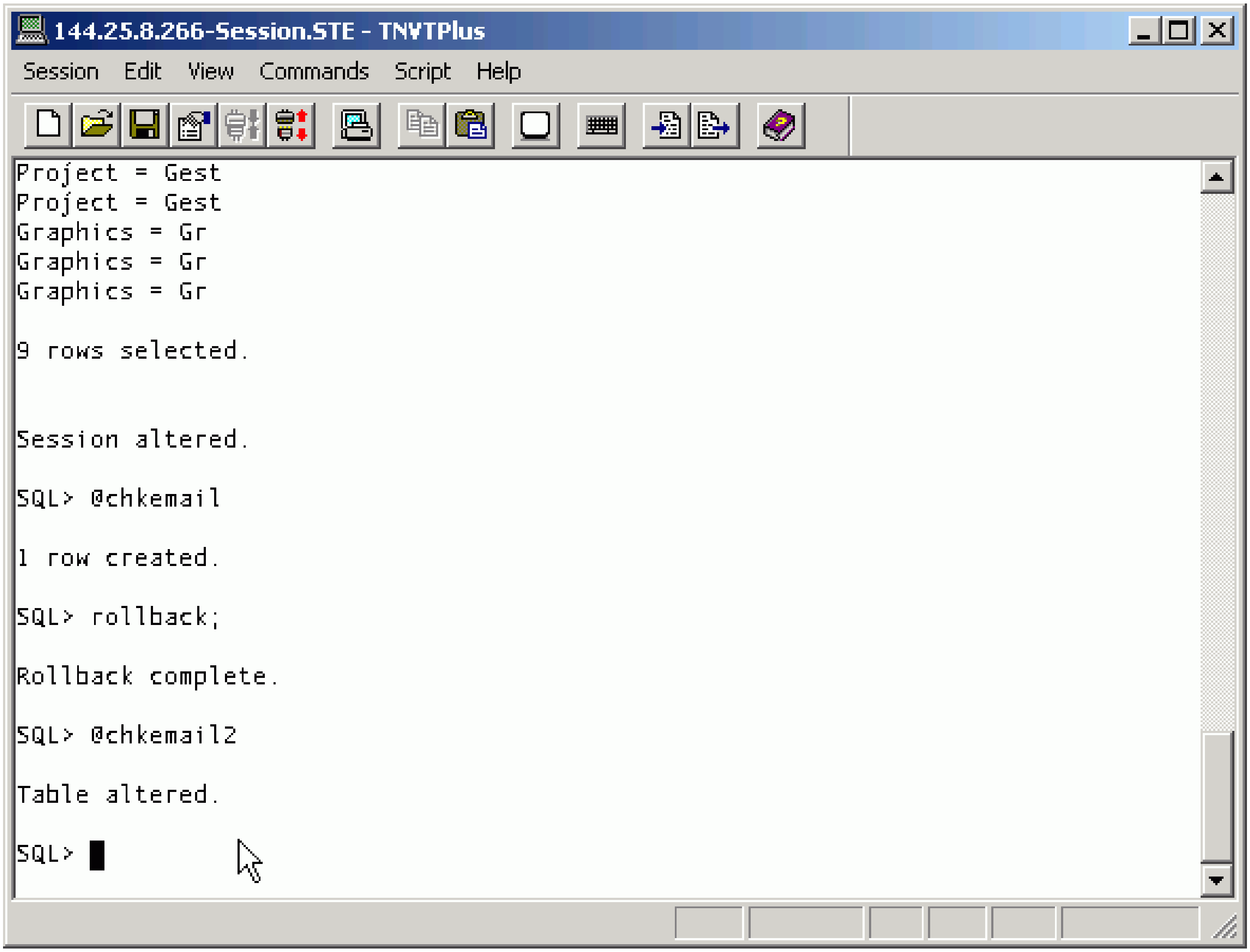
```
GP = GP
GP = GP
GP = GP
Project = Gest
Project = Gest
Project = Gest
Graphics = Gr
Graphics = Gr
Graphics = Gr

9 rows selected.


Session altered.

SQL> @chkemail

1 row created.

SQL> rollback;

Rollback complete.

SQL>
```

☐ **2.** Implement the constraint by executing the following script:

```
@chkemail2.sql
```

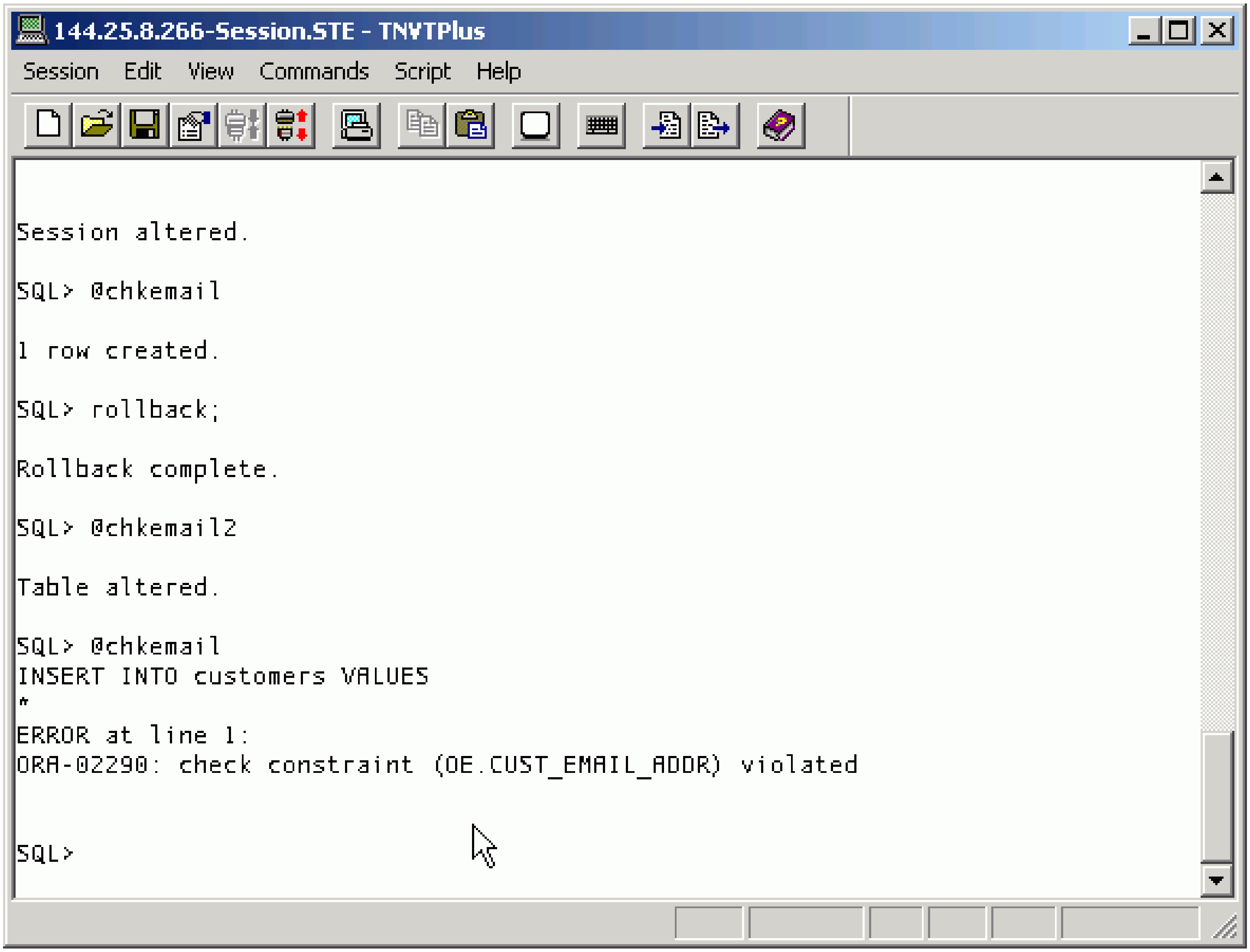
The **`chkemail2.sql`** contains the following SQL:

```
ALTER TABLE customers
  ADD CONSTRAINT cust_email_addr
      CHECK(REGEXP_LIKE(cust_email,'@'))NOVALIDATE ;
```


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
Project = Gest
Project = Gest
Graphics = Gr
Graphics = Gr
Graphics = Gr

9 rows selected.


Session altered.

SQL> @chkemail

1 row created.

SQL> rollback;

Rollback complete.

SQL> @chkemail2

Table altered.

SQL>
```

☐ **3.** Test the constraint by executing **`@chkemail.sql`** again.


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
Session altered.

SQL> @chkemail

1 row created.

SQL> rollback;

Rollback complete.

SQL> @chkemail2

Table altered.

SQL> @chkemail
INSERT INTO customers VALUES
*
ERROR at line 1:
ORA-02290: check constraint (OE.CUST_EMAIL_ADDR) violated


SQL>
```

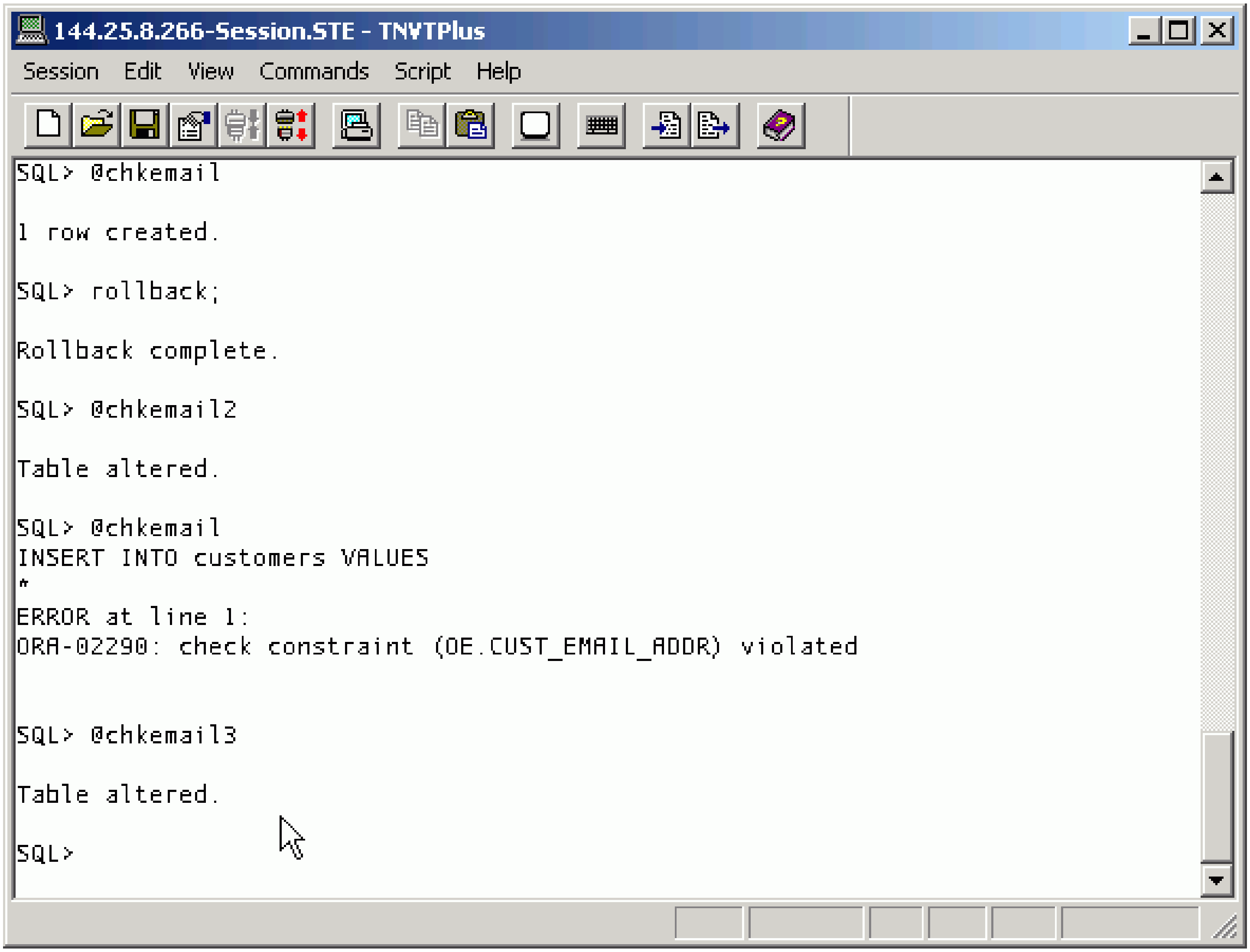
The check constraint is violated because the email address does not contain the required symbol. The clause ' `NOVALIDATE` ' ensures that existing data is not checked.

☐ **4.** Remove the constraint by executing the following script:

```
@chkemail3.sql
```

The **`chkemail3.sql`** contains the following SQL:

```
ALTER TABLE customers DROP CONSTRAINT cust_email_addr ;
```


144.25.8.266-Session.STE - TNVTPlus
Session Edit View Commands Script Help

```
SQL> @chkemail

1 row created.

SQL> rollback;

Rollback complete.

SQL> @chkemail2

Table altered.

SQL> @chkemail
INSERT INTO customers VALUES
*
ERROR at line 1:
ORA-02290: check constraint (OE.CUST_EMAIL_ADDR) violated


SQL> @chkemail3

Table altered.

SQL>
```