

Oracle Forms Developer

Procedure Builder リファレンス

リリース 6*i*

2000 年 4 月

部品番号 : J01125-01

Oracle Forms Developer Procedure Builder リファレンス リリース 6i

部品番号: J01125-01

原本名: Oracle Forms Developer: Procedure Builder Reference, Release 6i

原本部品番号: A73076-01

Copyright © Oracle Corporation 1997, 1999. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム（ソフトウェアおよびドキュメントを含む）の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、Oracle Corporation（米国オラクル）または日本オラクル株式会社（日本オラクル）を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation（米国オラクル）およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

目次

はじめに	vii
前提条件	viii
表記規則	viii
関連資料	viii
Procedure Builderリファレンス	1
インタプリタ・コマンドの使用	2
コマンドのアルファベット順リスト	4
バインド変数コマンド	7
データベース・コマンド	7
デバッグ・アクション・コマンド	7
デバッグ・コマンド	8
ライブラリ・コマンド	8
ロード・パス・コマンド	9
ロギング・コマンド	9
プログラム単位コマンド	9
セッション・コマンド	10
インタプリタ・コマンド	10
バインド変数コマンド	11
CREATE (バインド変数) コマンド	12
DELETE (バインド変数) コマンド	13
データベース・コマンド	15
CONNECTコマンド	16
DESCRIBE (表およびビュー) コマンド	17
DISCONNECTコマンド	17
GRANTコマンド (データベース・コマンド)	18
REVOKEコマンド (データベース・コマンド)	18
STOREコマンド	19
デバッグ・アクション・コマンド	21
BREAKコマンド	22
DELETE (デバッグ・アクション) コマンド	24
DESCRIBE (デバッグ・アクション) コマンド	25
DISABLE (デバッグ・アクション) コマンド	26

ENABLE (デバッグ・アクション) コマンド	26
LIST (デバッグ・アクション) コマンド	27
SHOW (デバッグ・アクション) コマンド	28
TRIGGERコマンド	28
デバック・コマンド	33
DESCRIBE (ローカル) コマンド	34
GOコマンド	34
RESETコマンド	35
SET (有効範囲) コマンド	35
SHOW (コール・スタック) コマンド	37
STEPコマンド	37
ライブラリ・コマンド	39
ATTACHコマンド	40
CLOSEコマンド	41
COMPILE (ライブラリ) コマンド	41
CREATE (ライブラリ) コマンド	42
DELETE (ライブラリ) コマンド	43
DELETE (ライブラリ・プログラム単位) コマンド	44
DESCRIBE (ライブラリ) コマンド	45
DETACHコマンド	45
EXPORT (ライブラリ) コマンド	46
GENERATEコマンド	47
GRANTコマンド (ライブラリ・コマンド)	48
INSERT (ライブラリ・プログラム単位) コマンド	48
LOAD (ライブラリ・プログラム単位) コマンド	50
OPENコマンド	51
RENAME (ライブラリ) コマンド	52
REVERTコマンド	52
REVOKEコマンド (ライブラリ・コマンド)	53
SAVEコマンド	53
SHOW (ライブラリ) コマンド	55
ロード・パス・コマンド	57
DELETE (ロード・パス) コマンド	58
DESCRIBE (ロード・パス) コマンド	58
INSERT (ロード・パス) コマンド	58
ロギング・コマンド	61
DISABLE (ロギング) コマンド	62
ENABLE (ロギング) コマンド	62

LOGコマンド	63
プログラム単位コマンド	65
COMPILE (プログラム単位) コマンド	66
DELETE (プログラム単位) コマンド	66
DESCRIBE (プログラム単位) コマンド	67
DISABLE (コンパイラ・オプション) コマンド	68
ENABLE (コンパイラ・オプション) コマンド	69
EXECUTEコマンド	70
EXPORT (プログラム単位) コマンド	70
EXPORT (ストアド・プログラム単位) コマンド	72
LIST (プログラム単位) コマンド	73
LOAD (プログラム単位) コマンド	75
LOAD (ストアド・プログラム単位) コマンド	76
SHOW (ローカル) コマンド	77
SHOW (プログラム単位) コマンド	78
セッション・コマンド	79
DESCRIBE (バージョン) コマンド	80
HELPコマンド	80
INTERPRETコマンド	81
QUITコマンド	82
索引	83

はじめに

Oracle Forms Developer Procedure Builderリファレンス、リリース6iによろこそ。

このリファレンス・ガイドでは、Oracle Forms DeveloperのProcedure Builderを効果的に利用できるようにするための情報と、コマンドに関する詳細な情報が説明されています。

ここでは、このガイドの構成を説明し、Procedure Builderを使用する際に参考になるその他の情報源を紹介します。

前提条件

まず、ご使用のコンピュータおよびそのオペレーティング・システムについて精通している必要があります。たとえば、ファイルの削除およびコピーのコマンドの知識があり、検索パス、サブディレクトリおよびパス名を概念を理解していなければなりません。詳細は、各オペレーティング・システムの製品マニュアルを参照してください。

アプリケーション・ウィンドウの要素などのMicrosoft Windowsの基本要素も理解している必要があります。エクスプローラ、タスクバー、タスクマネージャ、またはレジストリなどのプログラムに精通している必要があります。

表記規則

このマニュアルでは、次のような表記上の規則を使用しています。

規則	意味
固定幅フォント	固定幅フォントのテキストは、表示されたとおりに入力するコマンドを示します。PCに入力するテキストでは、特に断りのない限り大文字と小文字を区別しません。 コマンドでは、大カッコと縦線以外の句読点は表示されているとおり正確に入力する必要があります。
小文字	コマンド文の小文字は変数を表します。適切な値に置き換えてください。
大文字	テキスト内の大文字は、コマンド名、SQL予約語、キーワードを表します。
ゴシック・テキスト	メニュー選択項目やボタンなど、ユーザー・インタフェース項目を示すには、ゴシック・テキストが使用されます。
C>	C>はDOSプロンプトを表します。実際とは異なる場合があります。

関連資料

次のOracleマニュアルを参照することもできます。

タイトル	部品番号
『Oracle Forms Developer and Oracle Reports Developer アプリケーション作成ガイド リリース6i』	J00449-01

Procedure Builder リファレンス

インタプリタ・コマンドの使用

コマンドは、次の基本構文を使用します。

```
.command-name [option...]
```

つまりコマンドは、ピリオド(.)、コマンド名、0個以上のキーワードおよびキーワード値の引数から構成され、この順で並びます。

コマンド・オプションの形式は、一般に次のようになります。

```
keyword
```

または

```
keyword value(s)
```

このようにオプションは、キーワード単独か、またはキーワードと1つ以上の値引数で構成されています。コマンド名およびキーワード、引数値は、スペースで区切られます。コマンド名およびキーワード、引数値は、大/小文字の区別はありません。

たとえば、次のDESCRIBEコマンドを起動すると、Procedure Builderコマンド構文の基本要素が示されます。

```
.DESCRIBE PROCEDURE proc1 BODY
```

コマンド名DESCRIBEの後に、キーワードのPROCEDUREとBODYが続きます。PROCEDUREには、1つの引数値`proc1`が指定されていますが、BODYキーワードには引数値はありません。

多値引数

キーワードの引数に複数の値が指定される場合があります。その場合、個々の値は次のようにカンマで区切られます。

```
value, value...
```

カンマと次の値の間には、スペースを置くこともできます。

前述の構文に従って、複数の値を取り得るキーワードの引数は、次のように表示されます。

```
name[, name...]
```

たとえば、LOADコマンドには、次のような構文があります。

```
.LOAD FILE name[, name...]
```

したがって、ファイル引数を次のように1つのみ指定することができます。

```
.LOAD FILE file1
```

または、次のように、複数の値を指定することもできます。

```
.LOAD FILE file1, file2, file3
```

キーワードの位置について

構文説明で明示的に指定されていない場合、キーワードは任意の順序で指定できます。たとえば、コマンド

```
.DESCRIBE PROCEDURE proc1 BODY
```

は、次のように入力することもできます。

```
.DESCRIBE BODY PROCEDURE proc1
```

マルチライン・コマンド

通常、コマンドは、復帰改行文字かキャリッジ・リターンによって終了します。しかし、複数の行にまたがるコマンドを作成することも考えられます。このような場合、各行の最後の文字が次に続くことを示す継続文字(デフォルトでは`¥`)を挿入することで、複数行のコマンドを作成できます。たとえば、次の継続文字は、LOADコマンドに対する各ファイル名の引数値を、別々の行に分けて指定するため使用されています。

```
.LOAD FILE long_file_name_number_one, ¥  
long_file_name_number_two, ¥  
long_file_name_number_three
```

引数値の引用符使用

非数値コマンドの引数値は、任意に二重引用符で囲むことができます。二重引用符はデリミタの役割を果たしているのみで、引数値の一部とは見なされません。これは特に、スペースまたはカンマ、ワイルドカード文字を含む引数値を指定する場合に便利です。たとえば、固有のオペレーティング・システムで二重引用符がサポートされている場合、スペースを含むファイル名はロード・コマンドで次のように指定されます。

```
.LOAD FILE "my file"
```

二重引用符は、さらに別の二重引用符で囲むことによって、引数値の一部として組み込むことができます。たとえば、コマンド

```
.LOAD FILE "\"quoted file\""
```

は、2つの二重引用符(最初と最後に1つずつ)が付いた名前のファイルをロードします。

キーワード短縮

コマンド・キーワードは、同じコマンドで受け入れられる他のすべてのキーワードと区別するのに必要な文字を入力することによって、短縮することができます。

コマンド名は短縮できません。これは、PL/SQLの名前領域と競合することを最小限に抑え、コマンドとPL/SQLコードの区別の混乱を避けるためです。

PL/SQLコードの入力

インタプリタは、コマンドの他に、PL/SQL構成体（文、ブロック、プロシージャ定義など）およびSQL文も受け入れ評価します。Procedure Builderでは、有効なコマンド名以外のものから始まっている行を、PL/SQL文、ブロック、プログラム単位またはSQL文の始めと解釈します。

コマンドが1行のみ（継続文字が使用されていない場合）なのに対し、PL/SQLまたはSQL文には何行でも使用することができ、継続文字は不要で、使用することもできません。

必要に応じて、PL/SQL構成体は、ブロック・デリミタのBEGINとENDで囲むことによって、コマンドと常に区別できます。

表記規則

次の表は、この項で使用されるコマンド構文の表記規則の説明です。

機能	例	説明
大文字	BREAK	コマンドまたはキーワードの名前。大文字で入力する必要はありません。
小文字イタリック	<i>数値</i>	キーワード値。適切な値を代入します。
垂直バー		オプションまたは必須の構文要素の選択肢を区切ります。
中カッコ	{STACK SCOPE}	必須項目の選択肢。 で分離された項目の1つを入力します。中カッコや垂直バーは入力しないでください。
大カッコ	[前 後]	1つ以上のオプション項目。垂直バーで分離された2つの項目が表示される場合は、それらの項目の1つを入力します。大カッコまたは垂直バーは入力しないでください。
下線	[前 後]	デフォルト値。何も入力しない場合、この値が使用されます。

他の句読点（カンマなど）は、コマンド構文どおりに入力します。

コマンドのアルファベット順リスト

ATTACH

BREAK

CLOSE

COMPILE（ライブラリ）

COMPILE (プログラム単位)
CONNECT
CREATE (バインド変数)
CREATE (ライブラリ)
DELETE (バインド変数)
DELETE (デバッグ・アクション)
DELETE (ライブラリ)
DELETE (ライブラリ・プログラム単位)
DELETE (ロード・パス)
DELETE (プログラム単位)
DESCRIBE (デバッグ・アクション)
DESCRIBE (ロード・パス)
DESCRIBE (ローカル)
DESCRIBE (ライブラリ)
DESCRIBE (プログラム単位)
DESCRIBE (表およびビュー)
DESCRIBE (バージョン)
DETACH
DISABLE (コンパイラ・オプション)
DISABLE (デバッグ・アクション)
DISABLE (ロギング)
DISCONNECT
ENABLE (コンパイラ・オプション)
ENABLE (デバッグ・アクション)
ENABLE (ロギング)

EXECUTE
EXPORT
GENERATE
GO
GRANT
HELP
INSERT (ライブラリ・プログラム単位)
INSERT (ロード・パス)
INTERPRET
LIST (デバッグ・アクション)
LIST (プログラム単位)
LOAD (ライブラリ・プログラム単位)
LOAD (プログラム単位)
LOAD (ストアド・プログラム単位)
LOG
OPEN
QUIT
RENAME
RESET
REVERT
REVOKE
SAVE
SET (有効範囲)
SHOW (コール・スタック)
SHOW (デバッグ・アクション)
SHOW (ライブラリ)

SHOW (ローカル)
SHOW (プログラム単位)
STEP
STORE
TRIGGER

バインド変数コマンド

CREATE (バインド変数)
DELETE (バインド変数)

データベース・コマンド

CONNECT
DESCRIBE (表およびビュー)
DISCONNECT
GRANT
REVOKE
STORE

デバッグ・アクション・コマンド

BREAK
DELETE (デバッグ・アクション)
DESCRIBE (デバッグ・アクション)
DISABLE (デバッグ・アクション)
ENABLE (デバッグ・アクション)
LIST (デバッグ・アクション)
SHOW (デバッグ・アクション)

TRIGGER

デバッグ・コマンド

DESCRIBE (ローカル)

GO

RESET

SET (有効範囲)

SHOW (コール・スタック)

STEP

ライブラリ・コマンド

ATTACH

CLOSE

COMPILE (ライブラリ)

CREATE (ライブラリ)

DELETE (ライブラリ)

DELETE (ライブラリ・プログラム単位)

DESCRIBE (ライブラリ)

DETACH

EXPORT (ライブラリ)

GENERATE

GRANT

INSERT (ライブラリ・プログラム単位)

LOAD (ライブラリ・プログラム単位)

OPEN

RENAME

REVERT

REVOKE

SAVE

SHOW (ライブラリ)

ロード・パス・コマンド

DELETE (ロード・パス)

DESCRIBE (ロード・パス)

INSERT (ロード・パス)

ロギング・コマンド

DISABLE (ロギング)

ENABLE (ロギング)

LOG

プログラム単位コマンド

COMPILE (プログラム単位)

DELETE (プログラム単位)

DESCRIBE (プログラム単位)

DISABLE (コンパイラ・オプション)

ENABLE (コンパイラ・オプション)

EXECUTE

EXPORT (プログラム単位)

EXPORT (ストアド・プログラム単位)

LIST (プログラム単位)

LOAD (プログラム単位)

LOAD (ストアド・プログラム単位)

SHOW (ローカル)

SHOW (プログラム単位)

セッション・コマンド

DESCRIBE (バージョン)

HELP

INTERPRET

QUIT

インタプリタ・コマンド

バインド変数コマンド

データベース・コマンド

デバッグ・アクション・コマンド

デバッグ・コマンド

ライブラリ・コマンド

ロード・パス・コマンド

ロギング・コマンド

プログラム単位コマンド

セッション・コマンド

バインド変数コマンド

CREATE (バインド変数) コマンド

説明

バインド変数を作成します。このコマンドは、Procedure Builderがスタンドアロン・セッションとして起動される場合にのみ有効です。

構文

```
CREATE CHAR var_name [LENGTH number]
CREATE NUMBER var_name [PRECISION number] [SCALE number]
CREATE RAW var_name [LENGTH number]
CREATE DATE var_name
```

キーワードおよび値

CHAR <i>var_name</i>	CHARデータ型のバインド変数 <i>var_name</i> を指定します。
LENGTH <i>number</i>	CHARバインド変数の長さを任意に指定します。
DATE <i>var_name</i>	DATEデータ型のバインド変数 <i>var_name</i> を指定します。
NUMBER <i>var_name</i>	NUMBERデータ型のバインド変数 <i>var_name</i> を指定します。
PRECISION <i>number</i>	変数の数値の最大桁数を指定します。
SCALE <i>number</i>	四捨五入する位を指定します。
RAW <i>var_name</i>	RAWデータ型のバインド変数 <i>var_name</i> を指定します。

コメント

CHARデータ型にLENGTHを指定しない場合、CHARデータ型のLENGTH属性は、デフォルトで1バイトになります。LENGTHの最大値は32767バイトです。

PRECISIONの最大値は38字です。SCALEの範囲は、-84 ~ 127です。SCALEの値を指定しない場合、デフォルトは0(ゼロ)です。これは、数値が最も近い整数に丸められることを意味します。

データ型とそれらの属性の詳細は、『PL/SQLユーザーズ・ガイドおよびリファレンス』を参照してください。

CREATE (バインド変数) コマンドの例

次のコマンドは、近似の百の位に四捨五入するデータ型NUMBERのバインド変数*x*を作成します。

```
.CREATE NUMBER x SCALE 2
```

DELETE (バインド変数) コマンド

説明

1つ以上のバインド変数を削除します。このコマンドは、Procedure Builderがスタンドアロン・セッションとして起動される場合にのみ有効です。

構文

```
DELETE BINDVAR name [, name...]  
DELETE CHAR name [, name...]  
DELETE DATE name [, name...]  
DELETE NUMBER name [, name...]
```

キーワードおよび値

BINDVAR <i>name</i>	バインド変数、または任意のデータ型の一連のバインド変数を指定します。
CHAR <i>name</i>	バインド変数、またはCHARデータ型の一連のバインド変数を指定します。
DATE <i>name</i>	バインド変数、またはDATEデータ型の一連のバインド変数を指定します。
NUMBER <i>name</i>	バインド変数、またはNUMBERデータ型の一連のバインド変数を指定します。

DELETE (バインド変数) コマンドの例

次のコマンドは、データ型CHARのバインド変数*y*を削除します。

```
.DELETE CHAR y
```

次のコマンドは、異なるデータ型の一連のバインド変数 (*x*, *y*, および *z*) を削除します。

```
.DELETE BINDVAR x, y, z
```


データベース・コマンド

CONNECTコマンド

説明

データベースの接続を確立します。このコマンドは、Procedure Builderがスタンドアロン・セッションとして起動される場合にのみ有効です。

構文

```
CONNECT DB [username/password@ |
           network_device: |
           datasource_node: |
           datasource_name]
[SILENT]
```

キーワードおよび値

<i>username/password@</i>	接続するデータソースの有効なユーザー名およびパスワードを示します。 '@'記号は、残りのデータベース位置指定子に先行する必要があります。
<i>network_device:</i>	リモート・データベースへの接続に使用されるネットワーク・デバイス・ドライバを指定します。
<i>datasource_node:</i>	接続するリモート・データソースのネットワーク・ノードを指定します。
<i>datasource_name</i>	接続するリモートまたはローカル・データベースのIDを指定します。
SILENT	インタプリタが発行する状態メッセージを、任意に非表示にします。

注意: ODBCデータソースに接続する場合は、次の構文を使用します。

```
username/password@ODBC:datasource[:dbname]
```

dbnameを指定しないと、現行のODBC接続用データベースが使用されます。

CONNECTコマンドの例

次のコマンドは、TCP/IPデバイス・ドライバを使用し"boston"ネットワーク・ノード上のリモート"inventory"データベースに接続します。

```
.CONNECT DB scott/tiger@t:boston:inventory
```

"inventory"データベースがローカル・データベースの場合、次のコマンドで接続できます。

```
.CONNECT DB scott/tiger@inventory
```

DESCRIBE (表およびビュー) コマンド

説明

データベースの表およびビューの詳細情報を表示します。

構文

```
DESCRIBE TABLE name
```

```
DESCRIBE VIEW name
```

キーワードおよび値

TABLE <i>name</i>	現在のデータベースの表を指定します。
VIEW <i>name</i>	現在のデータベースのビューを指定します。

コメント

表示される表やビューの情報には、列とそのタイプが含まれます。

DESCRIBE (表およびビュー) コマンドの例

次のコマンドは、EMP表の情報を表示します。

```
.DESCRIBE TABLE emp
```

次のコマンドは、ASSOCIATEという名前のビューの情報を表示します。

```
.DESC V associate
```

DISCONNECTコマンド

説明

現在接続しているデータベースから切断します。このコマンドは、Procedure Builderがスタンドアロン・セッションとして起動される場合にのみ有効です。

構文

```
DISCONNECT
```

GRANTコマンド (データベース・コマンド)

説明

ユーザーに、データベースに格納されたライブラリへのアクセス権限を付与します。

構文

```
GRANT LIBRARY name USER name
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリを指定します。
USER <i>name</i>	ユーザー名を指定します。

コメント

任意のユーザー名を1つ、またはPUBLIC (すべてのユーザー) を指定できます。

GRANTコマンドの例 (データベース・コマンド)

次のコマンドは、ユーザーSCOTTにデータベース・ライブラリ*lib1*へのアクセス権限を付与します。

```
.GRANT LIB lib1 USER scott
```

REVOKEコマンド (データベース・コマンド)

説明

データベースに格納されているライブラリに対するユーザーのアクセス権限を取り消します。

構文

```
REVOKE LIBRARY name USER name
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリを指定します。
USER <i>name</i>	ユーザーを指定します。

コメント

任意のユーザー名を1つ、またはPUBLIC (すべてのユーザー) を指定できます。

REVOKEコマンドの例 (データベース・コマンド)

次のコマンドは、データベース・ライブラリ *lib1* に対するユーザー *SCOTT* のアクセス権限を取り消します。

```
.REVOKE LIB lib1 USER scott
```

STOREコマンド

説明

データベースに1つ以上のプログラム単位を格納します。

構文

```
STORE PROGRAMUNIT name [, name...]
    FILE [directory] name [extension]
    [SPECIFICATION | BODY]
    [NOWARN]
STORE PACKAGE name [, name...]
    [OWNER name]
    [SPECIFICATION | BODY]
STORE SUBPROGRAM name [, name...]
    [OWNER name]
    [SPECIFICATION | BODY]
STORE PROCEDURE name [, name...]
    [OWNER name]
    [SPECIFICATION | BODY]
STORE FUNCTION name [, name...]
    [OWNER name]
    [SPECIFICATION | BODY]
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
OWNER <i>name</i>	ストアド・プログラム単位の所有者を指定します。
SPECIFICATIONまたはBODY	パッケージの仕様部または本体をデータベースに格納するかどうかを、それぞれ示します。

コメント

OWNERを指定しない場合、Procedure Builderでは現在接続されているユーザーが、ストアド・プログラムの所有者として割り当てられます。

SPECIFICATIONまたはBODYのどちらも指定しない場合、指定したパッケージに本体と仕様部があれば、両方ともデータベースに格納されます。

STOREコマンドの例

次のコマンドは、現行のデータベースにプロシージャ*proc1*とファンクション*func2*を格納します。

```
.STORE PROGRAMUNIT proc1, func2
```

次のコマンドは、パッケージ*pack1*の仕様部および本体を格納し、所有者をSCOTTに指定します。

```
.STORE PACK pack1 OWNER scott
```

デバック・アクション・コマンド

BREAKコマンド

説明

プログラム単位内の指定したソース行に、ブレークポイントを設定します。

構文

```
BREAK {[USER schema] PROGRAMUNIT name | PROGRAMUNIT [schema.]name}
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK {USER schema PACKAGE name | PACKAGE schema.name}
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK {USER schema SUBPROGRAM name | SUBPROGRAM schema.name}
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK {USER schema PROCEDURE name | PROCEDURE schema.name}
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK {USER schema FUNCTION name | FUNCTION schema.name}
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK ACTION number
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK BREAKPOINT number
    [LINE number]
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK .
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
BREAK PC
    [ENABLED | DISABLED]
    [TRIGGER plsql-block]
```

BREAK SCOPE
 [ENABLED | DISABLED]
 [TRIGGER *plsql-block*]

キーワードおよび値

USER <i>schema</i>	ストアド・プログラム単位があるデータベース内のスキーマ名を指定します。
PROGRAMUNIT <i>name</i>	プログラム単位本体を指定します。
PACKAGE <i>name</i>	パッケージ本体を指定します。
SUBPROGRAM <i>name</i>	サブプログラム本体を指定します。
PROCEDURE <i>name</i>	プロシージャ本体を指定します。
FUNCTION <i>name</i>	ファンクション本体を指定します。
ACTION <i>number</i>	デバッグ・アクション（ブレーク・ポイントまたはトリガー）を指定します。
BREAKPOINT <i>number</i>	ブレーク・ポイントを指定します。
.	現行のソース位置を指定する。これがデフォルトです。
PC	現行の実行位置を指定します。
SCOPE	現行の有効範囲位置を指定します。
LINE <i>number</i>	プログラム単位内のブレークポイントを設定する行を指定します。
ENABLEDまたはDISABLED	ブレークポイントを最初に使用可能または使用不可のどちらにするかを指定します。デフォルトはENABLEDです。
TRIGGER <i>pl/sql-block</i>	ブレークポイントのPL/SQLトリガーを定義します。トリガーは、ブレークポイントに達するたびに起動します。

注意: TRIGGERキーワードを指定する場合は、コマンド・オプションの最後に指定してください。

コメント

BREAKは、実行可能なソース行でのみ機能します。

トリガー・ブロックは、複数の入力行にまたがる場合があります。インタプリタの別の場所にPL/SQL構成体を入力する場合と同様、トリガー本体を入力するのに行継続文字は必要ありません（これらを使用することもできません）。

プログラムに条件付きで割り込みたい場合、DEBUG.BREAK例外と一緒にTRIGGERコマンドを使用してください。

PL/SQLの実行中に、ブレークポイントを指定した文に達した場合、Procedure Builderは、文の実行の直前に実行を停止し、制御をインタプリタに渡します。この時点で、様々なProcedure Builderファンクションを使用してプログラム状態を検査したり、変更したりすることもできます。

条件が満たされれば、GOまたはSTEPコマンドを使用して実行を再開できます。RESETコマンドを使用して実行を中止することもできます。

BREAKコマンドの例

次のコマンドは、現行のソースの位置にブレークポイントを設定します。

```
.BREAK .
```

次のコマンドは、*my_proc*という名前のプロシージャの2行目に、ブレークポイントを設定します。

```
.BREAK PROCEDURE my_proc LINE 2
```

次のコマンドは、*my_proc*の10行目にブレークポイントを設定し、ブレークポイントになると、必ずすべてのローカル変数とその値が表示されるようにします。

```
.BREAK PROC my_proc LINE 10 TRIGGER  
  debug.interpret('.SHOW LOCALS')
```

次のコマンドは、デバッグ・アクション4を含むプログラム単位の12行目に、ブレークポイントを設定します。

```
.BREAK ACTION 4 LINE 12
```

次のコマンドは、ユーザー*scott*が所有するスキーマから、サーバー側のプログラム単位*my_proc*内の現行のソースの位置に、ブレークポイントを設定します。

```
.BREAK USER scott PROC my_proc
```

または

```
.BREAK PROC scott.my_proc
```

DELETE (デバッグ・アクション) コマンド

説明

1つ以上のデバッグ・アクションを削除します。

構文

```
DELETE ACTION number [, number...]  
DELETE BREAKPOINT number [, number...]  
DELETE TRIGGER number [, number...]
```

キーワードおよび値

ACTION <i>number</i>	1つ以上のデバッグ・アクション(ブレークポイントまたはトリガー)を番号により指定する。
BREAKPOINT <i>number</i>	1つ以上のブレークポイントを番号で指定します。
TRIGGER <i>number</i>	1つ以上のデバッグ・トリガーを番号で指定します。

コメント

このコマンドは、1つ以上のデバッグ・アクションを、永続的に削除します。デバッグ・アクションを一時的に削除する場合は、かわりにDISABLEコマンドを使用します。

DELETE (デバッグ・アクション) コマンドの例

次のコマンドは、デバッグ・アクション2および3を削除します。

```
.DELETE ACTION 2,3
```

DESCRIBE (デバッグ・アクション) コマンド

説明

指定するデバッグ・アクションの詳細情報を表示します。

構文

```
DESCRIBE ACTION number  
DESCRIBE BREAKPOINT number  
DESCRIBE TRIGGER number
```

キーワードおよび値

ACTION <i>number</i>	デバッグ・アクション (ブレークポイントまたはトリガー) を指定します。
BREAKPOINT <i>number</i>	ブレーク・ポイントを指定します。
TRIGGER <i>number</i>	トリガーを指定します。

コメント

表示されるデバッグ・アクションの情報には、IDおよび対応付けられたソースの位置、使用可能であるかどうかが含まれます。

DESCRIBE (デバッグ・アクション) コマンドの例

次のコマンドは、ブレークポイント2の情報を表示します。

```
.DESCRIBE BREAK 2
```

次のコマンドは、デバッグ・アクション3の情報を表示します。

```
.DESCRIBE ACTION 3
```

DISABLE (デバッグ・アクション) コマンド

説明

1つ以上のデバッグ・アクションを、一時的に使用不可にします。

構文

```
DISABLE ACTION number [, number...]  
DISABLE BREAKPOINT number [, number...]  
DISABLE TRIGGER number [, number...]
```

キーワードおよび値

ACTION <i>number</i>	1つ以上のデバッグ・アクション (ブレークポイントまたはトリガー) を指定します。
BREAKPOINT <i>number</i>	1つ以上のブレークポイントを指定します。
TRIGGER <i>number</i>	1つ以上のトリガーを指定します。

コメント

DISABLEは、すでに使用不可になっているデバッグ・アクションについては無効です。ENABLE コマンドを使用すると、使用不可のデバッグ・アクションをアクティブに復元できます。

DISABLE (デバッグ・アクション) コマンドの例

次のコマンドは、ブレークポイント2を使用不可にします。

```
.DISABLE BREAK 2
```

次のコマンドは、デバッグ・アクション3を使用不可にします。

```
.DISABLE ACTION 3
```

ENABLE (デバッグ・アクション) コマンド

説明

使用不可のデバッグ・アクションを再びアクティブにします。

構文

```
ENABLE ACTION number [, number...]  
ENABLE BREAKPOINT number [, number...]  
ENABLE TRIGGER number [, number...]
```

キーワードおよび値

ACTION <i>number</i>	デバッグ・アクションを指定します。
BREAKPOINT <i>number</i>	ブレーク・ポイントを指定します。
TRIGGER <i>number</i>	トリガーを指定します。

コメント

ENABLEは、すでにアクティブになっているデバッグ・アクションについては無効です。デバッグ・アクションを一時的に使用不可にするには、DISABLEコマンドを使用します。

ENABLE (デバッグ・アクション) コマンドの例

次のコマンドは、使用不可になっていたブレークポイント2をアクティブにします。

```
.ENABLE BREAK 2
```

次のコマンドは、デバッグ・アクション1をアクティブにします。

```
.ENABLE ACTION 1
```

LIST (デバッグ・アクション) コマンド

説明

指定したデバッグ・アクションが付加されるプログラム単位のソース・テキストを表示します。

構文

```
LIST ACTION number
LIST BREAKPOINT number
LIST TRIGGER number
```

キーワードおよび値

ACTION <i>number</i>	デバッグ・アクション (ブレーク・ポイントまたはトリガー) を指定します。
BREAKPOINT <i>number</i>	ブレーク・ポイントを指定します。
TRIGGER <i>number</i>	トリガーを指定します。

コメント

LISTは、指定したデバッグ・アクションに関係付けられたテキストを、インタプリタのソース・ペインに表示します。指定したデバッグ・アクションの表示される行が、現行のソースの位置になります。

LIST (デバッグ・アクション) コマンドの例

次のコマンドは、ブレークポイント1を表示し、ソースの位置を設定します。

```
.LIST BREAK 1
```

次のコマンドは、デバッグ・アクション3を表示し、現行のソースの位置を設定します。

```
.LIST ACTION 3
```

SHOW (デバッグ・アクション) コマンド

説明

開発セッションで現在定義されているデバッグ・アクションを表示します。

構文

```
SHOW ACTION  
SHOW BREAKPOINTS  
SHOW TRIGGERS
```

キーワードおよび値

ACTION	すべてのデバッグ・アクションを指定します。
BREAKPOINTS	すべてのブレークポイントを指定します。
TRIGGERS	すべてのトリガーを指定します。

SHOW (デバッグ・アクション) コマンドの例

次のコマンドは、現在設定されているすべてのブレークポイントをリストします。

```
.SHOW BREAKPOINTS
```

TRIGGERコマンド

説明

指定したソースの位置に関係付けられたPL/SQLブロックから構成される、デバッグ・トリガーを作成します。

構文

```
TRIGGER { [USER schema] PROGRAMUNIT name | PROGRAMUNIT [schema.] name }  
        [LINE number]  
        [ENABLED | DISABLED]
```

```
[IS plsql-block]  
TRIGGER {USER schema PACKAGE name | PACKAGE schema.name}  
[LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER {USER schema SUBPROGRAM name | SUBPROGRAM schema.name}  
[LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER {USER schema PROCEDURE name | PROCEDURE schema.name}  
[LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER {USER schema FUNCTION name | FUNCTION schema.name}  
[LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER ACTION number [LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER BREAKPOINT number [LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER TRIGGER number [LINE number]  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER .  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER PC  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER SCOPE  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER DEBUG  
[ENABLED | DISABLED]  
[IS plsql-block]  
TRIGGER *  
[ENABLED | DISABLED]  
[IS plsql-block]
```

キーワードおよび値

USER <i>schema</i>	ストアド・プログラム単位があるデータベース内のスキーマ名を指定します。
PROGRAMUNIT <i>name</i>	プログラム単位を指定します。
PACKAGE <i>name</i>	パッケージを指定します。
SUBPROGRAM <i>name</i>	サブプログラムを指定します。
PROCEDURE <i>name</i>	プロシージャを指定します。
FUNCTION <i>name</i>	ファンクションを指定します。
ACTION <i>number</i>	デバック・アクション(ブレーク・ポイントまたはトリガー)を指定します。
BREAKPOINT <i>number</i>	ブレーク・ポイントを指定します。
TRIGGER <i>number</i>	トリガーを指定します。
LINE <i>number</i>	トリガーの設定が必要なプログラム単位の行を指定します。
.	現行のソース位置を指定する。
PC	現行の実行位置を指定します。
SCOPE	現行の有効範囲位置を指定します。
DEBUG	(ブレークポイントやプログラムのステップングなどのためプログラムの実行が中断される場合などに) デバッガへのエントリを指定します。
*	すべてのPL/SQLソース行を指定します。このようにトリガーを*に置くと、指定したブロックは すべての PL/SQLソース行を実行する直前に評価されます。
ENABLEDまたはDISABLED	トリガーを最初に、使用可能または使用不可のどちらにするかを示します。デフォルトはENABLEDです。
IS <i>pl/sql-block</i>	トリガーの本体を定義します。

注意: ISは、コマンド・オプションの最後に指定してください。

コメント

Procedure Builderは、プログラムがTRIGGERコマンドを指定した場所に達する**直前に**トリガーを実行します。トリガー・ブロックは、複数の入力行にまたがる場合があります。インタプリタの別の場所にPL/SQL構成体を入力する場合と同様に、トリガー本体を入力するのに、行継続文字は必要ありません(これらを使用することもできません)。

TRIGGERは、特に条件付きのブレークポイントを作成するのに便利です。これは、トリガー本体の複雑な制御ロジック内から、例外DEBUG.BREAKを呼び出すことによって行われます。例外はデバッガによって検出され、ブレークポイントがトリガーの位置に設定されているかのようにプログラムの実行に割り込み、インタプリタに制御を渡します。

TRIGGERコマンドの例

次のトリガーは、ローカルNUMBER変数*i*が100を超える場合にのみ到達する*my_proc*の10行目に、条件付きブレークポイントを確立します。

```
.TRIGGER PROC my_proc LINE 10 IS  
IF DEBUG.GETN('i') > 100 THEN  
    RAISE DEBUG.BREAK;  
END IF;
```

トリガーは、プログラムの実行をトレースするのにも使用することができます。次のトリガーは、すべてのソース文を実行時にリストします。

```
.TRIGGER * IS debug.interpret('LIST PC');
```

次のコマンドは、ユーザー*scott*が所有するスキーマから、サーバー側のプログラム単位*my_proc*の8行目にトリガーを設定します。

```
.TRIGGER USER scott PROC my_proc LINE 8
```

または

```
.TRIGGER PROC scott.my_proc LINE 8
```


デバック・コマンド

DESCRIBE (ローカル) コマンド

説明

現行の有効範囲の位置に対してローカルな変数またはパラメータの名前、タイプおよび値を表示します。

構文

```
DESCRIBE LOCAL name
DESCRIBE PARAMETER name
DESCRIBE VARIABLE name
```

キーワードおよび値

LOCAL <i>name</i>	現行の有効範囲の位置に対してローカルなパラメータまたは変数を指定します。
PARAMETER <i>name</i>	現行の有効範囲の位置に対してローカルなパラメータを指定します。
VARIABLE <i>name</i>	現行の有効範囲の位置に対してローカルな変数を指定します。

DESCRIBE (ローカル) コマンドの例

次のコマンドは、パラメータ *p1* の情報を表示します。

```
.DESCRIBE PARAM p1
```

次のコマンドは、ローカル変数 *sal* の情報を表示します。

```
.DESCRIBE LOCAL sal
```

GO コマンド

説明

ブレークポイントまたはデバッグ・トリガーの後、プログラムの実行を再開し、続行します。

構文

```
GO
```

コメント

GOはプログラムの実行を再開し、現在実行中のプログラムが終了するか、デバッグ・アクションが割り込むまで続けます。

GOコマンドの例

次のコマンドは、プログラムの実行を再開します。

```
.GO
```

RESETコマンド

説明

現行のデバッグ・レベルでの実行を継続せずに、外部のデバッグ・レベルに制御を戻します。

構文

```
RESET LEVEL number
```

キーワードおよび値

LEVEL <i>number</i>	Specifies an outer debug level.
---------------------	---------------------------------

コメント

RESETは、現行のデバッグ・レベルと、場合によっては上位のデバッグ・レベルで、実行を中止します。

レベル数に負の値を指定すると、相対的なリセットを実行できます。オプションを指定しないでRESETを起動すると、常に1番上のレベルに戻ります。

RESETコマンドの例

次のコマンドは、前のデバッグ・レベルにリセットします。

```
.RESET LEVEL -1
```

次のコマンドは、1番上のレベルにリセットします。

```
.RESET
```

SET (有効範囲) コマンド

説明

現行の有効範囲の位置を、指定したコール・スタック・フレームに変更します。現行の有効範囲の位置は、ローカル変数参照がインタプリタでどのように処理されるかに影響します。

構文

```

SET SCOPE FRAME number
SET SCOPE UP [COUN number]
SET SCOPE DOWN [COUNT number]
SET SCOPE TOP
SET SCOPE BOTTOM
SET SCOPE PROGRAMUNIT name
SET SCOPE PACKAGE name
SET SCOPE SUBPROGRAM name
SET SCOPE PROCEDURE name
SET SCOPE FUNCTION name

```

キーワードおよび値

FRAME <i>number</i>	スタックを番号で指定します。
UP	スタックの1番上への相対的な移動を指定します。
DOWN	スタックの1番下への相対的な移動を指定します。
COUNT <i>number</i>	指定した方向 (UPまたはDOWN) への繰返し回数を指定します。デフォルトは1です。
TOP	コール・スタックの1番上を指定します。
BOTTOM	コール・スタックの1番下を指定します。
PROGRAMUNIT <i>name</i>	プログラム単位を指定します。
PACKAGE <i>name</i>	パッケージを指定します。
SUBPROGRAM <i>name</i>	サブプログラムを指定します。
PROCEDURE <i>name</i>	プロシージャを指定します。
FUNCTION <i>name</i>	ファンクションを指定します。

コメント

スタックフレームには、0 (1番上) から n (1番下) まで番号が付けられています。

SETコマンドの例

次のコマンドは、1つ上のスタック・フレームに移動します。

```
.SET SCOPE UP
```

次のコマンドは、2つ下のスタック・フレームに移動します。

```
.SET SCOPE DOWN COUNT 2
```

次のコマンドは、ファンクション *func1* に関係付けられたスタック・フレームに移動します。

```
.SET SCOPE FUNCTION func1
```

次のコマンドは、スタック・フレームの1番前へ移動します。

```
.SET SCOPE TOP
```

次のコマンドは、5番目のスタック・フレームに移動します。

```
.SET SCOPE FRAME 5
```

SHOW (コール・スタック) コマンド

説明

現行のコール・スタック上のスタック・フレームをリストします。

構文

```
SHOW STACK
SHOW SCOPE
```

キーワードおよび値

STACK	コール・スタック上のすべてのスタック・フレームについて、プログラム単位名と行番号をリストします。
SCOPE	コール・スタックの1番上のスタック・フレームから、現行の有効範囲の位置を含む枠までをリストします。

SHOW (コール・スタック) コマンドの例

次のコマンドは、現行のコール・スタックをリストします。

```
.SHOW STACK
```

STEPコマンド

説明

割り込まれたプログラム単位の実行を進めます。

構文

```
STEP INTO
STEP OVER
STEP OUT
STEP TO PROGRAMUNIT name[LINE number]
STEP TO PACKAGE name [LINE number]
STEP TO SUBPROGRAM name [LINE number]
STEP TO PROCEDURE name [LINE number]
STEP TO FUNCTION name [LINE number]
```

```
STEP TO ACTION number
STEP TO BREAKPOINT number
STEP TO TRIGGER number
STEP TO . [LINE number]
STEP COUNT number
```

キーワードおよび値

INTO	サブプログラム・コールへの内部処理を可能にします。これはキーワードが指定されない場合のデフォルトです。
OVER	コールされたサブプログラム内のステップ実行を防ぎます。
OUT	実行を再開して、現行のサブプログラムが終了するまで続けます。
TO ...	指定したソースの位置まで、実行を継続します。T0オプションを使用することは、指定した位置に一時的なブレークポイントを設定するのと似ています。
PROGRAMUNIT <i>name</i>	プログラム単位を指定します。
PACKAGE <i>name</i>	パッケージを指定します。
SUBPROGRAM <i>name</i>	サブプログラムを指定します。
PROCEDURE <i>name</i>	プロシージャを指定します。
FUNCTION <i>name</i>	ファンクションを指定します。
ACTION <i>number</i>	デバッグ・アクション (ブレーク・ポイントまたはトリガー) を指定します。
BREAKPOINT <i>number</i>	ブレーク・ポイントを指定します。
TRIGGER <i>number</i>	トリガーを指定します。
.	現行のソース位置を指定する。
LINE <i>number</i>	プログラム単位の行を指定します。
COUNT <i>number</i>	STEPコマンドを (他のオプションで指定した内容を含め) 何回繰り返すかを示します。デフォルトは1です。

コメント

制御は、指定した一連の文が実行されると、現行のデバッグ・レベルに戻ります。

STEPコマンドの例

次のコマンドは、実行を再開し、最初のブレークポイントまで続けます。

```
.STEP TO BREAK 1
```

次のコマンドは、5つの行に対して実行を再開します。

```
.STEP COUNT 5
```

ライブラリ・コマンド

ATTACHコマンド

説明

現行のセッションにPL/SQLライブラリを連結します。

構文

```
ATTACH LIBRARY [directory] name [extension]
                [FILESYSTEM | DB]
                [BEFORE library]
                [AFTER library]
                [START | END]
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリ名を指定します。ファイル・システムに格納されている場合は、オプションでディレクトリ・パスと拡張子も指定できます。
FILESYSTEMまたはDB	指定されたライブラリが、ファイル・システムと、現在接続されているデータベースのどちらに格納されているかを指定します。どちらのキーワードも指定されない場合、まずファイル・システム内の指定されたライブラリのアクセスを試み、ライブラリが見つからない場合、現行のデータベースをアクセスします。
BEFORE <i>library</i>	指定された連結ライブラリの前にライブラリを連結することを指定します。
AFTER <i>library</i>	指定された連結ライブラリの後ろにライブラリを連結することを指定します。
STARTまたはEND	連結ライブラリを、連結リストの始めと終わりのどちらに置くかを指定します。デフォルトはSTARTです。

コメント

ファイル・システム内でライブラリ連結しようとする、*directory*と*extension*が明示的に指定されない限り、*name*にロード・パスと拡張子.pllが適用されます。*directory*の構文が、オペレーティング・システム固有であることを注意してください。構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

ライブラリは読み込み専用で連結されます。ライブラリの内容を変更する場合は、OPENコマンドを使用してそのライブラリを編集用にオープンします。

ATTACHコマンドの例

次のコマンドは、ファイル*lib1*に常駐するライブラリに連結します。

```
.ATTACH LIB lib1 FILE
```

CLOSEコマンド

説明

現在のオープンされているライブラリの中から、1つ以上のライブラリをクローズします。

構文

```
CLOSE LIBRARY name [, name...] [DISCARD]
```

キーワードおよび値

LIBRARY <i>name</i>	現在オープンしているライブラリの中で、クローズするものを1つ以上指定します。
DISCARD	最後の保存以降ライブラリに対して行われた変更を廃棄します。

コメント

ライブラリをクローズすると、そのライブラリから現在の環境にロードされたプログラム単位がすべて削除されます。ライブラリを表すのに使用されている名前領域も削除されます。

ライブラリを自動的にクローズすると、オープン以降ライブラリに対して行われた変更がすべて保存されます。DISCARDを指定すると、最後の保存操作以降ライブラリに対して行われた変更が廃棄されます。

CLOSEコマンドの例

次のコマンドは、ライブラリ *lib1* および *lib2* をクローズします。

```
.CLOSE LIB lib1, lib2
```

COMPILE (ライブラリ) コマンド

説明

1つ以上のオープンされているライブラリのすべてのプログラム単位を、コンパイルまたは再コンパイルします。

構文

```
COMPILE LIBRARY name [, name...] [INCREMENTAL]
```

キーワードおよび値

LIBRARY <i>name</i>	コンパイルするプログラム単位を含む、1つ以上のオープンされているライブラリを指定します。
---------------------	--

INCREMENTAL	ライブラリ内の、コンパイルが必要なプログラム単位のみをコンパイルする。
-------------	-------------------------------------

コメント

起動時にCOMPILEはまず、現在ロードされているプログラム単位の中にコンパイル対象のライブラリ内のプログラム単位と名前およびタイプが一致するものがあるかどうかチェックします。一致するものが少なくとも1つあれば、コンパイルを続行するかどうかの確認を求められます。

「はい」と答えると、一致するすべてのプログラム単位が現在の環境から削除され、コンパイルされた後、指定されたオープンされているライブラリ内に保存されます。「いいえ」と答えると、操作を終了します。

注意: コンパイルされたプログラム単位は、オープンされているライブラリに保存されますが、現在の環境には再ロードされません。それらを環境内に再ロードするには、(インタプリタのコマンド・ラインまたは「ファイル」→「ロード」を介して) LOADコマンドを起動します。

INCREMENTALが指定されないと、ライブラリ内のすべてのプログラム単位がコンパイルされます。

COMPILE (ライブラリ) コマンドの例

次のコマンドは、*lib1*というオープンされているライブラリ内のすべてのプログラム単位をコンパイルします。

```
.COMPILE LIB lib1
```

CREATE (ライブラリ) コマンド

説明

ファイル・システムまたは現行のデータベースのどちらかに格納する、新しいライブラリを作成します。

構文

```
CREATE LIBRARY [directory] lib-name [extension]  
    [SOURCE pld-file]  
    [NOCOMPILE]  
    [FILESYSTEM | DB]  
    [BEFORE | AFTER]  
    [NOCONFIRM]
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリ名を指定します。ファイル・システムに作成されている場合は、オプションでディレクトリ・パスと拡張子も指定できます。
---------------------	---

SOURCE <i>pld-file</i>	1つ以上のプログラム単位のソースを含むファイルを指定します。
NOCOMPILE	新規に作成されたライブラリの内容がコンパイルされないようにします。
FILESYSTEMまたはDB	指定されたライブラリを、ファイル・システムと、現在接続されているデータベースのどちらに作成するかを示します。デフォルトはFILESYSTEMです。
BEFOREまたはAFTER	連結ライブラリが連結リストの最初に配置されるか、最後に配置されるかを指示する。デフォルトはBEFOREです。
NOCONFIRM	既存のライブラリを確認のプロンプトなしで、上書きするように指定します。

コメント

ファイル・システムに作成されるライブラリでは、ライブラリ名はファイルのベース名(ファイル名全体から先頭に付いているディレクトリと後続の拡張子を差し引いたもの)で指定されます。たとえばUNIXでは、ファイル/private/libs/emplib.pllには、*emplib*という名前のライブラリが入っています。

*directory*の構文はオペレーティング・システムに固有のものです。構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

新しく作成されたライブラリは、自動的にオープンします。ライブラリがオープンされると、INSERTおよびDELETEコマンドを使用しその内容を変更できます。

SOURCEキーワードを使用して、指定された*pld-file*に含まれている1つ以上のプログラム単位のソースを、ただちに新規作成のライブラリに挿入することができます。NOCOMPILEキーワードを指定しない限り、その後ライブラリは(COMPILER LIBRARYコマンドにより)コンパイルされます。

既存のライブラリと同じ名前のライブラリを作成しようとする、メッセージ・ボックスが表示され、既存のライブラリを上書きするかどうかの確認が求められます。コマンド文字列にNOCONFIRMを指定すると、警告は抑止されます。

CREATE (ライブラリ) コマンドの例

UNIXでは、次のコマンドが、ファイル/private/libs/lib1.pllに存在するlib1という名前の新ライブラリを作成します。

```
.CREATE LIBRARY /private/libs/lib1.pll FILE
```

DELETE (ライブラリ) コマンド

説明

現行のデータベースに存在する1つ以上のライブラリを削除します。

構文

```
DELETE LIBRARY name [, name...]
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリを指定します。
---------------------	--------------

コメント

現在連結されているライブラリを削除することはできません。削除の前に、DETACHコマンドでライブラリを連結解除します。

DELETE (ライブラリ) コマンドの例

次のコマンドは、データベースからライブラリ *lib1* を削除します。

```
.DELETE LIB lib1
```

DELETE (ライブラリ・プログラム単位) コマンド

説明

オープンされているライブラリから、1つ以上のプログラム単位を削除します。

構文

```
DELETE PROGRAMUNIT name [, name...]  
    LIBRARY name  
    [SPECIFICATION | BODY]  
DELETE PACKAGE name [, name...]  
    LIBRARY name  
    [SPECIFICATION | BODY]  
DELETE SUBPROGRAM name [, name...]  
    LIBRARY name  
    [SPECIFICATION | BODY]  
DELETE PROCEDURE name [, name...]  
    LIBRARY name  
    [SPECIFICATION | BODY]  
DELETE FUNCTION name [, name...]  
    LIBRARY name  
    [SPECIFICATION | BODY]
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
-------------------------	---------------------

PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
LIBRARY <i>name</i>	プログラム単位の削除元ライブラリを指定します。
SPECIFICATIONまたはBODY	プログラム単位の仕様部と本体のどちらを削除するかを指定します。

DELETE (ライブラリ・プログラム単位) コマンドの例

次のコマンドは、*lib1*という名前のライブラリから*p2*という名前のパッケージを削除します。

```
.DELETE PACKAGE p2 LIBRARY lib1
```

DESCRIBE (ライブラリ) コマンド

説明

連結ライブラリの詳細を表示します。

構文

```
DESCRIBE LIBRARY name
```

キーワードおよび値

LIBRARY <i>name</i>	連結ライブラリの名前を指定します。
---------------------	-------------------

コメント

ライブラリについて表示される情報には、連結されているモードとディレクトリ位置、内容が含まれています。

DESCRIBE (ライブラリ) コマンドの例

次のコマンドは、*lib1*という名前のライブラリについての情報を表示します。

```
.DESCRIBE LIBRARY lib1
```

DETACHコマンド

説明

現行の連結ライブラリの中から、1つ以上のライブラリを削除します。

構文

```
DETACH LIBRARY name[, name...]
```

キーワードおよび値

LIBRARY <i>name</i>	1つ以上の連結ファイルを指定します。
---------------------	--------------------

コメント

ライブラリを連結解除すると、そのライブラリから現在の環境にロードされた未変更のプログラム単位がすべて削除されます。ライブラリからロードされるプログラム単位が環境内で変更（たとえば、コンパイルによって）されると、ライブラリの連結解除時に削除されないことに注意してください。

DETACHコマンドの例

次のコマンドは、ライブラリ *lib1* と *lib2* を連結解除します。

```
.DETACH LIBRARY lib1, lib2
```

EXPORT (ライブラリ) コマンド

説明

ライブラリのソースをテキスト・ファイルに書き込みます。

構文

```
EXPORT {LIBRARY name}  
      FILE [directory] name [extension]  
      [NOWARN]
```

キーワードおよび値

LIBRARY <i>name</i>	連結ライブラリを指定します。
FILE <i>name</i>	オプションのディレクトリ・パスおよび拡張子を含めて、ファイル名を指定します。
NOWARN	ビルトイン・プログラム単位が連結ライブラリに追加されなかった、という警告を抑止します。

コメント

指定がない場合のデフォルトのファイル拡張子は.pldです。*directory*の構文はオペレーティング・システムに固有のもので、構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

EXPORT (ライブラリ) コマンドの例

次のコマンドは、ライブラリ`my_lib`のソースを、ファイル`lib1.pld`に書き込みます。

```
.EXPORT LIB my_lib FILE lib1
```

GENERATEコマンド

説明

現在のオープンされているライブラリのランタイム・バージョンを作成します。

構文

```
GENERATE LIBRARY name
        FILE [directory]name
        [INCREMENTAL]
```

キーワードおよび値

LIBRARY <i>name</i>	オープンされているライブラリの名前を指定します。
FILE <i>name</i>	オプションのディレクトリ・パスを含めて、ランタイム・ライブラリのファイル名を指定します。デフォルトのファイル拡張子 <code>.plx</code> が、自動的に割り当てられます。
INCREMENTAL	オープンされているライブラリ内のコンパイルされていないプログラム単位のみをコンパイルするように指定します。

コメント

このコマンドは、テンポラリ・ライブラリを作成し、オープンされているライブラリからすべてのプログラム単位をコピーしてテンポラリ・ライブラリに挿入し、プログラム単位をコンパイルし、NOSOURCEとNODIANAキーワードを使用してテンポラリ・ライブラリでSAVEコマンドを実行します。

その結果作成されるライブラリは、`.plx`ファイル拡張子のランタイム専用ライブラリとして認識されます。異なるファイル拡張子を指定すると、その拡張子が`.plx`のかわりに使用されます。

GENERATEコマンドの例

次のコマンドは、オープンされているライブラリ`mylib.pll`をベースにして、`runlib1.plx`という名前のランタイム・ライブラリを作成します。

```
.GENERATE LIB mylib FILE runlib1
```

INCREMENTALキーワードが指定されていないので、ライブラリ`mylib`内のプログラム単位がすべて強制的にコンパイルされます。

GRANTコマンド (ライブラリ・コマンド)

説明

ユーザーに、データベースに格納されたライブラリへのアクセス権限を付与します。

構文

```
GRANT LIBRARY name USER name
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリを指定します。
USER <i>name</i>	ユーザー名を指定します。

コメント

任意のユーザー名を1つ、またはPUBLIC (すべてのユーザー) を指定できます。

GRANTコマンドの例 (ライブラリ・コマンド)

次のコマンドは、ユーザーSCOTTにデータベース・ライブラリ*lib1*へのアクセス権限を付与します。

```
.GRANT LIB lib1 USER scott
```

INSERT (ライブラリ・プログラム単位) コマンド

説明

オープンされているライブラリに、1つ以上のプログラム単位を挿入します。

構文

```
INSERT PACKAGE name [, name...]
    [SPECIFICATION | BODY]
    LIBRARY name
    [NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]
INSERT SUBPROGRAM name [, name...]
    [SPECIFICATION | BODY]
    LIBRARY name
    [NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]
INSERT PROCEDURE name [, name...]
    [SPECIFICATION | BODY]
    LIBRARY name
```

```

[NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]
INSERT FUNCTION name [, name...]
[SPECIFICATION | BODY]
LIBRARY name
[NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]

```

キーワードおよび値

PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
SPECIFICATIONまたはBODY	指定されたプログラム単位の仕様部あるいは本体を指定します。どちらのキーワードも指定されない場合は、双方がライブラリに挿入されます。
LIBRARY <i>name</i>	ターゲット・ライブラリを指定します。
NOPCODE	PCODEなしでライブラリにプログラム単位を追加します。
NODIANA	DIANAなしでライブラリにプログラム単位を追加します。
NOSOURCE	ソース・コードなしでライブラリにプログラム単位を追加します。
NOWARN	ビルトイン・プログラム単位がオープンされているライブラリに挿入されていないという警告を抑制します。

コメント

NODIANAとNOSOURCEを使用して、PL/SQLライブラリのサイズを大幅に削減することができます。

注意: DIANAを含まないプログラム単位の参照情報をコンパイルすることはできないので、NODIANAとNOSOURCEでライブラリに挿入されたプログラム単位は、ランタイム環境でのみ使用可能です。

ビルトイン・プログラム単位をオープンされているライブラリに挿入しようとする（たとえば .INSERT PROG * LIB lib3）、インタプリタ・ペインに警告（警告: プログラム単位<progunit name>に、挿入するソースがありません...）が表示されます。コマンド文字列にNOWARNを指定すると、警告は抑止されます。

INSERT (ライブラリ・プログラム単位) コマンドの例

次のコマンドは、パッケージ*p1*と*p2*を、*lib1*という名前のライブラリに挿入します。

```
.INSERT PACKAGE p1,p2 LIBRARY lib1
```

LOAD (ライブラリ・プログラム単位) コマンド

説明

連結ライブラリから、1つ以上のプログラム単位をロードします。

構文

```
LOAD PROGRAMUNIT name [, name...]
    LIBRARY name
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD PACKAGE name [, name...]
    LIBRARY name
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD SUBPROGRAM name [, name...]
    LIBRARY name
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD PROCEDURE name [, name...]
    LIBRARY name
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD FUNCTION name [, name...]
    LIBRARY name
    [SPECIFICATION | BODY]
    [NOCONFIRM]
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
LIBRARY <i>name</i>	連結ライブラリを指定します。
SPECIFICATIONまたはBODY	プログラム単位の仕様部と本体のどちらをロードするかを指定します。どちらも指定しない場合、両方がロードされます。
NOCONFIRM	既存のプログラム単位を、確認のプロンプトなしで再定義するように指定します。

コメント

ライブラリ・プログラム単位を既存のプログラム単位と同じ名前およびタイプでロードしようとする、メッセージ・ボックスが表示され、既存のプログラム単位を再定義するかどうかの確認を求められます。コマンド文字列にNOCONFIRMを指定すると、警告は抑止されます。

LOAD (ライブラリ・プログラム単位) コマンドの例

次のコマンドは、ファンクション $f1$ とプロシージャ $p3$ をロードします。どちらも連結ライブラリ $lib1$ に格納されています。

```
.LOAD PROGRAMUNIT f1, p3 LIB lib1
```

OPENコマンド

説明

変更するライブラリをオープンします。

構文

```
OPEN LIBRARY [directory] lib-name [extension]  
          [FILESYSTEM | DB]
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリ名を指定します。ファイル・システムに格納されている場合は、オプションでディレクトリ・パスと拡張子も指定できます。
FILESYSTEMまたはDB	指定されたライブラリが、ファイル・システムと、現在接続されているデータベースのどちらに格納されているかを指定します。デフォルトはFILESYSTEMです。

コメント

FILESYSTEMとDBがどちらも指定されていない場合は、Procedure Builderは、まずファイル・システムの指定されたライブラリにアクセスします。アクセスに失敗した場合には、現行のデータベースにアクセスします。

ファイル・システム内でライブラリをオープンしようとする、*directory*と*extension*が明示的に指定されない限り、*lib-name*にロード・パスと拡張子 $.pll$ が適用されます。*directory*の構文が、オペレーティング・システム固有であることを注意してください。構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

OPENコマンドの例

次のコマンドは、*lib1*という名前のライブラリをオープンします。このライブラリはファイル・システムに格納されています。

```
.OPEN LIB /private/libs/lib1
```

次のコマンドは、*libdb*という名前のライブラリをオープンします。このライブラリは現行のデータベースに格納されています。

```
.OPEN LIB libdb DB
```

RENAME (ライブラリ) コマンド

説明

現行のデータベースに存在するライブラリを改名します。

構文

```
RENAME LIBRARY oldname TO newname
```

キーワードおよび値

LIBRARY <i>oldname</i>	現行のライブラリ名を指定します。
TO <i>newname</i>	新ライブラリ名を指定します。

コメント

現在連結されているライブラリの名前に改名することはできません。DETACHコマンドで、新規名で指定されているライブラリを連結解除するか、または異なる新規名を使用してください。

このコマンドは、ファイルに格納されているライブラリの改名には使用できません。オペレーティング・システムのコマンドを使用する必要があります。

RENAME (ライブラリ) コマンドの例

次のコマンドは、データベース・ライブラリ*lib1*を*lib4*に改名します。

```
.RENAME LIB lib1 TO lib4
```

REVERTコマンド

説明

1つ以上のライブラリを以前の保存状態に戻します。

構文

```
REVERT LIBRARY name[, name...]
```

キーワードおよび値

LIBRARY <i>name</i>	1つ以上のオープンされているライブラリを指定します。
---------------------	----------------------------

REVERTコマンドの例

次のコマンドは、ライブラリ *lib1* を回復します。

```
.REVERT LIB lib1
```

REVOKEコマンド (ライブラリ・コマンド)

説明

データベースに格納されているライブラリに対するユーザーのアクセス権限を取り消します。

構文

```
REVOKE LIBRARY name USER name
```

キーワードおよび値

LIBRARY <i>name</i>	ライブラリを指定します。
USER <i>name</i>	ユーザーを指定します。

コメント

任意のユーザー名を1つ、またはPUBLIC (すべてのユーザー) を指定できます。

REVOKEコマンドの例 (ライブラリ・コマンド)

次のコマンドは、データベース・ライブラリ *lib1* に対するユーザー SCOTT のアクセス権限を取り消します。

```
.REVOKE LIB lib1 USER scott
```

SAVEコマンド

説明

1つ以上のオープン・ファイルに対する変更を保存します。

構文

```
SAVE LIBRARY name [, name...]  
    [AS name]  
    [NOPCODE]  
    [NODIANA]  
    [NOSOURCE]
```

キーワードおよび値

LIBRARY <i>name</i>	1つ以上のオープンされているライブラリを指定します。
AS <i>name</i>	保存するライブラリの新規名を指定します。
NOPCODE	PCODEなしでライブラリを保存します。
NODIANA	DIANAなしでライブラリを保存します。
NOSOURCE	ソース・コードなしでライブラリを保存します。

コメント

ライブラリを保存すると、暗黙的COMMITが発行されます。この操作は、ライブラリ・オブジェクトのみでなく、あらゆるデータベース・オブジェクトに対する変更をコミットします。COMMITに関する詳細は、『Oracle8 SQLリファレンス・マニュアル』を参照してください。

NODIANAとNOSOURCEを使用して、PL/SQLライブラリのサイズを大幅に削減することができます。

注意: DIANAを含まないライブラリ・プログラム単位の参照情報をコンパイルすることはできないので、NODIANAとNOSOURCEで保存されたライブラリは、ランタイム環境でのみ使用可能です。

SAVEコマンドの例

次のコマンドは、ライブラリ`lib1`および`lib2`を保存します。

```
.SAVE LIB lib1, lib2
```

次のコマンドは、ライブラリ`lib1`を`lib1a`として保存します。

```
.SAVE LIB lib1 AS lib1a
```

次のコマンドは、ライブラリ`lib1`および`lib2`をDIANA、ソースなしで保存します。

```
.SAVE LIB lib1, lib2 NODIANA NOSOURCE
```

SHOW (ライブラリ) コマンド

説明

現在のライブラリを表示します。

構文

```
SHOW LIBRARIES
```

SHOW (ライブラリ) コマンドの例

次のコマンドは、現在のライブラリを表示します。

```
.SHOW LIB
```


ロード・パス・コマンド

DELETE (ロード・パス) コマンド

説明

ロード・パスをリセットして、要素を含まないようにします。

構文

```
DELETE LOADPATH
```

DELETE (ロード・パス) コマンドの例

次のコマンドは、ロード・パスのすべてのエントリを消去します。

```
.DELETE LOADPATH
```

DESCRIBE (ロード・パス) コマンド

説明

現行のロード・パスを表示します。

構文

```
DESCRIBE LOADPATH
```

DESCRIBE (ロード・パス) コマンドの例

次のコマンドは、現行のロード・パスのすべてのエントリを表示します。

```
.DESCRIBE LOADPATH
```

INSERT (ロード・パス) コマンド

説明

ロード・パスにディレクトリを追加します。

構文

```
INSERT LOADPATH  
    {DIRECTORY path [, path...] | CURRENTDIR}  
    [BEFORE | AFTER]
```

キーワードおよび値

LOADPATH	現在のロード・パスを指定します。
DIRECTORY <i>path</i>	1つ以上のディレクトリを指定します。
CURRENTDIR	現在のディレクトリ・パスを指定します。
BEFOREまたはAFTER	ロード・パスにある既存のディレクトリの前と後ろのどちらにディレクトリを挿入するかを指定します。デフォルトはAFTERです。

コメント

*path*の構文は、オペレーティング・システム固有です。構文の詳細は、使用しているオペレーティング・システム用のOracle製品マニュアルを参照してください。

INSERT (ロード・パス) コマンドの例

UNIXでは、次のコマンドで、ディレクトリ/*usr/plsql*をロード・パスに追加します。

```
.INSERT LOADPATH DIRECTORY /usr/plsql
```


ロギング・コマンド

DISABLE (ロギング) コマンド

説明

現行のログ・ファイルへのログ作成を一時停止します。

構文

```
DISABLE LOGGING
```

コメント

ログ・ファイルが(LOGコマンドで)指定されていない場合、あるいはログ作成がすでに使用不可になっている場合には、DISABLEはロギングには影響しません。

ENABLEコマンドにより停止中のログを再開できます。

DISABLE (ロギング) コマンド例

次のコマンドは、一時的にロギングを停止します。

```
.DISABLE LOG
```

ENABLE (ロギング) コマンド

説明

現行のログ・ファイルへのログ作成を再開します。

構文

```
ENABLE LOGGING
```

コメント

ログ・ファイルがLOGコマンドで指定されていない場合、あるいはログ作成がすでに使用可能になっている場合には、ENABLEは影響しません。

DISABLEコマンドで、一時的にログ作成を使用不可にすることができます。

ENABLE (ロギング) コマンド例

次のコマンドは、一時停止されたログ作成を再開します。

```
.ENABLE LOG
```

LOGコマンド

説明

インタプリタの入力と出力のコピーを、指定されたログ・ファイルに保存します。

構文

```
LOG FILE [directory] name[extension]
      [APPEND]
      [ENABLED | DISABLED]
      [OFF]
```

キーワードおよび値

FILE <i>name</i>	ログ・ファイルの名前(および、オプションとしてディレクトリ・パスとファイル拡張子)を指定します。
APPEND	指定されたファイルにログ出力を追加することを指示します。指定しない場合には、ファイルが上書きされます。
ENABLEDまたはDISABLED	ログ取得の初期設定を使用可能と使用不可のどちらにするかを指定します。デフォルトはENABLEDです。
OFF	ログ作成を終了し、ログ・ファイルを保存します。

コメント

ログ・ファイル拡張子を指定しないと、デフォルトで.logになります。*directory*の構文はオペレーティング・システムに固有のものです。構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

指定されたログ・ファイルが存在しない場合は、指定されたディレクトリに新規ファイルが作成されます。ディレクトリ・パスが指定されていない場合は、現行のディレクトリにログ・ファイルが作成されます。

DISABLEコマンドを使用して一時的にログ作成を使用不可にした後、ENABLEコマンドを使用して再び使用可能にすることができます。

LOGコマンド例

次のコマンドは、インタプリタの入力と出力の、現行のディレクトリにあるファイルdebug.logへのログ作成を開始します。

```
.LOG FILE debug
```

次のコマンドは、ログ作成を終了し、ログ・ファイルを保存します。

```
.LOG OFF
```


プログラム単位コマンド

COMPILE (プログラム単位) コマンド

説明

指定されたプログラム単位を、コンパイルまたは再コンパイルします。

構文

```
COMPILE PROGRAMUNIT name [, name...]
    [SPECIFICATION | BODY]
COMPILE PACKAGE name [, name...]
    [SPECIFICATION | BODY]
COMPILE SUBPROGRAM name [, name...]
    [SPECIFICATION | BODY]
COMPILE PROCEDURE name [, name...]
    [SPECIFICATION | BODY]
COMPILE FUNCTION name [, name...]
    [SPECIFICATION | BODY]
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
SPECIFICATIONまたはBODY	プログラム単位の仕様部と本体のどちらをコンパイルするかを指定します。デフォルトでは、両方がコンパイルされます。

COMPILE (プログラム単位) コマンドの例

次のコマンドは、プロシージャ*proc1*およびパッケージ*pkg1*をコンパイルします。

```
.COMPILE PROG proc1, pkg1
```

DELETE (プログラム単位) コマンド

説明

現行のセッションから、1つ以上のプログラム単位を削除します。

構文

```
DELETE PROGRAMUNIT name [, name...]
```

```

    [SPECIFICATION | BODY]
DELETE PACKAGE name [, name...]
    [SPECIFICATION | BODY]
DELETE SUBPROGRAM name [, name...]
    [SPECIFICATION | BODY]
DELETE PROCEDURE name [, name...]
    [SPECIFICATION | BODY]
DELETE FUNCTION name [, name...]
    [SPECIFICATION | BODY]

```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
SPECIFICATIONまたはBODY	プログラム単位の仕様部と本体のどちらを削除するかを指定します。デフォルトでは、どちらも削除されます。

コメント

プログラム単位とそのサブオブジェクト(タイプ、変数、サブプログラムなど)は、一度現行のセッションから削除されると、セッション内における定義が消滅します。現行のセッションからプログラム単位を削除すると、それに依存する他のプログラム単位が無効になることがあります。これは、プログラム単位の仕様部を変更したときに発生する状況に似ています。

DELETE (プログラム単位) コマンドの例

次のコマンドは、現行のセッションから *p2* というパッケージを削除します。

```
.DELETE PACKAGE p2
```

DESCRIBE (プログラム単位) コマンド

説明

特定のプログラム単位の詳細を表示します。

構文

```

DESCRIBE PROGRAMUNIT name [SPECIFICATION | BODY]
DESCRIBE PACKAGE name [SPECIFICATION | BODY]
DESCRIBE SUBPROGRAM name [SPECIFICATION | BODY]
DESCRIBE PROCEDURE name [SPECIFICATION | BODY]

```

DESCRIBE FUNCTION *name* [SPECIFICATION | BODY]

キーワードおよび値

PROGRAMUNIT <i>name</i>	プログラム単位を指定します。
PACKAGE <i>name</i>	パッケージを指定します。
SUBPROGRAM <i>name</i>	サブプログラムを指定します。
PROCEDURE <i>name</i>	プロシージャを指定します。
FUNCTION <i>name</i>	ファンクションを指定します。
SPECIFICATIONまたはBODY	プログラム単位の仕様部と本体のどちらをライブラリに追加するかを指定します。デフォルトはSPECIFICATIONです。

コメント

プログラム単位の名前およびタイプ、パラメータ（ある場合）、ディレクトリ位置、コンパイルされているかどうか、編集のためにオープンしているかどうか、クロス・リファレンス情報が表示されます。さらに、パッケージの記述では、そのパッケージがビルトイン・プログラム単位であるかどうか、およびそのパッケージがSTANDARDパッケージに対する拡張であるかどうかも示されます。

パッケージ仕様部の記述では、仕様部に定義されているすべてのサブプログラムがリストされません。

DESCRIBE (プログラム単位) コマンドの例

次のコマンドは、*pkg1*というパッケージの本体に関する情報を表示します。

```
.DESCRIBE BODY PACKAGE pkg1
```

DISABLE (コンパイラ・オプション) コマンド

説明

1つ以上のコンパイラ・オプションを、一時的に削除します。

構文

```
DISABLE COMPILER SIZECHECK
```

コメント

SIZECHECKコンパイラ・オプションは、バッチ・コンパイルでは自動的に使用不可になります。コンパイラ・オプションを再度アクティブにするには、ENABLEコマンドを使用します。

DISABLE (コンパイラ・オプション) コマンドの例

次のコマンドは、一時的に、サイズチェック・コンパイル・オプションを使用不可にします。

```
.DISABLE COMPILER SIZECHECK
```

ENABLE (コンパイラ・オプション) コマンド

説明

1つ以上のコンパイラ・オプションをアクティブ化または再アクティブ化します。

構文

```
ENABLE COMPILER SIZECHECK
```

コメント

SIZECHECKコンパイラ・オプションをアクティブ化できるのは、対話形式のコンパイルに対してのみです。いったん使用可能になると、このコンパイラ・オプションは使用不可にされるまでアクティブ状態が続きます。バッチ・コンパイルの場合、このオプションは自動的に使用不可になります。

プログラム単位のソースのサイズがオペレーティング・システムのメモリー割当て制限に近づいたときに、警告が出るようにするため、プログラム単位をコンパイルする前にはサイズチェック・オプションを使用可能にしてください。ソースのサイズがオペレーティング・システムの制限に近い場合、そのソースがコンパイルされた状態では、より大きくなることが多いので、オペレーティング・システムの制限を超えることがあります。

プログラム単位のコンパイルされた状態が、オペレーティング・システム固有のメモリー割当て制限に近づいたり超えたりする場合にも、サイズチェック・オプションは警告を出します。

プログラム単位のソースまたはコンパイル済みのプログラム単位がオペレーティング・システム固有のメモリー割当ての制限を超える場合は、プログラム単位を小さいプログラム単位に分割することをお勧めします。

ご使用のプラットフォームのメモリー割当て制限については、オペレーティング・システムのマニュアルを確認してください。

コンパイラ・オプションを一時的に削除するには、DISABLEコマンドを使用します。

ENABLE (コンパイラ・オプション) コマンドの例

次のコマンドは、サイズチェック・コンパイル・オプションを使用可能にします。

```
.ENABLE COMPILER SIZECHECK
```

EXECUTEコマンド

説明

無名ブロックまたはパラメータがないプロシージャを実行します。このコマンドは、Procedure Builderがスタンドアロン・セッションとして起動される場合にのみ有効です。

構文

```
EXECUTE PROGRAMUNIT name
```

```
EXECUTE PROCEDURE name
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	無名ブロックを指定します。
PROCEDURE <i>name</i>	パラメータがないプロシージャを指定します。

コメント

ソースがないか、またはコンパイルされていないプログラム単位およびプロシージャを実行するために、EXECUTEコマンドを使用します。

EXECUTEコマンドの例

*lib1.pll*というライブラリを作成します。NOSOURCEキーワードおよびNODIANAキーワードを使用して、そのライブラリにプロシージャ*x1*を挿入します。*x1*プロシージャはコンパイルされていませんが、次のようにEXECUTEコマンドを使用して、このプロシージャを実行できます。

```
.EXECUTE PROC x1
```

EXPORT (プログラム単位) コマンド

説明

1つ以上のプログラム単位のソースを、テキスト・ファイルに書き込みます。

構文

```
EXPORT PROGRAMUNIT name [, name...]  
    FILE [directory] name [extension]  
    [SPECIFICATION | BODY]  
    [NOWARN]  
EXPORT PACKAGE name [, name...]  
    FILE [directory] name [extension]
```

```

[SPECIFICATION | BODY]
[NOWARN]
EXPORT SUBPROGRAM name [, name...]
    FILE [directory] name [extension]
[SPECIFICATION | BODY]
[NOWARN]
EXPORT PROCEDURE name [, name...]
    FILE [directory] name [extension]
[SPECIFICATION | BODY]
[NOWARN]
EXPORT PROCEDURE name [, name...]
    FILE [directory] name [extension]
[SPECIFICATION | BODY]
[NOWARN]

```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
FILE <i>name</i>	ファイル名を指定します。オプションでディレクトリ・パスおよび拡張子を含めることもできます。
SPECIFICATIONまたはBODY	指定されたプログラム単位 (1つまたは複数) の仕様部と本体のどちらかを指定します。デフォルトでは、両方ともファイルに書き込まれます。
NOWARN	ビルトイン・プログラム単位がエクスポート用のソースを持っていないという警告を抑止します。

コメント

複数のプログラム単位をエクスポートすると、Procedure Builderは前方参照を避けるためにプログラム単位をソートします。つまり各プログラム単位は、参照されるプログラム単位が、参照するプログラム単位の前に置かれます。これにより、エクスポートされたプログラム単位を、INTERPRETを使用してProcedure Builderに再ロードできます。

特に指定しない限り、ファイル拡張子はデフォルトで.PLDになります。*directory*の構文はオペレーティング・システムに固有のもので、構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

ビルトイン・プログラム単位をテキスト・ファイルにエクスポートしようとする(例: .EXPORT PROG * FILE allprogs.txt)、インタプリタ・ペインに警告(警告: プログラム単位<progunit name>にエクスポートするソースがありません...)が表示されます。コマンド文字列にNOWARNを指定すると、警告は抑止されます。

EXPORT (プログラム単位) コマンドの例

次のコマンドは、プロシージャ *p1* およびファンクション *f3* のソースを、ファイル *p11.pld* に書き込みます。

```
.EXPORT PROG p1,f3 FILE p11
```

EXPORT (ストアド・プログラム単位) コマンド

説明

1つ以上のストアド・プログラム単位のソースをテキスト・ファイルに書き込みます。

構文

```
EXPORT {STORED} PROGRAMUNIT schema.name [, schema.name...]
    FILE [directory] name [extension]
    [SPECIFICATION | BODY]
EXPORT {STORED} PACKAGE schema.name [, schema.name...]
    FILE [directory] name [extension]
    [SPECIFICATION | BODY]
EXPORT {STORED} SUBPROGRAM schema.name [, schema.name...]
    FILE [directory] name [extension]
    [SPECIFICATION | BODY]
EXPORT {STORED} PROCEDURE schema.name [, schema.name...]
    FILE [directory] name [extension]
    [SPECIFICATION | BODY]
EXPORT {STORED} FUNCTION schema.name [, schema.name...]
    FILE [directory] name [extension]
    [SPECIFICATION | BODY]
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。
FILE <i>name</i>	ファイル名を指定します。オプションで、ディレクトリ・パスおよび拡張子を含めることもできます。
SPECIFICATIONまたはBODY	指定されたストアド・プログラム単位の仕様部または本体を指定します。デフォルトでは、両方ともファイルに書き込まれます。

コメント

複数のストアド・プログラム単位をエクスポートする場合、Procedure Builderは前方参照を避けるためにストアド・プログラム単位をソートします。つまり各ストアド・プログラム単位は、その参照先のストアド・プログラム単位よりも後ろに配置されます。これにより、エクスポートしたストアド・プログラム単位をINTERPRETを使用してProcedure Builderに再ロードできるようになります。

特に指定しない限り、ファイル拡張子はデフォルトで.PLDになります。*directory*の構文はオペレーティング・システムに固有のもので、構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

EXPORT (ストアド・プログラム単位) コマンドの例

次のコマンドは、ストアド・プロシージャ*p1*およびストアド・ファンクション*f3*のソースをファイル*pl1.pld*に書き込みます。

```
.EXPORT STORED PROG SCOTT.p1,SCOTT.f3 FILE pl1
```

LIST (プログラム単位) コマンド

説明

指定したプログラム単位のテキストを表示し、現行のソース位置を設定します。

構文

```
LIST { [USER schema] PROGRAMUNIT name | PROGRAMUNIT [schema.]name }
      { . | PC | SCOPE }
      [LINE number]
      [SPECIFICATION | BODY]
LIST { [USER schema] PACKAGE name | PACKAGE schema.name }
      { . | PC | SCOPE }
      [LINE number]
      [SPECIFICATION | BODY]
LIST { [USER schema] SUBPROGRAM name | SUBPROGRAM schema.name }
      { . | PC | SCOPE }
      [LINE number]
      [SPECIFICATION | BODY]
LIST { [USER schema] PROCEDURE name | PROCEDURE schema.name }
      { . | PC | SCOPE }
      [LINE number]
      [SPECIFICATION | BODY]
LIST { [USER schema] FUNCTION name | FUNCTION schema.name }
      { . | PC | SCOPE }
```

[LINE *number*]
 [SPECIFICATION | BODY]

キーワードおよび値

USER <i>schema</i>	ストアド・プログラム単位があるデータベース内のスキーマ名を指定します。
PROGRAMUNIT <i>name</i>	プログラム単位を指定します。
PACKAGE <i>name</i>	パッケージを指定します。
SUBPROGRAM <i>name</i>	サブプログラムを指定します。
PROCEDURE <i>name</i>	プロシージャを指定します。
FUNCTION <i>name</i>	ファンクションを指定します。
.	現行のソース位置を指定する。これがデフォルトです。
PC	現行の実行位置を指定します。
SCOPE	現行の有効範囲位置を指定します。
LINE <i>number</i>	現行のソース位置になるプログラム単位の行を指定します。
SPECIFICATIONまたはBODY	プログラム単位の仕様部と本体のどちらを表示するかを指定します。デフォルトはSPECIFICATIONです。

コメント

LISTは、インタプリタのソース・ペインにあるプログラム単位のテキストを表示します。LINEを使用して行を指定しない場合、プログラム単位の第1行が現行のソース位置になります。

この規則は、PCおよびSCOPEが指定された場合には適用されません。またはPCを指定すると、ソース位置が現行の実行位置に設定されます。SCOPEを指定すると、ソース位置は現行の有効範囲位置に設定されます。

注意: PCおよびSCOPEが有効なのは、プログラム実行が中断されている場合のみです。

LIST (プログラム単位) コマンドの例

次のコマンドは、プロシージャ*p1*のソース・テキストを表示し、ソース位置を1行目に設定します。

```
.LIST PROC p1
```

次のコマンドは、*p1*のソース・テキストを表示し、ソース位置を第18行に設定します。

```
.LIST PROGRAMUNIT p1 LINE 18
```

次のコマンドは、ソース位置を現行の実行位置に設定し、ソース・テキストを表示します。

```
.LIST PC
```

次のコマンドは、ユーザー*scott*が所有するスキーマから、サーバー側のプログラム単位*my_proc*のソース・テキストを表示し、現行のソース位置を保持します。

```
.LIST USER scott PROC my_proc
```

または

```
.LIST PROC scott.my_proc
```

LOAD (プログラム単位) コマンド

説明

ファイル・システムから1つ以上のプログラム単位をロードします。

構文

```
LOAD FILE [directory] name[extension]
        [, [directory] name...]
        [NOCONFIRM]
```

キーワードおよび値

FILE <i>name</i>	プログラム単位テキストを含む、1つ以上のファイルを指定します。
NOCONFIRM	既存のプログラム単位を、確認のプロンプトなしで再定義するように指定します。

コメント

指定する各ファイルには、1つのプログラム単位のソース・テキストが含まれていなければなりません。指定しない場合、デフォルト・ディレクトリは現行ディレクトリ、デフォルトのファイル拡張子は`.pls`です。

*directory*の構文はオペレーティング・システムに固有のものです。構文の詳細は、ご使用のオペレーティング・システム用のOracle製品マニュアルを参照してください。

ソース・テキストは、ロード時にコンパイルされます。結果としてできたプログラム単位が名前付きエンティティ（つまりサブプログラムまたはパッケージ）である場合、処理は完了します。プログラム単位が無名ブロックである場合、それは実行され、完了時に自動的に廃棄されます。

プログラム単位を既存のプログラム単位と同じ名前およびタイプでロードしようとする、メッセージ・ボックスが表示され、既存のプログラム単位を再定義するかどうかの確認を求められます。コマンド文字列にNOCONFIRMを指定すると、警告は抑止されます。

LOAD (プログラム単位) コマンドの例

次のコマンドは、ソースが現行のディレクトリ内のファイル`proc1.pls`および`func2.pls`に含まれるプログラム単位をロードします。

```
.LOAD FILE proc1, func2
```

LOAD (ストアド・プログラム単位) コマンド

説明

データベースに保存された、1つ以上のプログラム単位をロードします。

構文

```
LOAD STORED PROGRAMUNIT [owner] name[, [owner] name...]
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD STORED PACKAGE [owner] name[, [owner] name...]
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD STORED SUBPROGRAM [owner] name[, [owner] name...]
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD STORED PROCEDURE [owner] name[, [owner] name...]
    [SPECIFICATION | BODY]
    [NOCONFIRM]
LOAD STORED FUNCTION [owner] name[, [owner] name...]
    [SPECIFICATION | BODY]
    [NOCONFIRM]
```

キーワードおよび値

PROGRAMUNIT <i>name</i>	1つ以上のプログラム単位を指定します。オプションで所有者も指定できます。
PACKAGE <i>name</i>	1つ以上のパッケージを指定します。オプションで所有者も指定できません。
SUBPROGRAM <i>name</i>	1つ以上のサブプログラムを指定します。オプションで所有者も指定できます。
PROCEDURE <i>name</i>	1つ以上のプロシージャを指定します。オプションで所有者も指定できません。
FUNCTION <i>name</i>	1つ以上のファンクションを指定します。オプションで所有者も指定できます。
SPECIFICATIONまたはBODY	ストアド・プログラム単位の仕様部と本体のどちらをロードするかを指定します。どちらも指定しない場合、両方がロードされます。
NOCONFIRM	既存のプログラム単位を、確認のプロンプトなしで再定義するように指定します。

コメント

ソース・テキストは、ロード時にコンパイルされます。結果としてできたプログラム単位が名前付きエンティティ (つまりサブプログラムまたはパッケージ) である場合、処理は完了します。プログラム単位が無名ブロックである場合、それは実行され、完了時に自動的に廃棄されます。

プログラム単位を既存のプログラム単位と同じ名前およびタイプでロードしようとする、メッセージ・ボックスが表示され、既存のプログラム単位を再定義するかどうかの確認を求められます。コマンド文字列にNOCONFIRMを指定すると、警告は抑止されます。

LOAD (ストアド・プログラム単位) コマンドの例

次のコマンドは、ストアド・プロシージャ *proc1.pls* およびストアド・アクション *func2.pls* に含まれるプログラム単位をロードします。

```
.LOAD STORED PROG scott.proc1, scott.func2
```

SHOW (ローカル) コマンド

説明

現行の有効範囲内に定義されている、現行のローカル変数およびパラメータをリストします。

構文

```
SHOW LOCALS  
SHOW PARAMETERS  
SHOW VARIABLES
```

キーワードおよび値

LOCALS	すべてのパラメータおよび変数を指定します。
PARAMETERS	すべてのパラメータを指定します。
VARIABLES	すべての変数を指定します。

SHOW (ローカル) コマンドの例

次のコマンドは、すべての現行のパラメータの情報を表示します。

```
.SHOW PARAMETERS
```

SHOW (プログラム単位) コマンド

説明

現行セッションで、現在定義されているプログラム単位を表示します。

構文

```
SHOW PROGRAMUNITS
    [USER | BUILTIN]
    [SPECIFICATION | BODY]
SHOW PACKAGES
    [USER | BUILTIN]
    [SPECIFICATION | BODY]
SHOW SUBPROGRAMS
    [USER | BUILTIN]
    [SPECIFICATION | BODY]
SHOW PROCEDURES
    [USER | BUILTIN]
    [SPECIFICATION | BODY]
SHOW FUNCTIONS
    [USER | BUILTIN]
    [SPECIFICATION | BODY]
```

キーワードおよび値

PROGRAMUNITS	すべてのプログラム単位を指定します。
PACKAGES	すべてのパッケージを指定します。
SUBPROGRAMS	すべてのサブプログラムを指定します。
PROCEDURES	すべてのプロシージャを指定します。
FUNCTIONS	すべてのファンクションを指定します。
USERまたはBUILTIN	ユーザー定義のプログラム単位を表示するか、またはビルトイン・プログラム単位を表示するかを指定します。デフォルトはUSERです。
SPECIFICATIONまたはBODY	仕様部をリストするか、本体をリストするかを指定します。デフォルトでは、どちらもリストされます。

SHOW (プログラム単位) コマンドの例

次のコマンドは、すべての現行のユーザー定義のプログラム単位の名前とタイプをリストします。

```
.SHOW PROGRAMUNITS
```

次のコマンドは、すべてのビルトイン・パッケージ仕様部をリストします。

```
.SHOW PACK SPEC BUILT
```

セッション・コマンド

DESCRIBE (バージョン) コマンド

説明

Procedure BuilderとPL/SQLコンパイラの現行バージョンの詳細情報を表示します。

構文

```
DESCRIBE VERSION
```

DESCRIBE (バージョン) コマンドの例

次のコマンドは、Procedure Builderの詳細情報を表示します。

```
.DESCRIBE VER
```

HELPコマンド

説明

コマンドの説明と構文を表示します。

構文

```
HELP [COMMAND name] [SYNTAX]
```

キーワードおよび値

COMMAND <i>name</i>	Procedure Builderコマンドを指定します。
SYNTAX	指定したコマンドの構文を表示します。

コメント

コマンド名を指定しない場合、すべてのProcedure Builderコマンドのヘルプリストを表示します。

HELPコマンドの例

次のコマンドは、BREAKコマンドの簡単な説明と構文を表示します。

```
.HELP COM BREAK SYNTAX
```

次のコマンドは、すべてのProcedure Builderコマンドのヘルプリストを表示します。

```
.HELP
```

INTERPRETコマンド

説明

1つ以上のProcedure Builderスクリプトを実行します。

構文

```
INTERPRET FILE name [, name...]
      [ECHO]
      [SILENT]
      [NOCONFIRM]
```

キーワードおよび値

FILE <i>name</i>	Procedure Builderスクリプトを含むファイルを指定します。
ECHO	実行時、スクリプトの各行を表示します。
SILENT	インタプリタによって発行されたステータス・メッセージを非表示にします。
NOCONFIRM	既存のプログラム単位を、確認のプロンプトなしで再定義するように指定します。

コメント

Procedure Builderスクリプトは、プログラム単位、Procedure BuilderコマンドおよびSQL文の任意の組合せである一連の構成体から構成されます。スクリプトは、その内容がインタプリタに直接入力されたかのように処理されます。スクリプト内の各PL/SQL構成体は、LOADコマンドによって個別にロードされたかのように処理されます。

指定しない場合、ファイル拡張子のデフォルトは.pldです。

プログラム単位を既存のプログラム単位と同じ名前およびタイプでロードしようとする、メッセージ・ボックスが表示され、既存のプログラム単位を再定義するかどうかの確認を求められます。NOCONFIRMを指定すると、警告は表示されません。

スクリプトにSQL文を組み込むことはできますが、SQL*Plusの文と構文はサポートされていません。

INTERPRETを使用すると、1つのファイルから複数のプログラム単位をロードできます。しかし、Procedure Builderではソース・テキストを事前に解析し、プログラム単位をPL/SQLコンパイラに一度に1つずつ送らなければならないため、実際にはINTERPRETは同時に複数プログラムのLOADを実行することはできません。

INTERPRETコマンドの例

次のコマンドは、*script1*というファイルのスクリプトを解釈します（ECHO使用可能）。

```
.INTERPRET FILE script1 ECHO
```

QUITコマンド

説明

現行のProcedure Builderセッションを終了します。このコマンドは、Procedure Builderがスタンドアロン・セッションとして起動される場合にのみ有効です。

構文

```
QUIT [NOCONFIRM]
```

A

ATTACHコマンド, 40

B

BREAKコマンド, 22

C

CLOSEコマンド, 41

COMPILEコマンド, 41

 COMPILE (プログラム単位) コマンド, 66

 COMPILE (ライブラリ) コマンド, 41

CONNECTコマンド, 16

CREATEコマンド, 12

 CREATE (バインド変数) コマンド, 12

 CREATE (ライブラリ) コマンド, 42

D

DELETEコマンド, 13

 DELETE (デバッグ・アクション) コマンド, 24

 DELETE (バインド変数) コマンド, 13

 DELETE (プログラム単位) コマンド, 66

 DELETE (ライブラリ) コマンド, 43

 DELETE (ライブラリ・プログラム単位) コマ
 ンド, 44

 DELETE (ロード・パス) コマンド, 58

DESCRIBEコマンド, 25

DESCRIBE (デバッグ・アクション) コマンド,
25

DESCRIBE (バージョン) コマンド, 80

DESCRIBE (表およびビュー) コマンド, 17

DESCRIBE (プログラム単位) コマンド, 67

DESCRIBE (ライブラリ) コマンド, 45

DESCRIBE (ローカル) コマンド, 34

DESCRIBE (ロード・パス) コマンド, 58

DETACHコマンド, 45

DISABLEコマンド, 68

 DISABLE (コンパイラ・オプション) コマンド,
68

 DISABLE (デバッグ・アクション) コマンド, 26

 DISABLE (ロギング) コマンド, 62

DISCONNECTコマンド, 17

E

ENABLEコマンド, 69

 ENABLE (コンパイラ・オプション) コマンド,
69

 ENABLE (デバッグ・アクション) コマンド, 26

 ENABLE (ロギング) コマンド, 62

EXECUTEコマンド, 70

EXPORTコマンド, 46

 EXPORT (ストアド・プログラム単位) コマン
 ド, 72

 EXPORT (プログラム単位) コマンド, 70

 EXPORT (ライブラリ) コマンド, 46

G

GENERATEコマンド, 47

GOコマンド, 34

GRANT (データベース) コマンド, 18
GRANT (ライブラリ) コマンド, 48

H

HELPコマンド, 80

I

INSERTコマンド, 48
 INSERT (ライブラリ・プログラム単位) コマ
 ンド, 48
 INSERT (ロード・パス) コマンド, 58
INTERPRETコマンド, 81

L

LISTコマンド, 27
 LIST (デバッグ・アクション) コマンド, 27
 LIST (プログラム単位) コマンド, 73
LOADコマンド, 50
 LOAD (ストアド・プログラム単位) コマンド,
 76
 LOAD (プログラム単位) コマンド, 75
 LOAD (ライブラリ・プログラム単位) コマン
 ド, 50
LOGコマンド, 63

O

OPENコマンド, 51

P

PL/SQLインタプリタ・コマンド インタプリタ・コ
マンドを参照, 10

Q

QUITコマンド, 82

R

RENAME (ライブラリ) コマンド, 52
RESETコマンド, 35
REVERTコマンド, 52
REVOKE (データベース) コマンド, 18
REVOKE (ライブラリ) コマンド, 53

S

SAVEコマンド, 53
SET (有効範囲) コマンド, 35
SHOWコマンド, 37
 SHOW (コール・スタック) コマンド, 37
 SHOW (デバッグ・アクション) コマンド, 28
 SHOW (プログラム単位) コマンド, 78
 SHOW (ライブラリ) コマンド, 55
 SHOW (ローカル) コマンド, 77
STEPコマンド, 37
STOREコマンド, 19

T

TRIGGERコマンド, 28

い

インタプリタ・コマンド, 10
 使用, 2, 3, 4
 すべて, 10